

VOLUME 1



THE PRACTICE OF SYSTEM AND NETWORK ADMINISTRATION

THIRD EDITION



THOMAS A. LIMONCELLI • STRATA R. CHALUP • CHRISTINA J. HOGAN

VOLUME 1



THE PRACTICE OF SYSTEM AND NETWORK ADMINISTRATION

THIRD EDITION



THOMAS A. LIMONCELLI • STRATA R. CHALUP • CHRISTINA J. HOGAN

About This E-Book

EPUB is an open, industry-standard format for e-books. However, support for EPUB and its many features varies across reading devices and applications. Use your device or app settings to customize the presentation to your liking. Settings that you can customize often include font, font size, single or double column, landscape or portrait mode, and figures that you can click or tap to enlarge. For additional information about the settings and features on your reading device or app, visit the device manufacturer's Web site.

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the e-book in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a “Click here to view code image” link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

The Practice of System and Network Administration

Volume 1

**Third Edition Thomas A. Limoncelli
Christina J. Hogan
Strata R. Chalup** ♦♦Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam •
Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi •
Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Catalog Number: 2016946362

Copyright © 2017 Thomas A. Limoncelli, Christina J. Lear née Hogan, Virtual.NET Inc., Lumeta Corporation

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

Page 4 excerpt: “Noël,” Season 2 Episode 10. The West Wing. Directed by Thomas Schlamme. Teleplay by Aaron Sorkin. Story by Peter Parnell. Scene performed by John Spencer and Bradley Whitford. Original broadcast December 20, 2000. Warner Brothers Burbank Studios, Burbank, CA. Aaron Sorkin, John

Wells Production, Warner Brothers Television, NBC © 2000. Broadcast television.

[Chapter 26](#) photos © 2017 Christina J. Lear née Hogan.

ISBN-13: 978-0-321-91916-8

ISBN-10: 0-321-91916-5

Text printed in the United States of America.

1 16

Contents at a Glance

[Contents](#)

[Preface](#)

[Acknowledgments](#)

[About the Authors](#)

[Part I Game-Changing Strategies](#)

[Chapter 1 Climbing Out of the Hole](#)

[Chapter 2 The Small Batches Principle](#)

[Chapter 3 Pets and Cattle](#)

[Chapter 4 Infrastructure as Code](#)

[Part II Workstation Fleet Management](#)

[Chapter 5 Workstation Architecture](#)

[Chapter 6 Workstation Hardware Strategies](#)

[Chapter 7 Workstation Software Life Cycle](#)

[Chapter 8 OS Installation Strategies](#)

[Chapter 9 Workstation Service Definition](#)

[Chapter 10 Workstation Fleet Logistics](#)

[Chapter 11 Workstation Standardization](#)

[Chapter 12 Onboarding](#)

[Part III Servers](#)

[Chapter 13 Server Hardware Strategies](#)

[Chapter 14 Server Hardware Features](#)

[Chapter 15 Server Hardware Specifications](#)

[Part IV Services](#)

[Chapter 16 Service Requirements](#)

[Chapter 17 Service Planning and Engineering](#)

[Chapter 18 Service Resiliency and Performance Patterns](#)

[Chapter 19 Service Launch: Fundamentals](#)

[Chapter 20 Service Launch: DevOps](#)

[Chapter 21 Service Conversions](#)

[Chapter 22 Disaster Recovery and Data Integrity](#)

[Part V Infrastructure](#)

[Chapter 23 Network Architecture](#)

[Chapter 24 Network Operations](#)

[Chapter 25 Datacenters Overview](#)

[Chapter 26 Running a Datacenter](#)

[Part VI Helpdesks and Support](#)

[Chapter 27 Customer Support](#)

[Chapter 28 Handling an Incident Report](#)

[Chapter 29 Debugging](#)

[Chapter 30 Fixing Things Once](#)

[Chapter 31 Documentation](#)

[Part VII Change Processes](#)

[Chapter 32 Change Management](#)

[Chapter 33 Server Upgrades](#)

[Chapter 34 Maintenance Windows](#)

[Chapter 35 Centralization Overview](#)

[Chapter 36 Centralization Recommendations](#)

[Chapter 37 Centralizing a Service](#)

Part VIII Service Recommendations

Chapter 38 Service Monitoring

Chapter 39 Namespaces

Chapter 40 Nameservices

Chapter 41 Email Service

Chapter 42 Print Service

Chapter 43 Data Storage

Chapter 44 Backup and Restore

Chapter 45 Software Repositories

Chapter 46 Web Services

Part IX Management Practices

Chapter 47 Ethics

Chapter 48 Organizational Structures

Chapter 49 Perception and Visibility

Chapter 50 Time Management

Chapter 51 Communication and Negotiation

Chapter 52 Being a Happy SA

Chapter 53 Hiring System Administrators

Chapter 54 Firing System Administrators

Part X Being More Awesome

Chapter 55 Operational Excellence

Chapter 56 Operational Assessments

Epilogue

Part XI Appendices

Appendix A What to Do When . . .

[Appendix B The Many Roles of a System Administrator](#)

[Bibliography](#)

[Index](#)

Contents

[*Preface*](#)

[*Acknowledgments*](#)

[*About the Authors*](#)

[**Part I Game-Changing Strategies**](#)

[**1 Climbing Out of the Hole**](#)

[1.1 Organizing WIP](#)

[1.1.1 Ticket Systems](#)

[1.1.2 Kanban](#)

[1.1.3 Tickets and Kanban](#)

[1.2 Eliminating Time Sinkholes](#)

[1.2.1 OS Installation and Configuration](#)

[1.2.2 Software Deployment](#)

[1.3 DevOps](#)

[1.4 DevOps Without Devs](#)

[1.5 Bottlenecks](#)

[1.6 Getting Started](#)

[1.7 Summary](#)

[Exercises](#)

[**2 The Small Batches Principle**](#)

[2.1 The Carpenter Analogy](#)

[2.2 Fixing Hell Month](#)

[2.3 Improving Emergency Failovers](#)

[2.4 Launching Early and Often](#)

[2.5 Summary](#)

[Exercises](#)

[**3 Pets and Cattle**](#)

[3.1 The Pets and Cattle Analogy](#)

[3.2 Scaling](#)

[3.3 Desktops as Cattle](#)

[3.4 Server Hardware as Cattle](#)

[3.5 Pets Store State](#)

[3.6 Isolating State](#)

[3.7 Generic Processes](#)

[3.8 Moving Variations to the End](#)

[3.9 Automation](#)

[3.10 Summary](#)

[Exercises](#)

[4 Infrastructure as Code](#)

[4.1 Programmable Infrastructure](#)

[4.2 Tracking Changes](#)

[4.3 Benefits of Infrastructure as Code](#)

[4.4 Principles of Infrastructure as Code](#)

[4.5 Configuration Management Tools](#)

[4.5.1 Declarative Versus Imperative](#)

[4.5.2 Idempotency](#)

[4.5.3 Guards and Statements](#)

[4.6 Example Infrastructure as Code Systems](#)

[4.6.1 Configuring a DNS Client](#)

[4.6.2 A Simple Web Server](#)

[4.6.3 A Complex Web Application](#)

[4.7 Bringing Infrastructure as Code to Your Organization](#)

[4.8 Infrastructure as Code for Enhanced Collaboration](#)

[4.9 Downsides to Infrastructure as Code](#)

[4.10 Automation Myths](#)

[4.11 Summary](#)

[Exercises](#)

Part II Workstation Fleet Management

5 Workstation Architecture

[5.1 Fungibility](#)

[5.2 Hardware](#)

[5.3 Operating System](#)

[5.4 Network Configuration](#)

[5.4.1 Dynamic Configuration](#)

[5.4.2 Hardcoded Configuration](#)

[5.4.3 Hybrid Configuration](#)

[5.4.4 Applicability](#)

[5.5 Accounts and Authorization](#)

[5.6 Data Storage](#)

[5.7 OS Updates](#)

[5.8 Security](#)

[5.8.1 Theft](#)

[5.8.2 Malware](#)

[5.9 Logging](#)

[5.10 Summary](#)

[Exercises](#)

6 Workstation Hardware Strategies

[6.1 Physical Workstations](#)

[6.1.1 Laptop Versus Desktop](#)

[6.1.2 Vendor Selection](#)

[6.1.3 Product Line Selection](#)

[6.2 Virtual Desktop Infrastructure](#)

[6.2.1 Reduced Costs](#)

[6.2.2 Ease of Maintenance](#)

[6.2.3 Persistent or Non-persistent?](#)

[6.3 Bring Your Own Device](#)

[6.3.1 Strategies](#)

[6.3.2 Pros and Cons](#)

[6.3.3 Security](#)

[6.3.4 Additional Costs](#)

[6.3.5 Usability](#)

[6.4 Summary](#)

[Exercises](#)

7 Workstation Software Life Cycle

[7.1 Life of a Machine](#)

[7.2 OS Installation](#)

[7.3 OS Configuration](#)

[7.3.1 Configuration Management Systems](#)

[7.3.2 Microsoft Group Policy Objects](#)

[7.3.3 DHCP Configuration](#)

[7.3.4 Package Installation](#)

[7.4 Updating the System Software and Applications](#)

[7.4.1 Updates Versus Installations](#)

[7.4.2 Update Methods](#)

[7.5 Rolling Out Changes . . . Carefully](#)

[7.6 Disposal](#)

[7.6.1 Accounting](#)

[7.6.2 Technical: Decommissioning](#)

[7.6.3 Technical: Data Security](#)

[7.6.4 Physical](#)

[7.7 Summary](#)

[Exercises](#)

8 OS Installation Strategies

[8.1 Consistency Is More Important Than Perfection](#)

[8.2 Installation Strategies](#)

[8.2.1 Automation](#)

[8.2.2 Cloning](#)

8.2.3 Manual

8.3 Test-Driven Configuration Development

8.4 Automating in Steps

8.5 When Not to Automate

8.6 Vendor Support of OS Installation

8.7 Should You Trust the Vendor's Installation?

8.8 Summary

Exercises

9 Workstation Service Definition

9.1 Basic Service Definition

9.1.1 Approaches to Platform Definition

9.1.2 Application Selection

9.1.3 Leveraging a CMDB

9.2 Refresh Cycles

9.2.1 Choosing an Approach

9.2.2 Formalizing the Policy

9.2.3 Aligning with Asset Depreciation

9.3 Tiered Support Levels

9.4 Workstations as a Managed Service

9.5 Summary

Exercises

10 Workstation Fleet Logistics

10.1 What Employees See

10.2 What Employees Don't See

10.2.1 Purchasing Team

10.2.2 Prep Team

10.2.3 Delivery Team

10.2.4 Platform Team

10.2.5 Network Team

10.2.6 Tools Team

[10.2.7 Project Management](#)

[10.2.8 Program Office](#)

[10.3 Configuration Management Database](#)

[10.4 Small-Scale Fleet Logistics](#)

[10.4.1 Part-Time Fleet Management](#)

[10.4.2 Full-Time Fleet Coordinators](#)

[10.5 Summary](#)

[Exercises](#)

[11 Workstation Standardization](#)

[11.1 Involving Customers Early](#)

[11.2 Releasing Early and Iterating](#)

[11.3 Having a Transition Interval \(Overlap\)](#)

[11.4 Ratcheting](#)

[11.5 Setting a Cut-Off Date](#)

[11.6 Adapting for Your Corporate Culture](#)

[11.7 Leveraging the Path of Least Resistance](#)

[11.8 Summary](#)

[Exercises](#)

[12 Onboarding](#)

[12.1 Making a Good First Impression](#)

[12.2 IT Responsibilities](#)

[12.3 Five Keys to Successful Onboarding](#)

[12.3.1 Drive the Process with an Onboarding Timeline](#)

[12.3.2 Determine Needs Ahead of Arrival](#)

[12.3.3 Perform the Onboarding](#)

[12.3.4 Communicate Across Teams](#)

[12.3.5 Reflect On and Improve the Process](#)

[12.4 Cadence Changes](#)

[12.5 Case Studies](#)

[12.5.1 Worst Onboarding Experience Ever](#)

[12.5.2 Lumeta's Onboarding Process](#)

[12.5.3 Google's Onboarding Process](#)

[12.6 Summary](#)

[Exercises](#)

Part III Servers

13 Server Hardware Strategies

[13.1 All Eggs in One Basket](#)

[13.2 Beautiful Snowflakes](#)

[13.2.1 Asset Tracking](#)

[13.2.2 Reducing Variations](#)

[13.2.3 Global Optimization](#)

[13.3 Buy in Bulk, Allocate Fractions](#)

[13.3.1 VM Management](#)

[13.3.2 Live Migration](#)

[13.3.3 VM Packing](#)

[13.3.4 Spare Capacity for Maintenance](#)

[13.3.5 Unified VM/Non-VM Management](#)

[13.3.6 Containers](#)

[13.4 Grid Computing](#)

[13.5 Blade Servers](#)

[13.6 Cloud-Based Compute Services](#)

[13.6.1 What Is the Cloud?](#)

[13.6.2 Cloud Computing's Cost Benefits](#)

[13.6.3 Software as a Service](#)

[13.7 Server Appliances](#)

[13.8 Hybrid Strategies](#)

[13.9 Summary](#)

[Exercises](#)

14 Server Hardware Features

[14.1 Workstations Versus Servers](#)

[14.1.1 Server Hardware Design Differences](#)

[14.1.2 Server OS and Management Differences](#)

[14.2 Server Reliability](#)

[14.2.1 Levels of Redundancy](#)

[14.2.2 Data Integrity](#)

[14.2.3 Hot-Swap Components](#)

[14.2.4 Servers Should Be in Computer Rooms](#)

[14.3 Remotely Managing Servers](#)

[14.3.1 Integrated Out-of-Band Management](#)

[14.3.2 Non-integrated Out-of-Band Management](#)

[14.4 Separate Administrative Networks](#)

[14.5 Maintenance Contracts and Spare Parts](#)

[14.5.1 Vendor SLA](#)

[14.5.2 Spare Parts](#)

[14.5.3 Tracking Service Contracts](#)

[14.5.4 Cross-Shipping](#)

[14.6 Selecting Vendors with Server Experience](#)

[14.7 Summary](#)

[Exercises](#)

[15 Server Hardware Specifications](#)

[15.1 Models and Product Lines](#)

[15.2 Server Hardware Details](#)

[15.2.1 CPUs](#)

[15.2.2 Memory](#)

[15.2.3 Network Interfaces](#)

[15.2.4 Disks: Hardware Versus Software RAID](#)

[15.2.5 Power Supplies](#)

[15.3 Things to Leave Out](#)

[15.4 Summary](#)

[Exercises](#)

Part IV Services

16 Service Requirements

[16.1 Services Make the Environment](#)

[16.2 Starting with a Kick-Off Meeting](#)

[16.3 Gathering Written Requirements](#)

[16.4 Customer Requirements](#)

[16.4.1 Describing Features](#)

[16.4.2 Questions to Ask](#)

[16.4.3 Service Level Agreements](#)

[16.4.4 Handling Difficult Requests](#)

[16.5 Scope, Schedule, and Resources](#)

[16.6 Operational Requirements](#)

[16.6.1 System Observability](#)

[16.6.2 Remote and Central Management](#)

[16.6.3 Scaling Up or Out](#)

[16.6.4 Software Upgrades](#)

[16.6.5 Environment Fit](#)

[16.6.6 Support Model](#)

[16.6.7 Service Requests](#)

[16.6.8 Disaster Recovery](#)

[16.7 Open Architecture](#)

[16.8 Summary](#)

[Exercises](#)

17 Service Planning and Engineering

[17.1 General Engineering Basics](#)

[17.2 Simplicity](#)

[17.3 Vendor-Certified Designs](#)

[17.4 Dependency Engineering](#)

[17.4.1 Primary Dependencies](#)

[17.4.2 External Dependencies](#)

[17.4.3 Dependency Alignment](#)

[17.5 Decoupling Hostname from Service Name](#)

[17.6 Support](#)

[17.6.1 Monitoring](#)

[17.6.2 Support Model](#)

[17.6.3 Service Request Model](#)

[17.6.4 Documentation](#)

[17.7 Summary](#)

[Exercises](#)

[18 Service Resiliency and Performance Patterns](#)

[18.1 Redundancy Design Patterns](#)

[18.1.1 Masters and Slaves](#)

[18.1.2 Load Balancers Plus Replicas](#)

[18.1.3 Replicas and Shared State](#)

[18.1.4 Performance or Resilience?](#)

[18.2 Performance and Scaling](#)

[18.2.1 Dataflow Analysis for Scaling](#)

[18.2.2 Bandwidth Versus Latency](#)

[18.3 Summary](#)

[Exercises](#)

[19 Service Launch: Fundamentals](#)

[19.1 Planning for Problems](#)

[19.2 The Six-Step Launch Process](#)

[19.2.1 Step 1: Define the Ready List](#)

[19.2.2 Step 2: Work the List](#)

[19.2.3 Step 3: Launch the Beta Service](#)

[19.2.4 Step 4: Launch the Production Service](#)

[19.2.5 Step 5: Capture the Lessons Learned](#)

[19.2.6 Step 6: Repeat](#)

[19.3 Launch Readiness Review](#)

[19.3.1 Launch Readiness Criteria](#)

[19.3.2 Sample Launch Criteria](#)

[19.3.3 Organizational Learning](#)

[19.3.4 LRC Maintenance](#)

[19.4 Launch Calendar](#)

[19.5 Common Launch Problems](#)

[19.5.1 Processes Fail in Production](#)

[19.5.2 Unexpected Access Methods](#)

[19.5.3 Production Resources Unavailable](#)

[19.5.4 New Technology Failures](#)

[19.5.5 Lack of User Training](#)

[19.5.6 No Backups](#)

[19.6 Summary](#)

[Exercises](#)

[20 Service Launch: DevOps](#)

[20.1 Continuous Integration and Deployment](#)

[20.1.1 Test Ordering](#)

[20.1.2 Launch Categorizations](#)

[20.2 Minimum Viable Product](#)

[20.3 Rapid Release with Packaged Software](#)

[20.3.1 Testing Before Deployment](#)

[20.3.2 Time to Deployment Metrics](#)

[20.4 Cloning the Production Environment](#)

[20.5 Example: DNS/DHCP Infrastructure Software](#)

[20.5.1 The Problem](#)

[20.5.2 Desired End-State](#)

[20.5.3 First Milestone](#)

[20.5.4 Second Milestone](#)

[20.6 Launch with Data Migration](#)

[20.7 Controlling Self-Updating Software](#)

[20.8 Summary](#)

[Exercises](#)

[21 Service Conversions](#)

[21.1 Minimizing Intrusiveness](#)

[21.2 Layers Versus Pillars](#)

[21.3 Vendor Support](#)

[21.4 Communication](#)

[21.5 Training](#)

[21.6 Gradual Roll-Outs](#)

[21.7 Flash-Cuts: Doing It All at Once](#)

[21.8 Backout Plan](#)

[21.8.1 Instant Roll-Back](#)

[21.8.2 Decision Point](#)

[21.9 Summary](#)

[Exercises](#)

[22 Disaster Recovery and Data Integrity](#)

[22.1 Risk Analysis](#)

[22.2 Legal Obligations](#)

[22.3 Damage Limitation](#)

[22.4 Preparation](#)

[22.5 Data Integrity](#)

[22.6 Redundant Sites](#)

[22.7 Security Disasters](#)

[22.8 Media Relations](#)

[22.9 Summary](#)

[Exercises](#)

[Part V Infrastructure](#)

[23 Network Architecture](#)

[23.1 Physical Versus Logical](#)

[23.2 The OSI Model](#)

[23.3 Wired Office Networks](#)

[23.3.1 Physical Infrastructure](#)

[23.3.2 Logical Design](#)

[23.3.3 Network Access Control](#)

[23.3.4 Location for Emergency Services](#)

[23.4 Wireless Office Networks](#)

[23.4.1 Physical Infrastructure](#)

[23.4.2 Logical Design](#)

[23.5 Datacenter Networks](#)

[23.5.1 Physical Infrastructure](#)

[23.5.2 Logical Design](#)

[23.6 WAN Strategies](#)

[23.6.1 Topology](#)

[23.6.2 Technology](#)

[23.7 Routing](#)

[23.7.1 Static Routing](#)

[23.7.2 Interior Routing Protocol](#)

[23.7.3 Exterior Gateway Protocol](#)

[23.8 Internet Access](#)

[23.8.1 Outbound Connectivity](#)

[23.8.2 Inbound Connectivity](#)

[23.9 Corporate Standards](#)

[23.9.1 Logical Design](#)

[23.9.2 Physical Design](#)

[23.10 Software-Defined Networks](#)

[23.11 IPv6](#)

[23.11.1 The Need for IPv6](#)

[23.11.2 Deploying IPv6](#)

[23.12 Summary](#)

[Exercises](#)

[**24 Network Operations**](#)

[24.1 Monitoring](#)

[24.2 Management](#)

[24.2.1 Access and Audit Trail](#)

[24.2.2 Life Cycle](#)

[24.2.3 Configuration Management](#)

[24.2.4 Software Versions](#)

[24.2.5 Deployment Process](#)

[24.3 Documentation](#)

[24.3.1 Network Design and Implementation](#)

[24.3.2 DNS](#)

[24.3.3 CMDB](#)

[24.3.4 Labeling](#)

[24.4 Support](#)

[24.4.1 Tools](#)

[24.4.2 Organizational Structure](#)

[24.4.3 Network Services](#)

[24.5 Summary](#)

[Exercises](#)

[25 Datacenters Overview](#)

[25.1 Build, Rent, or Outsource](#)

[25.1.1 Building](#)

[25.1.2 Renting](#)

[25.1.3 Outsourcing](#)

[25.1.4 No Datacenter](#)

[25.1.5 Hybrid](#)

[25.2 Requirements](#)

[25.2.1 Business Requirements](#)

[25.2.2 Technical Requirements](#)

[25.3 Summary](#)

[Exercises](#)

26 Running a Datacenter

26.1 Capacity Management

26.1.1 Rack Space

26.1.2 Power

26.1.3 Wiring

26.1.4 Network and Console

26.2 Life-Cycle Management

26.2.1 Installation

26.2.2 Moves, Adds, and Changes

26.2.3 Maintenance

26.2.4 Decommission

26.3 Patch Cables

26.4 Labeling

26.4.1 Labeling Rack Location

26.4.2 Labeling Patch Cables

26.4.3 Labeling Network Equipment

26.5 Console Access

26.6 Workbench

26.7 Tools and Supplies

26.7.1 Tools

26.7.2 Spares and Supplies

26.7.3 Parking Spaces

26.8 Summary

Exercises

Part VI Helpdesks and Support

27 Customer Support

27.1 Having a Helpdesk

27.2 Offering a Friendly Face

27.3 Reflecting Corporate Culture

27.4 Having Enough Staff

[27.5 Defining Scope of Support](#)
[27.6 Specifying How to Get Help](#)
[27.7 Defining Processes for Staff](#)
[27.8 Establishing an Escalation Process](#)
[27.9 Defining “Emergency” in Writing](#)
[27.10 Supplying Request-Tracking Software](#)
[27.11 Statistical Improvements](#)
[27.12 After-Hours and 24/7 Coverage](#)
[27.13 Better Advertising for the Helpdesk](#)
[27.14 Different Helpdesks for Different Needs](#)
[27.15 Summary](#)
[Exercises](#)

28 Handling an Incident Report

[28.1 Process Overview](#)
[28.2 Phase A—Step 1: The Greeting](#)
[28.3 Phase B: Problem Identification](#)
 [28.3.1 Step 2: Problem Classification](#)
 [28.3.2 Step 3: Problem Statement](#)
 [28.3.3 Step 4: Problem Verification](#)
[28.4 Phase C: Planning and Execution](#)
 [28.4.1 Step 5: Solution Proposals](#)
 [28.4.2 Step 6: Solution Selection](#)
 [28.4.3 Step 7: Execution](#)
[28.5 Phase D: Verification](#)
 [28.5.1 Step 8: Craft Verification](#)
 [28.5.2 Step 9: Customer Verification/Closing](#)
[28.6 Perils of Skipping a Step](#)
[28.7 Optimizing Customer Care](#)
 [28.7.1 Model-Based Training](#)
 [28.7.2 Holistic Improvement](#)

- [28.7.3 Increased Customer Familiarity](#)
- [28.7.4 Special Announcements for Major Outages](#)
- [28.7.5 Trend Analysis](#)
- [28.7.6 Customers Who Know the Process](#)
- [28.7.7 An Architecture That Reflects the Process](#)

[28.8 Summary](#)

[Exercises](#)

29 Debugging

- [29.1 Understanding the Customer's Problem](#)
- [29.2 Fixing the Cause, Not the Symptom](#)
- [29.3 Being Systematic](#)
- [29.4 Having the Right Tools](#)
 - [29.4.1 Training Is the Most Important Tool](#)
 - [29.4.2 Understanding the Underlying Technology](#)
 - [29.4.3 Choosing the Right Tools](#)
 - [29.4.4 Evaluating Tools](#)

[29.5 End-to-End Understanding of the System](#)

[29.6 Summary](#)

[Exercises](#)

30 Fixing Things Once

- [30.1 Story: The Misconfigured Servers](#)
- [30.2 Avoiding Temporary Fixes](#)
- [30.3 Learn from Carpenters](#)
- [30.4 Automation](#)
- [30.5 Summary](#)

[Exercises](#)

31 Documentation

- [31.1 What to Document](#)
- [31.2 A Simple Template for Getting Started](#)
- [31.3 Easy Sources for Documentation](#)

- [31.3.1 Saving Screenshots](#)
- [31.3.2 Capturing the Command Line](#)
- [31.3.3 Leveraging Email](#)
- [31.3.4 Mining the Ticket System](#)
- [31.4 The Power of Checklists](#)
- [31.5 Wiki Systems](#)
- [31.6 Findability](#)
- [31.7 Roll-Out Issues](#)
- [31.8 A Content-Management System](#)
- [31.9 A Culture of Respect](#)
- [31.10 Taxonomy and Structure](#)
- [31.11 Additional Documentation Uses](#)
- [31.12 Off-Site Links](#)
- [31.13 Summary](#)
- [Exercises](#)

Part VII Change Processes

32 Change Management

- [32.1 Change Review Boards](#)
- [32.2 Process Overview](#)
- [32.3 Change Proposals](#)
- [32.4 Change Classifications](#)
- [32.5 Risk Discovery and Quantification](#)
- [32.6 Technical Planning](#)
- [32.7 Scheduling](#)
- [32.8 Communication](#)
- [32.9 Tiered Change Review Boards](#)
- [32.10 Change Freezes](#)
- [32.11 Team Change Management](#)
 - [32.11.1 Changes Before Weekends](#)
 - [32.11.2 Preventing Injured Toes](#)

[32.11.3 Revision History](#)

[32.12 Starting with Git](#)

[32.13 Summary](#)

[Exercises](#)

[33 Server Upgrades](#)

[33.1 The Upgrade Process](#)

[33.2 Step 1: Develop a Service Checklist](#)

[33.3 Step 2: Verify Software Compatibility](#)

[33.3.1 Upgrade the Software Before the OS](#)

[33.3.2 Upgrade the Software After the OS](#)

[33.3.3 Postpone the Upgrade or Change the Software](#)

[33.4 Step 3: Develop Verification Tests](#)

[33.5 Step 4: Choose an Upgrade Strategy](#)

[33.5.1 Speed](#)

[33.5.2 Risk](#)

[33.5.3 End-User Disruption](#)

[33.5.4 Effort](#)

[33.6 Step 5: Write a Detailed Implementation Plan](#)

[33.6.1 Adding Services During the Upgrade](#)

[33.6.2 Removing Services During the Upgrade](#)

[33.6.3 Old and New Versions on the Same Machine](#)

[33.6.4 Performing a Dress Rehearsal](#)

[33.7 Step 6: Write a Backout Plan](#)

[33.8 Step 7: Select a Maintenance Window](#)

[33.9 Step 8: Announce the Upgrade](#)

[33.10 Step 9: Execute the Tests](#)

[33.11 Step 10: Lock Out Customers](#)

[33.12 Step 11: Do the Upgrade with Someone](#)

[33.13 Step 12: Test Your Work](#)

[33.14 Step 13: If All Else Fails, Back Out](#)

[33.15 Step 14: Restore Access to Customers](#)

[33.16 Step 15: Communicate Completion/Backout](#)

[33.17 Summary](#)

[Exercises](#)

[34 Maintenance Windows](#)

[34.1 Process Overview](#)

[34.2 Getting Management Buy-In](#)

[34.3 Scheduling Maintenance Windows](#)

[34.4 Planning Maintenance Tasks](#)

[34.5 Selecting a Flight Director](#)

[34.6 Managing Change Proposals](#)

[34.6.1 Sample Change Proposal: SecurID Server Upgrade](#)

[34.6.2 Sample Change Proposal: Storage Migration](#)

[34.7 Developing the Master Plan](#)

[34.8 Disabling Access](#)

[34.9 Ensuring Mechanics and Coordination](#)

[34.9.1 Shutdown/Boot Sequence](#)

[34.9.2 KVM, Console Service, and LOM](#)

[34.9.3 Communications](#)

[34.10 Change Completion Deadlines](#)

[34.11 Comprehensive System Testing](#)

[34.12 Post-maintenance Communication](#)

[34.13 Reenabling Remote Access](#)

[34.14 Be Visible the Next Morning](#)

[34.15 Postmortem](#)

[34.16 Mentoring a New Flight Director](#)

[34.17 Trending of Historical Data](#)

[34.18 Providing Limited Availability](#)

[34.19 High-Availability Sites](#)

[34.19.1 The Similarities](#)

[34.19.2 The Differences](#)

[34.20 Summary](#)

[Exercises](#)

[35 Centralization Overview](#)

[35.1 Rationale for Reorganizing](#)

[35.1.1 Rationale for Centralization](#)

[35.1.2 Rationale for Decentralization](#)

[35.2 Approaches and Hybrids](#)

[35.3 Summary](#)

[Exercises](#)

[36 Centralization Recommendations](#)

[36.1 Architecture](#)

[36.2 Security](#)

[36.2.1 Authorization](#)

[36.2.2 Extranet Connections](#)

[36.2.3 Data Leakage Prevention](#)

[36.3 Infrastructure](#)

[36.3.1 Datacenters](#)

[36.3.2 Networking](#)

[36.3.3 IP Address Space Management](#)

[36.3.4 Namespace Management](#)

[36.3.5 Communications](#)

[36.3.6 Data Management](#)

[36.3.7 Monitoring](#)

[36.3.8 Logging](#)

[36.4 Support](#)

[36.4.1 Helpdesk](#)

[36.4.2 End-User Support](#)

[36.5 Purchasing](#)

[36.6 Lab Environments](#)

[36.7 Summary](#)

[Exercises](#)

[37 Centralizing a Service](#)

[37.1 Understand the Current Solution](#)

[37.2 Make a Detailed Plan](#)

[37.3 Get Management Support](#)

[37.4 Fix the Problems](#)

[37.5 Provide an Excellent Service](#)

[37.6 Start Slowly](#)

[37.7 Look for Low-Hanging Fruit](#)

[37.8 When to Decentralize](#)

[37.9 Managing Decentralized Services](#)

[37.10 Summary](#)

[Exercises](#)

[Part VIII Service Recommendations](#)

[38 Service Monitoring](#)

[38.1 Types of Monitoring](#)

[38.2 Building a Monitoring System](#)

[38.3 Historical Monitoring](#)

[38.3.1 Gathering the Data](#)

[38.3.2 Storing the Data](#)

[38.3.3 Viewing the Data](#)

[38.4 Real-Time Monitoring](#)

[38.4.1 SNMP](#)

[38.4.2 Log Processing](#)

[38.4.3 Alerting Mechanism](#)

[38.4.4 Escalation](#)

[38.4.5 Active Monitoring Systems](#)

[38.5 Scaling](#)

[38.5.1 Prioritization](#)

[38.5.2 Cascading Alerts](#)

[38.5.3 Coordination](#)

[38.6 Centralization and Accessibility](#)

[38.7 Pervasive Monitoring](#)

[38.8 End-to-End Tests](#)

[38.9 Application Response Time Monitoring](#)

[38.10 Compliance Monitoring](#)

[38.11 Meta-monitoring](#)

[38.12 Summary](#)

[Exercises](#)

[39 Namespaces](#)

[39.1 What Is a Namespace?](#)

[39.2 Basic Rules of Namespaces](#)

[39.3 Defining Names](#)

[39.4 Merging Namespaces](#)

[39.5 Life-Cycle Management](#)

[39.6 Reuse](#)

[39.7 Usage](#)

[39.7.1 Scope](#)

[39.7.2 Consistency](#)

[39.7.3 Authority](#)

[39.8 Federated Identity](#)

[39.9 Summary](#)

[Exercises](#)

[40 Nameservices](#)

[40.1 Nameservice Data](#)

[40.1.1 Data](#)

[40.1.2 Consistency](#)

[40.1.3 Authority](#)

[40.1.4 Capacity and Scaling](#)

[40.2 Reliability](#)

[40.2.1 DNS](#)

[40.2.2 DHCP](#)

[40.2.3 LDAP](#)

[40.2.4 Authentication](#)

[40.2.5 Authentication, Authorization, and Accounting](#)

[40.2.6 Databases](#)

[40.3 Access Policy](#)

[40.4 Change Policies](#)

[40.5 Change Procedures](#)

[40.5.1 Automation](#)

[40.5.2 Self-Service Automation](#)

[40.6 Centralized Management](#)

[40.7 Summary](#)

[Exercises](#)

[41 Email Service](#)

[41.1 Privacy Policy](#)

[41.2 Namespaces](#)

[41.3 Reliability](#)

[41.4 Simplicity](#)

[41.5 Spam and Virus Blocking](#)

[41.6 Generality](#)

[41.7 Automation](#)

[41.8 Monitoring](#)

[41.9 Redundancy](#)

[41.10 Scaling](#)

[41.11 Security Issues](#)

[41.12 Encryption](#)

[41.13 Email Retention Policy](#)

[41.14 Communication](#)

[41.15 High-Volume List Processing](#)

[41.16 Summary](#)

[Exercises](#)

[42 Print Service](#)

[42.1 Level of Centralization](#)

[42.2 Print Architecture Policy](#)

[42.3 Documentation](#)

[42.4 Monitoring](#)

[42.5 Environmental Issues](#)

[42.6 Shredding](#)

[42.7 Summary](#)

[Exercises](#)

[43 Data Storage](#)

[43.1 Terminology](#)

[43.1.1 Key Individual Disk Components](#)

[43.1.2 RAID](#)

[43.1.3 Volumes and File Systems](#)

[43.1.4 Directly Attached Storage](#)

[43.1.5 Network-Attached Storage](#)

[43.1.6 Storage-Area Networks](#)

[43.2 Managing Storage](#)

[43.2.1 Reframing Storage as a Community Resource](#)

[43.2.2 Conducting a Storage-Needs Assessment](#)

[43.2.3 Mapping Groups onto Storage Infrastructure](#)

[43.2.4 Developing an Inventory and Spares Policy](#)

[43.2.5 Planning for Future Storage](#)

[43.2.6 Establishing Storage Standards](#)

[43.3 Storage as a Service](#)

[43.3.1 A Storage SLA](#)

[43.3.2 Reliability](#)

[43.3.3 Backups](#)

[43.3.4 Monitoring](#)

[43.3.5 SAN Caveats](#)

[43.4 Performance](#)

[43.4.1 RAID and Performance](#)

[43.4.2 NAS and Performance](#)

[43.4.3 SSDs and Performance](#)

[43.4.4 SANs and Performance](#)

[43.4.5 Pipeline Optimization](#)

[43.5 Evaluating New Storage Solutions](#)

[43.5.1 Drive Speed](#)

[43.5.2 Fragmentation](#)

[43.5.3 Storage Limits: Disk Access Density Gap](#)

[43.5.4 Continuous Data Protection](#)

[43.6 Common Data Storage Problems](#)

[43.6.1 Large Physical Infrastructure](#)

[43.6.2 Timeouts](#)

[43.6.3 Saturation Behavior](#)

[43.7 Summary](#)

[Exercises](#)

[44 Backup and Restore](#)

[44.1 Getting Started](#)

[44.2 Reasons for Restores](#)

[44.2.1 Accidental File Deletion](#)

[44.2.2 Disk Failure](#)

[44.2.3 Archival Purposes](#)

[44.2.4 Perform Fire Drills](#)

[44.3 Corporate Guidelines](#)

[44.4 A Data-Recovery SLA and Policy](#)

[44.5 The Backup Schedule](#)

[44.6 Time and Capacity Planning](#)

[44.6.1 Backup Speed](#)

[44.6.2 Restore Speed](#)

[44.6.3 High-Availability Databases](#)

[44.7 Consumables Planning](#)

[44.7.1 Tape Inventory](#)

[44.7.2 Backup Media and Off-Site Storage](#)

[44.8 Restore-Process Issues](#)

[44.9 Backup Automation](#)

[44.10 Centralization](#)

[44.11 Technology Changes](#)

[44.12 Summary](#)

[Exercises](#)

[45 Software Repositories](#)

[45.1 Types of Repositories](#)

[45.2 Benefits of Repositories](#)

[45.3 Package Management Systems](#)

[45.4 Anatomy of a Package](#)

[45.4.1 Metadata and Scripts](#)

[45.4.2 Active Versus Dormant Installation](#)

[45.4.3 Binary Packages](#)

[45.4.4 Library Packages](#)

[45.4.5 Super-Packages](#)

[45.4.6 Source Packages](#)

[45.5 Anatomy of a Repository](#)

[45.5.1 Security](#)

[45.5.2 Universal Access](#)

[45.5.3 Release Process](#)

[45.5.4 Multitiered Mirrors and Caches](#)

[45.6 Managing a Repository](#)

[45.6.1 Repackaging Public Packages](#)

[45.6.2 Repackaging Third-Party Software](#)

[45.6.3 Service and Support](#)

[45.6.4 Repository as a Service](#)

[45.7 Repository Client](#)

[45.7.1 Version Management](#)

[45.7.2 Tracking Conflicts](#)

[45.8 Build Environment](#)

[45.8.1 Continuous Integration](#)

[45.8.2 Hermetic Build](#)

[45.9 Repository Examples](#)

[45.9.1 Staged Software Repository](#)

[45.9.2 OS Mirror](#)

[45.9.3 Controlled OS Mirror](#)

[45.10 Summary](#)

[Exercises](#)

[46 Web Services](#)

[46.1 Simple Web Servers](#)

[46.2 Multiple Web Servers on One Host](#)

[46.2.1 Scalable Techniques](#)

[46.2.2 HTTPS](#)

[46.3 Service Level Agreements](#)

[46.4 Monitoring](#)

[46.5 Scaling for Web Services](#)

[46.5.1 Horizontal Scaling](#)

[46.5.2 Vertical Scaling](#)

[46.5.3 Choosing a Scaling Method](#)

[46.6 Web Service Security](#)

[46.6.1 Secure Connections and Certificates](#)

[46.6.2 Protecting the Web Server Application](#)

[46.6.3 Protecting the Content](#)

[46.6.4 Application Security](#)

[46.7 Content Management](#)

[46.8 Summary](#)

[Exercises](#)

[Part IX Management Practices](#)

[47 Ethics](#)

[47.1 Informed Consent](#)

[47.2 Code of Ethics](#)

[47.3 Customer Usage Guidelines](#)

[47.4 Privileged-Access Code of Conduct](#)

[47.5 Copyright Adherence](#)

[47.6 Working with Law Enforcement](#)

[47.7 Setting Expectations on Privacy and Monitoring](#)

[47.8 Being Told to Do Something Illegal/Unethical](#)

[47.9 Observing Illegal Activity](#)

[47.10 Summary](#)

[Exercises](#)

[48 Organizational Structures](#)

[48.1 Sizing](#)

[48.2 Funding Models](#)

[48.3 Management Chain's Influence](#)

[48.4 Skill Selection](#)

[48.5 Infrastructure Teams](#)

[48.6 Customer Support](#)

[48.7 Helpdesk](#)

[48.8 Outsourcing](#)

[48.9 Consultants and Contractors](#)

[48.10 Sample Organizational Structures](#)

[48.10.1 Small Company](#)

[48.10.2 Medium-Size Company](#)

[48.10.3 Large Company](#)

[48.10.4 E-commerce Site](#)

[48.10.5 Universities and Nonprofit Organizations](#)

[48.11 Summary](#)

[Exercises](#)

[49 Perception and Visibility](#)

[49.1 Perception](#)

[49.1.1 A Good First Impression](#)

[49.1.2 Attitude, Perception, and Customers](#)

[49.1.3 Aligning Priorities with Customer Expectations](#)

[49.1.4 The System Advocate](#)

[49.2 Visibility](#)

[49.2.1 System Status Web Page](#)

[49.2.2 Management Meetings](#)

[49.2.3 Physical Visibility](#)

[49.2.4 Town Hall Meetings](#)

[49.2.5 Newsletters](#)

[49.2.6 Mail to All Customers](#)

[49.2.7 Lunch](#)

[49.3 Summary](#)

[Exercises](#)

[50 Time Management](#)

[50.1 Interruptions](#)

[50.1.1 Stay Focused](#)

[50.1.2 Splitting Your Day](#)

[50.2 Follow-Through](#)

[50.3 Basic To-Do List Management](#)

[50.4 Setting Goals](#)

[50.5 Handling Email Once](#)

[50.6 Precompiling Decisions](#)

[50.7 Finding Free Time](#)

[50.8 Dealing with Ineffective People](#)

[50.9 Dealing with Slow Bureaucrats](#)

[50.10 Summary](#)

[Exercises](#)

51 Communication and Negotiation

[51.1 Communication](#)

[51.2 I Statements](#)

[51.3 Active Listening](#)

[51.3.1 Mirroring](#)

[51.3.2 Summary Statements](#)

[51.3.3 Reflection](#)

[51.4 Negotiation](#)

[51.4.1 Recognizing the Situation](#)

[51.4.2 Format of a Negotiation Meeting](#)

[51.4.3 Working Toward a Win-Win Outcome](#)

[51.4.4 Planning Your Negotiations](#)

[51.5 Additional Negotiation Tips](#)

[51.5.1 Ask for What You Want](#)

[51.5.2 Don't Negotiate Against Yourself](#)

[51.5.3 Don't Reveal Your Strategy](#)

[51.5.4 Refuse the First Offer](#)

[51.5.5 Use Silence as a Negotiating Tool](#)

[51.6 Further Reading](#)

[51.7 Summary](#)

[Exercises](#)

52 Being a Happy SA

[52.1 Happiness](#)

[52.2 Accepting Criticism](#)

[52.3 Your Support Structure](#)

[52.4 Balancing Work and Personal Life](#)

[52.5 Professional Development](#)

[52.6 Staying Technical](#)

[52.7 Loving Your Job](#)

[52.8 Motivation](#)

[52.9 Managing Your Manager](#)

[52.10 Self-Help Books](#)

[52.11 Summary](#)

[Exercises](#)

53 Hiring System Administrators

[53.1 Job Description](#)

[53.2 Skill Level](#)

[53.3 Recruiting](#)

[53.4 Timing](#)

[53.5 Team Considerations](#)

[53.6 The Interview Team](#)

[53.7 Interview Process](#)

[53.8 Technical Interviewing](#)

[53.9 Nontechnical Interviewing](#)

[53.10 Selling the Position](#)

[53.11 Employee Retention](#)

[53.12 Getting Noticed](#)

[53.13 Summary](#)

[Exercises](#)

54 Firing System Administrators

[54.1 Cooperate with Corporate HR](#)

[54.2 The Exit Checklist](#)

[54.3 Removing Access](#)

[54.3.1 Physical Access](#)

[54.3.2 Remote Access](#)

[54.3.3 Application Access](#)

[54.3.4 Shared Passwords](#)

[54.3.5 External Services](#)

[54.3.6 Certificates and Other Secrets](#)

[54.4 Logistics](#)

[54.5 Examples](#)

[54.5.1 Amicably Leaving a Company](#)

[54.5.2 Firing the Boss](#)

[54.5.3 Removal at an Academic Institution](#)

[54.6 Supporting Infrastructure](#)

[54.7 Summary](#)

[Exercises](#)

[Part X Being More Awesome](#)

[55 Operational Excellence](#)

[55.1 What Does Operational Excellence Look Like?](#)

[55.2 How to Measure Greatness](#)

[55.3 Assessment Methodology](#)

[55.3.1 Operational Responsibilities](#)

[55.3.2 Assessment Levels](#)

[55.3.3 Assessment Questions and Look-For's](#)

[55.4 Service Assessments](#)

[55.4.1 Identifying What to Assess](#)

[55.4.2 Assessing Each Service](#)

[55.4.3 Comparing Results Across Services](#)

[55.4.4 Acting on the Results](#)

[55.4.5 Assessment and Project Planning Frequencies](#)

[55.5 Organizational Assessments](#)

[55.6 Levels of Improvement](#)

[55.7 Getting Started](#)

[55.8 Summary](#)

[Exercises](#)

[56 Operational Assessments](#)

- [56.1 Regular Tasks \(RT\)](#)
- [56.2 Emergency Response \(ER\)](#)
- [56.3 Monitoring and Metrics \(MM\)](#)
- [56.4 Capacity Planning \(CP\)](#)
- [56.5 Change Management \(CM\)](#)
- [56.6 New Product Introduction and Removal \(NPI/NPR\)](#)
- [56.7 Service Deployment and Decommissioning \(SDD\)](#)
- [56.8 Performance and Efficiency \(PE\)](#)
- [56.9 Service Delivery: The Build Phase](#)
- [56.10 Service Delivery: The Deployment Phase](#)
- [56.11 Toil Reduction](#)
- [56.12 Disaster Preparedness](#)

Epilogue

Part XI Appendices

A What to Do When . . .

B The Many Roles of a System Administrator

- [B.1 Common Positive Roles](#)
- [B.2 Negative Roles](#)
- [B.3 Team Roles](#)
- [B.4 Summary](#)
- [Exercises](#)

Bibliography

Index

Preface

This is an unusual book. This is not a technical book. It is a book of strategies and frameworks and anecdotes and tacit knowledge accumulated from decades of experience as system administrators.

Junior SAs focus on learning which commands to type and which buttons to click. As you get more advanced, you realize that the bigger challenge is understanding why we do these things and how to organize our work. That's where strategy comes in.

This book gives you a framework—a way of thinking about system administration problems—rather than narrow how-to solutions to particular problems. Given a solid framework, you can solve problems every time they appear, regardless of the operating system (OS), brand of computer, or type of environment. This book is unique because it looks at system administration from this holistic point of view, whereas most other books for SAs focus on how to maintain one particular product. With experience, however, all SAs learn that the big-picture problems and solutions are largely independent of the platform. This book will change the way you approach your work as an SA.

This book is Volume 1 of a series. Volume 1 focuses on enterprise infrastructure, customer support, and management issues. Volume 2, *The Practice of Cloud System Administration* (ISBN: 9780321943187), focuses on web operations and distributed computing.

These books were born from our experiences as SAs in a variety of organizations. We have started new companies. We have helped sites to grow. We have worked at small start-ups and universities, where lack of funding was an issue. We have worked at midsize and large multinationals, where mergers and spinoffs gave rise to strange challenges. We have worked at fast-paced companies that do business on the Internet and where high-availability, high-performance, and scaling issues were the norm. We have worked at slow-paced companies at which “high tech” meant cordless phones. On the surface, these are very different environments with diverse challenges; underneath, they have the same building blocks, and the same fundamental principles apply.

Who Should Read This Book

This book is written for system administrators at all levels who seek a deeper insight into the best practices and strategies available today. It is also useful for

managers of system administrators who are trying to understand IT and operations.

Junior SAs will gain insight into the bigger picture of how sites work, what their roles are in the organizations, and how their careers can progress. Intermediate-level SAs will learn how to approach more complex problems, how to improve their sites, and how to make their jobs easier and their customers happier.

Whatever level you are at, this book will help you understand what is behind your day-to-day work, learn the things that you can do now to save time in the future, decide policy, be architects and designers, plan far into the future, negotiate with vendors, and interface with management.

These are the things that senior SAs know and your OS's manual leaves out.

Basic Principles

In this book you will see a number of principles repeated throughout:

- **Automation:** Using software to replace human effort. Automation is critical. We should not be doing tasks; we should be maintaining the system that does tasks for us. Automation improves repeatability and scalability, is key to easing the system administration burden, and eliminates tedious repetitive tasks, giving SAs more time to improve services. Automation starts with getting the process well defined and repeatable, which means documenting it. Then it can be optimized by turning it into code.
- **Small batches:** Doing work in small increments rather than large hunks. Small batches permit us to deliver results faster, with higher quality, and with less stress.
- **End-to-end integration:** Working across teams to achieve the best total result rather than performing local optimizations that may not benefit the greater good. The opposite is to work within your own silo of control, ignoring the larger organization.
- **Self-service systems:** Tools that empower others to work independently, rather than centralizing control to yourself. Shared services should be an enablement platform, not a control structure.
- **Communication:** The right people can solve more problems than hardware or software can. You need to communicate well with other SAs and with your customers. It is your responsibility to initiate communication. Communication ensures that everyone is working toward

the same goals. Lack of communication leaves people concerned and annoyed. Communication also includes documentation. Documentation makes systems easier to support, maintain, and upgrade. Good communication and proper documentation also make it easier to hand off projects and maintenance when you leave or take on a new role.

These principles are universal. They apply at all levels of the system. They apply to physical networks and to computer hardware. They apply to all operating systems running at a site, all protocols used, all software, and all services provided. They apply at universities, nonprofit institutions, government sites, businesses, and Internet service sites.

What Is an SA?

If you asked six system administrators to define their jobs, you would get seven different answers. The job is difficult to define because system administrators do so many things. An SA looks after computers, networks, and the people who use them. An SA may look after hardware, operating systems, software, configurations, applications, or security. An SA influences how effectively other people can or do use their computers and networks.

A system administrator sometimes needs to be a business-process consultant, corporate visionary, janitor, software engineer, electrical engineer, economist, psychiatrist, mindreader, and, occasionally, bartender.

As a result, companies give SAs different titles. Sometimes, they are called network administrators, system architects, system engineers, system programmers, operators, and so on.

This book is for “all of the above.”

We have a very general definition of system administrator: one who manages computer and network systems on behalf of another, such as an employer or a client. SAs are the people who make things work and keep it all running.

System Administration Matters

System administration matters because computers and networks matter. Computers are a lot more important than they were years ago.

Software is eating the world. Industry after industry is being taken over by software. Our ability to make, transport, and sell real goods is more dependent on software than on any other single element. Companies that are good at software are beating competitors that aren't.

All this software requires operational expertise to deploy and keep it running. In turn, this expertise is what makes SAs special.

For example, not long ago, manual processes were batch oriented. Expense reports on paper forms were processed once a week. If the clerk who processed them was out for a day, nobody noticed. This arrangement has since been replaced by a computerized system, and employees file their expense reports online, 24/7.

Management now has a more realistic view of computers. Before they had PCs on their desktops, most people's impressions of computers were based on how they were portrayed in films: big, all-knowing, self-sufficient, miracle machines. The more people had direct contact with computers, the more realistic people's expectations became. Now even system administration itself is portrayed in films. The 1993 classic *Jurassic Park* was the first mainstream movie to portray the key role that system administrators play in large systems. The movie also showed how depending on one person is a disaster waiting to happen. IT is a team sport. If only Dennis Nedry had read this book.

In business, nothing is important unless the CEO feels that it is important. The CEO controls funding and sets priorities. CEOs now consider IT to be important. Email was previously for nerds; now CEOs depend on email and notice even brief outages. The massive preparations for Y2K also brought home to CEOs how dependent their organizations have become on computers, how expensive it can be to maintain them, and how quickly a purely technical issue can become a serious threat. Most people do not think that they simply "missed the bullet" during the Y2K change, but rather recognize that problems were avoided thanks to tireless efforts by many people. A CBS Poll shows 63 percent of Americans believe that the time and effort spent fixing potential problems was worth it. A look at the news lineups of all three major network news broadcasts from Monday, January 3, 2000, reflects the same feeling.

Previously, people did not grow up with computers and had to cautiously learn about them and their uses. Now people grow up using computers. They consume social media from their phones (constantly). As a result they have higher expectations of computers when they reach positions of power. The CEOs who were impressed by automatic payroll processing are being replaced by people who grew up sending instant messages all day long. This new wave of management expects to do all business from their phones.

Computers matter more than ever. If computers are to work, and work well, system administration matters. We matter.

Organization of This Book

This book is divided into the following parts:

- **Part I, “[Game-Changing Strategies](#).”** This part describes how to make the next big step, for both those who are struggling to keep up with a deluge of work, and those who have everything running smoothly.
- **Part II, “[Workstation Fleet Management](#).”** This part covers all aspects of laptops and desktops. It focuses on how to optimize workstation support by treating these machines as mass-produced commodity items.
- **Part III, “[Servers](#).”** This part covers server hardware management—from the server strategies you can choose, to what makes a machine a server and what to consider when selecting server hardware.
- **Part IV, “[Services](#).”** This part covers designing, building, and launching services, converting users from one service to another, building resilient services, and planning for disaster recovery.
- **Part V, “[Infrastructure](#).”** This part focuses on the underlying infrastructure. It covers network architectures and operations, an overview of datacenter strategies, and datacenter operations.
- **Part VI, “[Helpdesks and Support](#).”** This part covers everything related to providing excellent customer service, including documentation, how to handle an incident report, and how to approach debugging.
- **Part VII, “[Change Processes](#).”** This part covers change management processes and describes how best to manage big and small changes. It also covers optimizing support by centralizing services.
- **Part VIII, “[Service Recommendations](#).”** This part takes an in-depth look at what you should consider when setting up some common services. It covers monitoring, nameservices, email, web, printing, storage, backups, and software depositories.
- **Part IX, “[Management Practices](#).”** This part is for managers and non-managers. It includes such topics as ethics, organizational structures, perception, visibility, time management, communication, happiness, and hiring and firing SAs.
- **Part X, “[Being More Awesome](#).”** This part is essential reading for all managers. It covers how to assess an SA team’s performance in a constructive manner, using the Capability Maturity Model to chart the way forward.
- **Part XI, “[Appendices](#).”** This part contains two appendices. The first is a

checklist of solutions to common situations, and the second is an overview of the positive and negative team roles.

What's New in the Third Edition

The first two editions garnered a lot of positive reviews and buzz. We were honored by the response. However, the passing of time made certain chapters look passé. Most of our bold new ideas are now considered common-sense practices in the industry.

The first edition, which reached bookstores in August 2001, was written mostly in 2000 before Google was a household name and modern computing meant a big Sun multiuser system. Many people did not have Internet access, and the cloud was only in the sky. The second edition was released in July 2007. It smoothed the rough edges and filled some of the major holes, but it was written when DevOps was still in its embryonic form.

The third edition introduces two dozen entirely new chapters and many highly revised chapters; the rest of the chapters were cleaned up and modernized. Longer chapters were split into smaller chapters. All new material has been rewritten to be organized around choosing strategies, and DevOps and SRE practices were introduced where they seem to be the most useful.

If you've read the previous editions and want to focus on what is new or updated, here's where you should look:

- [Part I, “Game-Changing Strategies” \(Chapters 1–4\)](#)
- [Part II, “Workstation Fleet Management” \(Chapters 5–12\)](#)
- [Part III, “Servers” \(Chapters 13–15\)](#)
- [Part IV, “Services” \(Chapters 16–20 and 22\)](#)
- [Chapter 23, “Network Architecture,”](#) and [Chapter 24, “Network Operations”](#)
- [Chapter 32, “Change Management”](#)
- [Chapter 35, “Centralization Overview,”](#) [Chapter 36, “Centralization Recommendations,”](#) and [Chapter 37, “Centralizing a Service”](#)
- [Chapter 43, “Data Storage”](#)
- [Chapter 45, “Software Repositories,”](#) and [Chapter 46, “Web Services”](#)
- [Chapter 55, “Operational Excellence,”](#) and [Chapter 56, “Operational Assessments”](#)

Books, like software, always have bugs. For a list of updates, along with news

and notes, and even a mailing list you can join, visit our web site:

www.EverythingSysAdmin.com

What's Next

Each chapter is self-contained. Feel free to jump around. However, we have carefully ordered the chapters so that they make the most sense if you read the book from start to finish. Either way, we hope that you enjoy the book. We have learned a lot and had a lot of fun writing it. Let's begin.

Thomas A. Limoncelli
Stack Overflow, Inc.
tom@limoncelli.com

Christina J. Hogan
chogan@chogan.com

Strata R. Chalup
Virtual.Net, Inc.
strata@virtual.net

Register your copy of *The Practice of System and Network Administration, Volume 1, Third Edition*, at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780321919168) and click Submit. Once the process is complete, you will find any available bonus content under “Registered Products.”

Acknowledgments

For the Third Edition

Everyone was so generous with their help and support. We have so many people to thank!

Thanks to the people who were extremely generous with their time and gave us extensive feedback and suggestions: Derek J. Balling, Stacey Frye, Peter Grace, John Pellman, Iustin Pop, and John Willis.

Thanks to our friends, co-workers, and industry experts who gave us support, inspiration, and cool stories to use: George Beech, Steve Blair, Kyle Brandt, Greg Bray, Nick Craver, Geoff Dalgas, Michelle Fredette, David Fullerton, Dan Gilmartin, Trey Harris, Jason Harvey, Mark Henderson, Bryan Jen, Gene Kim, Thomas Linkin, Shane Madden, Jim Maurer, Kevin Montrose, Steve Murawski, Xavier Nicollet, Dan O'Boyle, Craig Peterson, Jason Punyon, Mike Rembetsy, Neil Ruston, Jason Shantz, Dagobert Soergel, Kara Sowles, Mike Stoppay, and Joe Youn.

Thanks to our team at Addison-Wesley: Debra Williams Cauley, for her guidance; Michael Thurston, our developmental editor who took this sow's ear and made it into a silk purse; Kim Boedigheimer, who coordinated and kept us on schedule; Lori Hughes, our LATEX wizard; Julie Nahil, our production editor; Jill Hobbs, our copy editor; and Ted Laux for making our beautiful index!

Last, but not least, thanks and love to our families who suffered for years as we ignored other responsibilities to work on this book. Thank you for understanding! We promise this is our last book. Really!

For the Second Edition

In addition to everyone who helped us with the first edition, the second edition could not have happened without the help and support of Lee Damon, Nathan Dietsch, Benjamin Feen, Stephen Harris, Christine E. Polk, Glenn E. Sieb, Juhani Tali, and many people at the League of Professional System Administrators (LOPSA). Special 73s and 88s to Mike Chalup for love, loyalty, and support, and especially for the mountains of laundry done and oceans of dishes washed so Strata could write. And many cuddles and kisses for baby Joanna Lear for her patience.

Thanks to Lumeta Corporation for giving us permission to publish a second edition.

Thanks to Wingfoot for letting us use its server for our bug-tracking database. Thanks to Anne Marie Quint for data entry, copyediting, and a lot of great suggestions.

And last, but not least, a big heaping bowl of “couldn’t have done it without you” to Mark Taub, Catherine Nolan, Raina Chrobak, and Lara Wysong at Addison-Wesley.

For the First Edition

We can’t possibly thank everyone who helped us in some way or another, but that isn’t going to stop us from trying. Much of this book was inspired by Kernighan and Pike’s *The Practice of Programming* and John Bentley’s second edition of *Programming Pearls*.

We are grateful to Global Networking and Computing (GNAC), Synopsys, and Eircom for permitting us to use photographs of their datacenter facilities to illustrate real-life examples of the good practices that we talk about.

We are indebted to the following people for their helpful editing: Valerie Natale, Anne Marie Quint, Josh Simon, and Amara Willey.

The people we have met through USENIX and SAGE and the LISA conferences have been major influences in our lives and careers. We would not be qualified to write this book if we hadn’t met the people we did and learned so much from them.

Dozens of people helped us as we wrote this book—some by supplying anecdotes, some by reviewing parts of or the entire book, others by mentoring us during our careers. The only fair way to thank them all is alphabetically and to apologize in advance to anyone whom we left out: Rajeev Agrawala, Al Aho, Jeff Allen, Eric Anderson, Ann Benninger, Eric Berglund, Melissa Binde, Steven Branigan, Sheila Brown-Klinger, Brent Chapman, Bill Cheswick, Lee Damon, Tina Darmohray, Bach Thuoc (Daisy) Davis, R. Drew Davis, Ingo Dean, Arnold de Leon, Jim Dennis, Barbara Dijker, Viktor Dukhovni, Chelle-Marie Ehlers, Michael Erlinger, Paul Evans, Rémy Evard, Lookman Fazal, Robert Fulmer, Carson Gaspar, Paul Glick, David “Zonker” Harris, Katherine “Cappy” Harrison, Jim Hickstein, Sandra Henry-Stocker, Mark Horton, Bill “Whump” Humphries, Tim Hunter, Jeff Jensen, Jennifer Joy, Alan Judge, Christophe Kalt, Scott C. Kennedy, Brian Kernighan, Jim Lambert, Eliot Lear, Steven Levine, Les Lloyd, Ralph Loura, Bryan MacDonald, Sherry McBride, Mark Mellis, Cliff Miller, Hal Miller, Ruth Milner, D. Toby Morrill, Joe Morris, Timothy Murphy, Ravi Narayan, Nils-Peter Nelson, Evi Nemeth, William Ninke, Cat Okita, Jim

Paradis, Pat Parseghian, David Parter, Rob Pike, Hal Pomeranz, David Presotto, Doug Reimer, Tommy Reingold, Mike Richichi, Matthew F. Ringel, Dennis Ritchie, Paul D. Rohrigstamper, Ben Rosengart, David Ross, Peter Salus, Scott Schultz, Darren Shaw, Glenn Sieb, Karl Siil, Cicely Smith, Bryan Stansell, Hal Stern, Jay Stiles, Kim Supsinkas, Ken Thompson, Greg Tusar, Kim Wallace, The Rabbit Warren, Dr. Geri Weitzman, Glen Wiley, Pat Wilson, Jim Witthoff, Frank Wojcik, Jay Yu, and Elizabeth Zwicky.

Thanks also to Lumeta Corporation and Lucent Technologies/Bell Labs for their support in writing this book.

Last, but not least, the people at Addison-Wesley made this a particularly great experience for us. In particular, our gratitude extends to Karen Gettman, Mary Hart, and Emily Frey.

About the Authors

Thomas A. Limoncelli is an internationally recognized author, speaker, and system administrator. During his seven years at Google NYC, he was an SRE for projects such as Blog Search, Ganeti, and internal enterprise IT services. He now works as an SRE at Stack Overflow. His first paid system administration job was as a student at Drew University in 1987, and he has since worked at small and large companies, including AT&T/Lucent Bell Labs and Lumeta. In addition to this book series, he is known for his book *Time Management for System Administrators* (O'Reilly, 2005). His hobbies include grassroots activism, for which his work has been recognized at state and national levels. He lives in New Jersey.

Christina J. Hogan has 20 years of experience in system administration and network engineering, from Silicon Valley to Italy and Switzerland. She has gained experience in small start-ups, midsize tech companies, and large global corporations. She worked as a security consultant for many years; in that role, her customers included eBay, Silicon Graphics, and SystemExperts. In 2005, she and Tom shared the USENIX LISA Outstanding Achievement Award for the first edition of this book. Christina has a bachelor's degree in mathematics, a master's degree in computer science, a doctorate in aeronautical engineering, and a diploma in law. She also worked for six years as an aerodynamicist in a Formula 1 racing team and represented Ireland in the 1988 Chess Olympiad. She lives in Switzerland.

Strata R. Chalup has been leading and managing complex IT projects for many years, serving in roles ranging from project manager to director of operations. She started administering VAX Ultrix and Unisys Unix in 1983 at MIT and spent the dot-com years in Silicon Valley building Internet services for clients like iPlanet and Palm. She joined Google in 2015 as a technical project manager. She has served on the BayLISA and SAGE boards. Her hobbies include being a master gardener and working with new technologies such as Arduino and 2D CAD/CAM devices. She lives in Santa Clara County, California.

Part I: Game-Changing Strategies

Chapter 1. Climbing Out of the Hole

Some system administration teams are struggling. They're down in a hole trying to climb out. If this sounds like your team, this chapter will help you. If this doesn't sound like your team, you'll still find a lot of useful information about how successful teams stay successful.

There are two things we see at all successful sites that you won't see at sites that are struggling. Get those things right and everything else falls into place.

These two things are pretty much the same for all organizations—and we really mean *all*. We visit sites around the world: big and small, for-profit and nonprofit, well funded and nearly broke. Some are so large they have thousands of system administrators (SAs); others are so tiny that they have a part-time information technology (IT) person who visits only once a week. No matter what, these two attributes are found at the successful sites and are missing at the ones that struggle.

Most of this book is fairly aspirational: It provides the best way to design, build, and run IT services and infrastructure. The feedback that we get online and at conferences is usually along the lines of “Sounds great, but how can I do any of that if we're struggling with what we have?” If we don't help you fix those problems first, the rest of the book isn't so useful. This chapter should help you gain the time you need so you can do the other things in this book.

So what are these two things? The first is that successful SA teams have a way to track and organize their **work in progress (WIP)**. WIP is requests from customers¹ plus all the other tasks you need, want, or have to do, whether planned or unplanned. Successful teams have some kind of request-tracking system and a plan for how WIP flows through it. Struggling organizations don't, and requests frequently get lost or forgotten.

1. We use the term **customers** rather than users. We find this brings about a positive attitude shift. IT is a service industry, supporting the needs of people and the business.

The second thing is that successful SA teams have eliminated the two biggest **time sinkholes** that all IT teams face. A time sinkhole is an issue that consumes large amounts of time and causes a domino effect of other problems. Eliminating a time sinkhole is a game-changer not just because it saves time, but because it also prevents other trouble.

In IT, those time sinkholes are operating system (OS) installation and the software deployment process. Struggling teams invariably are doing those two

things manually. This approach not only unnecessarily saps their time but also creates new problems. Some problems are due to mistakes and variations between how different people do the same process. Some problems are due to how the same person does it while feeling refreshed on Monday versus tired on Friday afternoon.

Any struggling SA team can begin their turnaround by adopting better ways to

- **Track and control WIP:** Install some kind of helpdesk automation software, request-tracking system, ticket queue, or Kanban board.
- **Automate OS installation:** Start every machine in the same state by automating the installation and configuration of the OS and applications.
- **Adopt CI/CD for software pushes:** The process of pushing new software releases into production should be automated by using DevOps concepts such as continuous integration and delivery (CI/CD).

Obviously not all readers of this book are members of struggling organizations, but if you are, take care of those two things first. Everything else will become easier.

The remainder of this chapter is a quick-start guide for achieving those two goals. They are covered more in depth later in the book, but consider this chapter a way to crawl out of the hole you are in.

This guy's walking down the street when he falls in a hole. The walls are so steep he can't get out.

A doctor passes by and the guy shouts up, "Hey! Can you help me out?" The doctor writes a prescription, throws it down in the hole, and moves on.

Then a priest comes along and the guy shouts up, "Father, I'm down in this hole. Can you help me out?" The priest writes out a prayer, throws it down in the hole, and moves on.

Then a friend walks by. "Hey, Joe, it's me. Can you help me out?" And the friend jumps in the hole.

Our guy says, "Are you stupid? Now we're both down here."

The friend says, "Yeah, but I've been down here before and I know the way out!"

— Leo McGarry, *The West Wing*, "Noël," Season 2, Episode 10

1.1 Organizing WIP

Successful organizations have some system for collecting requests, organizing them, tracking them, and seeing them through to completion. Nearly every organization does it differently, but the point is that they have a system and they stick with it.

Operations science uses the term “work in progress” to describe requests from customers plus all the other tasks you need, want, or have to do, whether planned or unplanned. WIP comes from direct customer requests as well as tasks that are part of projects, emergency requests, and incidents such as outages and alerts from your monitoring system.

Struggling organizations are unorganized at best and disorganized at worst. Requests get lost or forgotten. Customers have no way of knowing the status of their requests, so they spend a lot of time asking SAs, “Is it done yet?”—or, even worse, they don’t ask and assume incompetence. When the team is overloaded with too much WIP, they go into panic mode and deal with the loudest complainers instead of the most important WIP. Some teams have been in this overloaded state for so long they don’t know there is any other way.

Whatever strategy is employed to organize WIP, it must have the characteristic that quick requests get a fast response, longer requests get attention, and emergencies get the immediate attention and resources required to resolve them.

A typical quick request is a customer who needs a password reset. Performing a password reset takes very little SA time, and not doing it blocks all other work for the customer; a customer can’t do anything if he or she can’t log in. If you were to do all WIP in the order in which it arrives, the password reset might not happen for days. That is unreasonable. Therefore we need a mechanism for such requests to be treated with higher priority.

The two most popular strategies for managing WIP are ticket systems and Kanban boards. Either is better than not having any system. If you are having difficulty deciding, default to a simple ticket system. You will have an easier time getting people to adopt it because it is the more common solution. Later you can add a Kanban board for project-oriented work.

1.1.1 Ticket Systems

A ticket system permits customers to submit requests electronically. The request is stored in a database where it can be prioritized, assigned, and tracked. As the issue is worked on, the ticket is updated with notes and information. The system

also records communication between the customer and the SA as the request is discussed or clarified.

Without such a system, WIP gets lost, forgotten, or confused. A customer is left wondering what the status is, and the only way to find out is to bother the SA, which distracts the SA from trying to get work done.

A ticket system permits a team to share work better. Team members can see the history of the ticket, making it easier to hand off a ticket from one SA to another.

A solo system administrator benefits from using a ticket system because it is like having an assistant assure that requests stay organized. This reduces the cognitive load involved in remembering the requests, their prioritization, and so on. Since the ticket system keeps the history of each WIP, you can return to an issue and more easily pick up where you left off.

The manager of the SA team can examine the queue, or list tickets, and balance workloads among team members. Management can also intervene to reject requests that are out of scope for the team, and see tickets that have stalled or need attention. Thus the manager can proactively solve problems rather than waiting for them to blow up.

A ticket system also introduces a degree of reality into the SA–customer relationship. Sometimes customers feel as if they’ve been waiting forever for a request to be completed and decide to complain to management. Without a ticket system, the SAs would bear the brunt of the customer’s anger whether or not the impatience was justified. In this scenario, it is the customer’s word against the SAs’.

With a ticket system the accusation of slow support can be verified. If true, the ticket history can be used to review the situation and a constructive conversation can be had. If the customer was just being impatient the manager has the opportunity to explain the service level agreement (SLA) and appropriate expectations. That said, we’ve seen situations where the customer claimed that the problem has been around a long time but the customer only recently had time to open a ticket about it. The manager can then decide how to politely and gracefully explain that “We can’t help you, unless you help us help you.” System administrators should have foresight, but they can’t be mind-readers.

Now we must find a way to assure that quick requests get fast responses.

Imagine your list of tickets sorted with the fast/urgent items at one end and the big/far-reaching items at the other. If you work on only the items from the fast/urgent end, you will please your customers in the short term, but you’ll

never have time for the big items that have far-reaching impact. If you work on only the items from the far-reaching end, your customers will never receive the direct support they need.

You must find a way to “eat from both ends of the queue” by assigning different people to each, or alternating between the two. Commonly organizations create a tiered support structure, where one tier of IT support handles quick requests and forwards larger requests to a second tier of people.

A large organization may have a helpdesk team (Tier 1) that receives all new requests. They handle the quick requests themselves, but other requests are prioritized and assigned to the system administration team (Tier 2). Often Tier 1 is expected to handle 80 percent of all requests; the remaining require Tier 2’s technically deeper and more specialized knowledge. This assures quick requests get the immediate response they deserve.

In a medium-size team Tier 1 and Tier 2 are members of the same team, but Tier 1 requests are handled by the more junior SAs.

In environments where WIP is mostly project oriented, and little arrives directly from customers, the team may have a single person who handles Tier 1 requests or team members may take turns in a weekly rotation.

A solo system administrator has a special challenge: There is never time for project work if you are always dealing with interruptions from customers. Project work takes sustained focus for hours at a time. Balancing this with the need to handle quick requests quickly requires strategy.

In this case, use a strategy of dividing the day into segments: customer-focused hours where you work on quick requests and tickets, and project-focused hours where you focus on projects. During project-focused hours anyone visiting with a quick request is politely asked to open a ticket, which will get attention later. Use polite phrases like, “I’m in the middle of an important project for my boss. Could you open a ticket and I’ll get to it this afternoon?”

Allocate the first two and last two hours of the day to be customer-focused work, leaving a full half-day of every day for project-focused work. This strategy works well because most quick requests happen early in the morning as people are starting their day.

Two things help make this strategy a success. The first is management support. Ideally your boss will see the value in a strategy that enables customers to receive attention when they typically need it most, yet enables you to complete important projects, too. Managers can make it policy that people should file tickets when possible and steer otherwise unavoidable personal visits

to the first and last two hours of the day.

The second key to success is that no matter which type of hour you are in, you should make an exception for service outages and ultra-quick requests. It should be obvious that reacting to an outage or other emergency should take priority. Unfortunately, this may result in all requests being described as emergencies. Supportive management can prevent this by providing a written definition of an emergency.

Ultra-quick requests are ones that can be accomplished faster than it would take a person to file a ticket. This includes things like password resets and answering very simple questions. Also, if the issue is that the printer is broken, this can become an ultra-quick request by simply asking the customer to use a different printer for now.

This schedule should be the same every day. This consistency helps you stick to it, and is less confusing to customers. Expecting them to remember your schedule is a stretch; expecting them to remember how it shifts and changes every day is unreasonable.

Sample Emergency Definitions

Here are some example definitions of “emergency” we’ve seen businesses use:

- A server or service being down
 - A user-facing issue that affects ten or more people
 - A technical problem causing the company to “lose money fast”
 - Customer proprietary data actively being leaked
 - (At a newspaper) anything that will directly prevent the newspapers from being printed and loaded onto trucks by 4:30 AM
-

There are many WIP-tracking systems to choose from. Open source software products include Request Tracker (from Best Practical), Trac, and OTRS. For Windows, SpiceWorks is a good option and is available at no cost.

This section is just an overview. [Chapter 27, “Customer Support,”](#) explains how to manage a helpdesk itself, with [Section 27.10](#) focusing directly on how to get the most out of request-tracking software. [Chapter 28, “Handling an Incident Report,”](#) focuses on how to process a single request really well.

1.1.2 Kanban

Kanban is another way of organizing WIP. It is more appropriate for project-focused SA teams. Kanban has the benefit of providing better transparency to stakeholders outside the SA team while at the same time reducing the mad crush of work that some IT organizations live under. We'll start with a simple example and work up to more sophisticated ones.

[Figure 1.1](#) depicts a simple Kanban board example. Unlike a ticket system, which organizes WIP into a single long list, Kanban divides WIP into three lists, or columns. The columns are used as follows:

- **Backlog:** The waiting room where new WIP arrives
- **Active:** Items being worked on
- **Completed:** Items that are finished

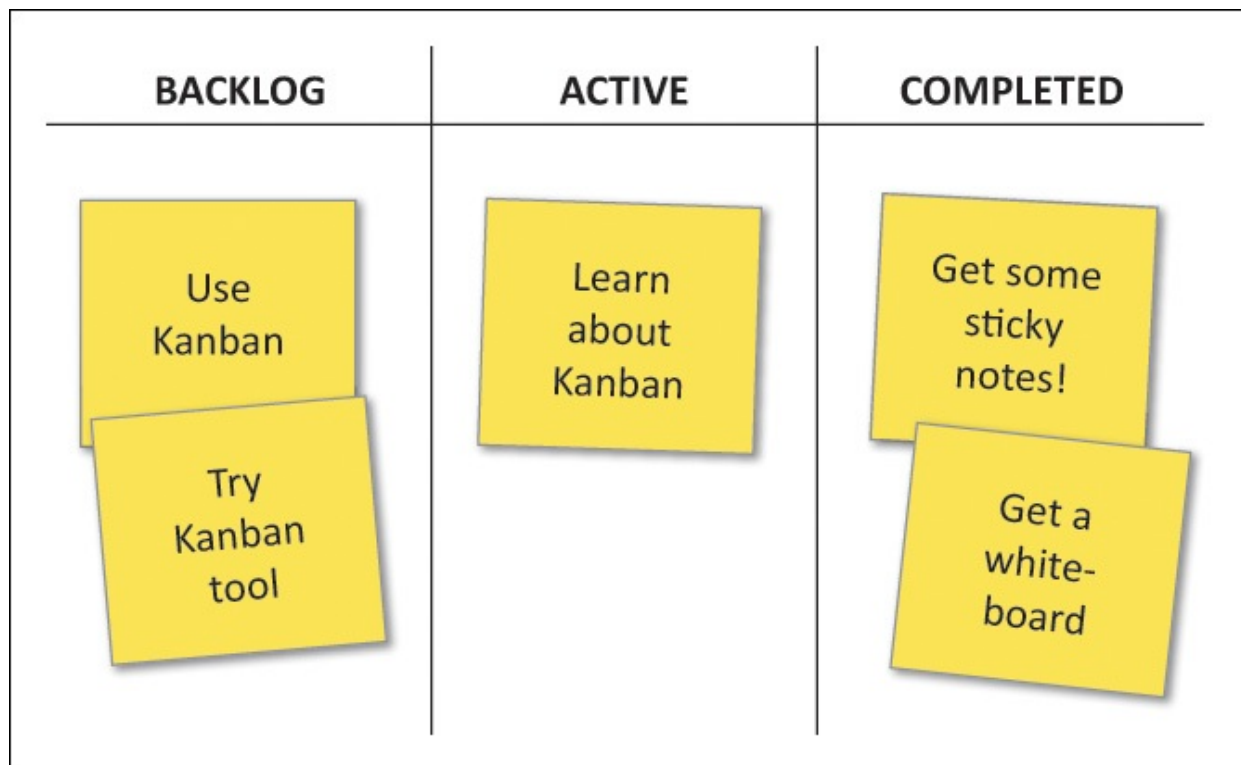


Figure 1.1: A simple Kanban board

All new WIP is recorded and placed in the backlog. The tasks are called **cards** because Kanban was originally done using index cards on a pin board. Each card represents one unit of work. Large projects are broken down into smaller cards, or steps, that can be reasonably achieved in one week.

Each Monday morning the team has a meeting where each member picks three cards from the backlog column to work on during that week. Those cards are moved to the Active column. By Friday all three tasks should be finished and

the cards moved to the Completed column. On Monday another round of items is selected and the process begins again.

Monday's meeting also includes a retrospective. This is a discussion of what went right and wrong during the previous week. Did everyone complete all their cards or was there more work to do? This review is not done to blame anyone for not finishing their items, but to troubleshoot the situation and improve. Does the team need to get better at estimating how much can be done in a week? Does the team need to break projects into smaller chunks? At the end of the retrospective, items in the Completed column are moved to an accomplishments box.

Picking three cards per week sets a cadence, or drumbeat, that assures progress happens at a smooth and predictable pace. Not all teams pick three cards each week. Some establish guidelines for what constitutes small, medium, and large tasks and then expect each person to pick perhaps one large and two small tasks, or two medium tasks, and so on. Keep it as simple as possible. People should have more than one card per week so that when one task is blocked, they have other tasks they can work on. A person should not have too many cards at a time, as the cognitive load this creates reduces productivity. Allocating the same number of cards each week makes the system more predictable and less chaotic. Chaos creates stress in workers, which also reduces productivity. It is better to adjust the amount of work a card represents than to change how many cards each person takes each week.

Some teams use more than three columns. For example, a Verifying column may be inserted after the Active column to hold items waiting to be verified. Some teams put new WIP in a New column, but move it to the backlog once it has been prioritized.

In addition to columns, rows can be added to further segregate the items. These are called swimlanes because they guide the item as it swims across the columns to the finish line. A team might have two swimlanes, one for development work and another for operational tasks. A large team might have a swimlane for each subteam, a configuration that is particularly useful if some people straddle many subteams. This is depicted in [Figure 1.2](#).

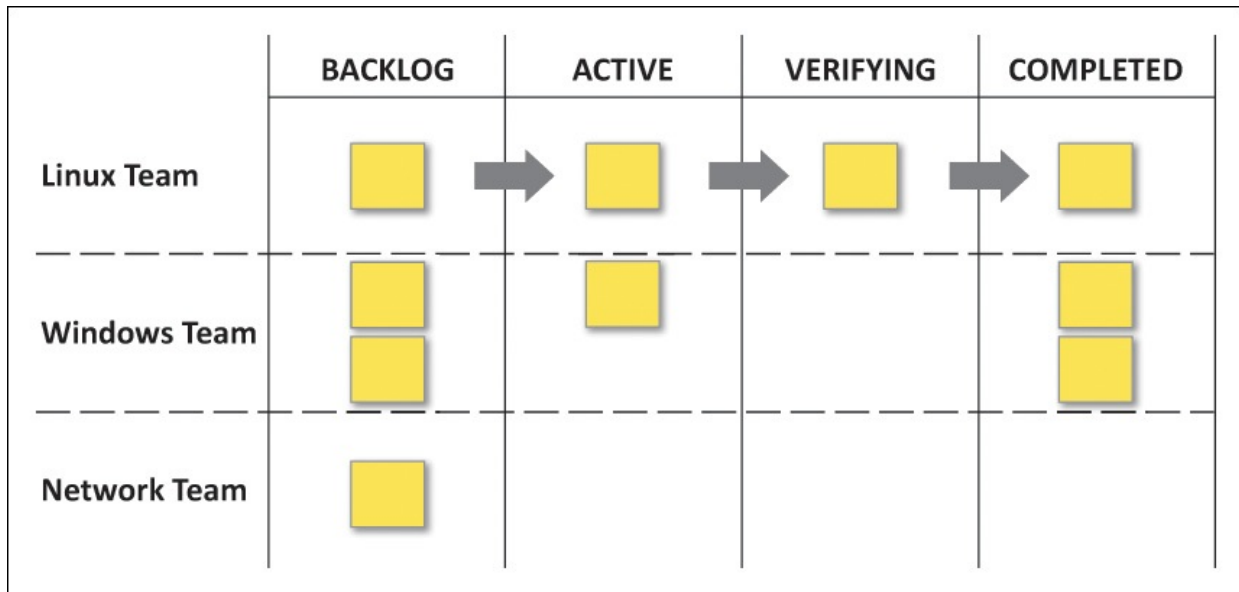


Figure 1.2: A Kanban board with separate swimlanes for Linux, Windows, and network projects

As mentioned earlier, there must be a way for quick requests to get attention quickly. Kanban wouldn't work very well if someone needing a password reset had to wait until Monday to have that task picked by someone.

A simple way to deal with this is to have one card each week labeled Quick Request Card (QRC). Whoever takes this card handles all quick requests and other interruptions for the week. Depending on the volume of quick requests in your environment, this might be the only card the person takes. This role is often called the interrupt sponge, but we prefer to call it hero of the week, a moniker coined at New Relic ([Goldfuss 2015](#)).

Kanban works better than ticket systems for a number of reasons.

First, the system becomes “pull” instead of “push.” A ticket system is a push system. Work gets accomplished at the rate that it can be pushed through the system. To get more work done, you push the employees harder. If there is a large backlog of incomplete tickets, the employees are pushed to work harder to get all tickets done. In practice, this system usually fails. Tasks cannot be completed faster just because we want them to. In fact, people are less productive when they are overwhelmed. When people are pushed, they naturally reduce quality in an effort to get more done in the same amount of time. Quality and speed could be improved by automating more, but it is difficult to set time aside for such projects when there is an overflow of tickets.

A Kanban system is a pull system: WIP is pulled through the system so that it gets completed by the weekly deadline. You select a fixed amount of work to be

done each week, and that work is pulled through the system. Because there is a fixed amount of time to complete the task, people can focus on doing the best job in that amount of time. If management needs more WIP completed each week, it can make this possible by hiring more people, allocating resources to fix bottlenecks, providing the means to do the work faster, and so on.

Second, Kanban improves transparency. When customers can see where their card is in relation to other cards, they get a better sense of company priorities. If they disagree with the priorities, they can raise the issue with management. This is better than having the debate brought to the SAs who have little control over priorities and become defensive. It is also better than having people just wonder why things are taking so long; in the absence of actual information they tend to assume the IT team is incompetent. One organization we interviewed hosted a monthly meeting where the managers of the departments they supported reviewed the Kanban board and discussed priorities. By involving all stakeholders, they created a more collaborative community.

Getting started with Kanban requires very few resources. Many teams start with index cards on a wall or whiteboard. Electronic Kanban systems improve logging of history and metrics, as well as make it easier for remote workers to be involved. There are commercial, web-based systems that are free or inexpensive, such as Trello and LeanKit. Open source solutions exist as well, such as Taiga and KanBoard.

Customize the Kanban board to the needs of your team over time. There is no one perfect combination of columns and swimlanes. Start simple. Add and change them as the team grows, matures, and changes.

Our description of Kanban is very basic. There are ways to handle blocks, rate limits, and other project management concepts. A more complete explanation can be found in *Personal Kanban: Mapping Work | Navigating Life* by Benson & Barry ([2011](#)) and *Kanban: Successful Evolutionary Change for Your Technology Business* by Anderson ([2010](#)). Project managers may find *Agile Project Management with Kanban* by Brechner ([2015](#)) to be useful.

Meat Grinder

The best-quality ground beef can be made by turning the crank of the meat grinder at a certain speed and stopping to clean the device every so often.

A push-based management philosophy focuses on speed. More meat can be ground by pushing harder and forcing it through the system,

bruising the meat in the process, but achieving the desired output quantity. Speed can be further improved by not pausing to clean the machine.

A pull-based management philosophy focuses on quality. A certain amount of meat is ground each day, not exceeding the rate that quality ground meat can be achieved. You can think of the meat as being pulled along at a certain pace or cadence. Management knows that to produce more ground meat, you need to use more grinding machines, or to develop new techniques for grinding and cleaning.

A helpdesk that is expected to complete all tickets by the end of each day is a push system. A helpdesk is a pull system if it accepts the n highest-priority tickets each week, or a certain number of Kanban cards per person each week. Another method is to accept a certain number of 30-minute appointments each day, as they do at the Apple Genius Bar.

1.1.3 Tickets and Kanban

Ticket systems and Kanban boards are not mutually exclusive. Many organizations use a ticket system for requests from customers and Kanban boards for project-related work. Some helpdesk automation systems are able to visually display the same set of requests as tickets in a queue or Kanban boards in columns. Kanban is a way to optimize workflow that can be applied to any kind of process.

1.2 Eliminating Time Sinkholes

Struggling teams always have a major time sinkhole that prevents them from getting around to the big, impactful projects that will fix the problem at its source. They're spending so much time mopping the floor that they don't have time to fix the leaking pipe. Often this sinkhole has been around so long that the team has become acclimated to it. They have become blind to its existence or believe that there is no solution.

Successful IT organizations are operating on a different level. They've eliminated the major time sinkholes and now have only minor inefficiencies or bottlenecks. They regularly identify the bottlenecks and assign people to eliminate them.

We could spend pages discussing how to identify your team's time sinkhole, but nine times out of ten it is OS installation and configuration. If it isn't that, it is the process of pushing new software releases into production, often called the

software delivery life cycle.

1.2.1 OS Installation and Configuration

We recently visited a site that was struggling. The site was a simple desktop environment with about 100 workstations (desktops and laptops), yet three three SAs were running around 50 hours a week and still unable to keep up with the flow of WIP. The source of the problem was that each workstation was configured slightly differently, causing a domino effect of other problems.

The definition of a broken machine is one that is in an unknown or undesired state. If each machine starts in an unknown state, then it is starting out broken. It cannot be fixed because fixing it means bringing it back to the original state, and we don't know what that original state is or should be. Maintaining a fleet of machines that are in unknown states is a fool's errand.

Each desktop was running the OS that had been preinstalled by the vendor. This was done, in theory, to save time. This made sense when the company had 3 desktops, but now it had 100. The SAs were now maintaining many different versions of Microsoft Windows, each installed slightly differently. This made each and every customer support request take twice as long, negating the time saved by using the vendor-installed OS. Any new software that was deployed would break in certain machines but not in others, making the SAs wary of deploying anything new.

If a machine's hard drive died, one SA would be occupied for an entire day replacing the disk, reloading the OS, and guessing which software packages needed to be installed for the customer. During one wave of malware attacks, the entire team was stuck wiping and reloading machines. All other work came to a halt. In the rush, no two machines turned out exactly the same, even if they were installed by the same person.

The SA team was always in firefighting mode. The SAs were always stressed out and overworked, which made them unhappy. Their customers were unhappy because their requests were being ignored by the overloaded SAs.

It became a vicious cycle: Nobody had time to solve the big problem because the big problem kept them from having time to work on it. This created stress and unhappiness, which made it more difficult to think about the big problems, which led to more stress and unhappiness.

Breaking the cycle was difficult. The company would have to assign one SA to focus exclusively on automating OS installations. This frightened the team because with one fewer person, more WIP would accumulate. The customers

might be even less happy. It took courageous management to make this decision.

It turns out that if customers are used to being ignored for days at a time, they can't tell if they're being ignored for slightly longer.

Soon OS installation was automated and a new batch of machines arrived. Their disks were wiped and OS reinstalled, starting all these machines in the same known state. They were more reliable because their hardware was newer but also because they could be supported better. Most importantly, setting up all these machines took hours instead of days.

Reinstalling a machine after a malware attack now took minutes. Moreover, because the new installation had properly installed anti-malware software, the machine stayed virus free.

Developers, being more technically inclined, were taught how to wipe and reinstall their own machines. Now, instead of waiting for an SA to do the task, developers were self-sufficient. This changed their workflows to include creating virtual machines on demand for testing purposes, which resulted in new testing methodologies that improved their ability to make quality software.

Soon the vicious cycle was broken and the SA team had more time for more important tasks. The fix didn't solve all their problems, but most of their problems couldn't be solved until they took care of this one major sinkhole.

Over time the old mix of configurations disappeared and all machines had the new, unified configuration. This uniformity made them, for the most part, interchangeable. In business terms the machines became fungible resources: Any one unit could be substituted for any other. Customers benefited from the ability to move between machines easily. Customer support improved as SAs focused all effort on a single configuration instead of having their attention diluted. Support requests were satisfied faster because less time was spent ascertaining what was unique about a machine. New applications could be introduced more rapidly because testing did not have to be repeated for each variation. The net effect was that all aspects of IT became higher quality and more efficient. There were fewer surprises.

If your team is suffering from this kind of time sinkhole and could benefit from better OS configuration and software installation methods, [Chapter 8, “OS Installation Strategies,”](#) provides more detail, especially [Section 8.4](#). You may also find motivation in the success story described in [Section 20.2](#).

1.2.2 Software Deployment

Once OS installation and configuration is conquered, the next most common

time sinkhole is deploying new software releases.

Deploying and updating software updates is a task much like the one in the story of Sisyphus in Greek mythology: rolling an immense boulder up a hill, only to watch it roll back down, and repeating this cycle for eternity. Not long after a software update is deployed, it seems as if a new update is inevitably announced.

Desktop software usually has a relatively simple upgrade process, which becomes more complex and laborious the larger the number of workstations a company has. Luckily it can be automated easily. This is the subject of [Chapter 7, “Workstation Software Life Cycle.”](#) For Microsoft Windows environments this typically means installing Windows Server Update Services (WSUS).

Server software has a more complex upgrade process. The more people who rely on the service, the more critical the service, and the more pressure for each upgrade to be a success. Extensive testing is required to detect and squash bugs before they reach production. More sophisticated upgrade processes are required to minimize or eliminate service disruptions. A service with ten users can be taken down for upgrades. A service with millions of users must be upgraded in place. This is like changing the tires on a car while it is barreling down the highway at full speed.

The phrase **software delivery life cycle (SDLC)** refers to the entire process of taking software from source code, through testing, packaging, and beta, and finally to installation in production. It requires a combined effort from developers and operations, which is where the term **DevOps** comes from. Each successful iteration through the SDLC results in a new software release running in production.

In some environments the SDLC process would be a hilarious month of missteps if it weren't happening to you. It is more likely to be a hellish month of stress and pain. In many organizations Hell Month arrives every 30 days.

Automating SDLC Phases

SDLC has the following three phases:

- 1. Integration:** Compiling the software, performing basic tests
- 2. Delivery:** Producing installable packages that are fully tested and ready to push into production, if the organization chooses to do so
- 3. Deployment:** Pushing the packages into a test or production environment

Each phase is predicated on the successful completion of the prior one. Each can be done manually or automated, or somewhere in between.

In continuous *integration* (CI), the automated process runs once for each change to the source code. In continuous *delivery* (CD), the delivery automation is triggered by any successful CI run. In continuous *deployment*, the deployment automation is triggered by any successful CD run. Continuous deployment is frequently used to push releases to a test environment, not all the way to production. Many companies, however, have improved their SDLC processes to the point where they can confidently use continuous deployment all the way to production.

A complete example of eliminating Hell Month can be found in [Section 2.2](#). This DevOps, or rapid-release, technique is covered in detail in [Chapter 20](#), “[Service Launch: DevOps](#).” Volume 2 of this book series is a deep dive into DevOps and service management in general. The definitive book on the topic is *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation* by Humble & Farley ([2010](#)).

1.3 DevOps

The DevOps movement is focused on making the SDLC as smooth and unhellish as possible. A highly automated SDLC process brings with it a new level of confidence in the company’s ability to successfully launch new software releases. This confidence means a company can push new releases into production more frequently, eliminating the friction that prevents companies from experimenting and trying new things. This ability to experiment then makes the company more competitive as new features appear more frequently, bugs disappear faster, and even minor annoyances are quickly alleviated. It enables a culture of innovation.

At the same time it makes life better for SAs. After working in an environment where such stress has been eliminated, people do not want to work anywhere else. It is as if a well-run DevOps environment is not just a business accelerator, but a fringe benefit for the employees. As author and DevOps advocate Gene Kim wrote, “The opposite of DevOps is despair.”

1.4 DevOps Without Devs

Organizations without developers will benefit from the DevOps principles, too. They apply to any complex process, and the system administration field is full of

them. The DevOps principles are as follows:

- **Turn chaotic processes into repeatable, measurable ones.** Processes should be documented until they are consistent, then automated to make them efficient and self-service. To make IT scale we must not do IT tasks but be the people who maintain the system that does IT tasks for us.
- **Talk about problems and issues within and between teams.** Don't hide, obscure, muddle through, or silently suffer through problems. Have channels for communicating problems and issues. Processes should amplify feedback and encourage communication across silos.
- **Try new things, measure the results, keep the successes, and learn from the failures.** Create a culture of experimentation and learning by using technology that empowers creativity and management practices that are blameless and reward learning. This is also called "Start, Stop, Continue, Change" management. Start doing the things you need to do. Stop doing things that don't work well. Continue doing things that do work well. Change the things that need improving.
- **Do work in small batches so we can learn and pivot along the way.** It is better to deliver some results each day than to hold everything back and deliver the entire result at the end. This strategy enables us to get feedback sooner, fix problems, and avoid wasted effort.
- **Drive configuration and infrastructure from machine-readable sources kept under a source code control system.** When we treat infrastructure as code (IaC), it becomes flexible and testable, and can benefit from techniques borrowed from software engineering.
- **Always be improving.** We are always identifying the next big bottleneck and experimenting to find ways to fix it. We do not wait for the next major disaster.
- **Pull, don't push.** Build processes to meet demand, not accommodate supply. Determine the desired weekly output, allocate the required resources, and pull the work through the system to completion.
- **Build community.** The IT department is part of the larger organism that is our company and we must be an active participant in its success. Likewise, we are part of the world's IT community and have a duty to be involved in sharing knowledge and encouraging professional development.

You'll see the DevOps principles demonstrated throughout this book, whether or not developers are involved. The best introduction to DevOps is *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win* by Kim,

Behr & Spafford ([2013](#)). We also recommend *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations* by Kim, Humble, Debois & Willis ([2016](#)) as a way to turn these ideas into action.

1.5 Bottlenecks

If OS installation and software deployment are not your time sinkhole, how can you identify what is?

Bring up the subject at lunch or a team meeting, and usually just a short amount of discussion will identify the biggest pain points, time wasters, inefficiencies, and so on. Solicit opinions from the stakeholders and teams you partner with. They already know your biggest problem.

A more scientific approach is to identify the bottleneck. In operational theory, the term **bottleneck** refers to the point in a system where WIP accumulates. Following are some examples.

Suppose your team is involved in deploying new PCs. Customers request them, and the hardware is subsequently purchased. Once they arrive the OS and applications are installed, and finally the new machine is delivered to the customer. If employees are standing in line at someone's desk waiting to talk to that person about ordering a machine, the bottleneck is the ordering process. If customers are waiting weeks for the hardware to arrive, the bottleneck is in the purchasing process. If purchased machines sit in their boxes waiting to be installed, the bottleneck is the installation step.

A team that produces a web-based application has different members performing each task: writing the code, testing it, deploying it to a beta environment, and deploying it to the production environment. Watch what happens when a bug is reported. If it sits in the bug tracking system, untouched and unassigned to anyone, then the bottleneck is the triage process. If it gets assigned to someone but that person doesn't work on the issue, the bottleneck is at the developer step. If the buggy code is fixed quickly but sits unused because it hasn't been put into testing or production, then the bottleneck is in the testing or deployment phase.

Once the bottleneck is identified, it is important to focus on fixing the bottleneck itself. There may be problems and things we can improve all throughout the system, but directing effort toward anything other than optimizations at the bottleneck will not help the total throughput of the system. Optimizations prior to the bottleneck simply make WIP accumulate faster at the

bottleneck. Optimizations after the bottleneck simply improve the part of the process that is starved for work.

Take the PC deployment example. Suppose the bottleneck is at the OS installation step. A technician can install and configure the OS on ten machines each week, but more than ten machines are requested each week. In this case, we will see the requests accumulate at the point the machines are waiting to have their OS installed.

Writing a web-based application to manage the requests will not change the number of machines installed each week. It is an improvement that people will like, it superficially enhances the system and it may even be fun to code, but, sadly, it will not change the fact that ten machines are being completed each week. Likewise, we could hire more people to deliver the machines to people's offices, but we know that downstream steps are idle, starved for work. Therefore, those improvements would be silly.

Only by expending effort at the bottleneck will the system complete more machines per week. In this case, we could hire more people or automate the process. You can guess what the authors would do.

Now imagine, instead, the problem is in the purchasing process. The hardware vendor loses orders, confuses orders, and frequently delivers the wrong types of machines. The person in charge of purchasing is spending so much time fixing these problems that orders are accumulating.

On the one hand, you could use your time to sit down with the purchasing person and the hardware vendor to figure out why orders keep getting lost. On the other hand, there is a minor technical issue with the OS installation process that makes the process take an extra 30 seconds. Fixing this problem has the benefit of sweet, sweet, instant gratification and should take a minute to fix. Of course, we know that nothing in IT actually takes a minute. Soon you will have spent half a day on the problem. That half a day could have been spent fixing the bottleneck.

The moral of this story is that you need to suppress the desire for instant gratification or the draw of superficial solutions and focus on the bottleneck itself.

High-performing teams minimize time spent firefighting. Instead, they are in a continuous loop of measuring their system, identifying bottlenecks, and fixing them. Unplanned WIP and emergencies are the exception rather than the rule.

Fixing the Biggest Time Drain

When Tom worked for Cibernet, he found that the company's London SA team was prevented from making any progress on critical, high-priority projects because it was drowning in requests for help with people's individual desktop PCs.

The senior SAs were overloaded, but hiring an additional senior SA would not work. Ramp-up time would be too long. Instead, he needed to free up their time by taking away tasks that could be done by someone else—for example, general desktop support requests. Entry-level Windows desktop support technicians were plentiful and inexpensive and wouldn't require much training. Management wouldn't let him hire such a person but finally agreed to bring someone in on a temporary six-month contract.

With that person handling the generic desktop problems—malware cleanup, new PC deployment, password resets, and so on—the remaining SAs were freed to complete the high-priority projects that were key to the company.

By the end of the six-month contract, management could see the improvement in the SAs' performance. The senior SAs finally had time to climb out of the hole and get the big projects done. The contract for the temporary employee was extended and later made permanent when management saw the benefit of specialization.

1.6 Getting Started

We've loaded you with a lot of great ideas and best practices. You even know how to identify the biggest, most pressing bottleneck. So how do you fix any of this? How do you get started? How can you convince your co-workers, team, or manager to adopt any of these best practices?

People are often surprised to find that no matter how bad your environment is, co-workers will resist change. Your co-workers may fear change, as they have become comfortable with how things are now. Suggesting change may be insulting to the co-workers who built the system you want to fix. Your boss may not love the idea of you spending time on projects other than the burning fires you were hired to put out.

Our answer is to start small. Don't try to fix everything at once. Pick one achievable goal, complete it, and repeat.

- Small changes need less approval. Big changes require a PowerPoint

presentation, agreement from the team, and three levels of management. Small changes merit a casual suggestion that is approved with a nod. Pick the smallest increment of work that creates meaningful change.

- Small changes get less resistance. Peers resist big changes that threaten the status quo. People like small changes that they completely understand, feel can be achieved, and fix pain points that affect them personally (but mostly the latter).
- Small successes breed confidence and make it easier to get approval for the next proposal.
- Small changes deliver results faster. It is better to fix a part of the problem now than have a complete solution later. Release version 1.0, which automates one aspect of the task, so that the entire team can benefit from this automation. A week later release version 1.1, which automates one more special case. During that last week, your co-workers have benefited from your work. Imagine if, instead, you worked for a year before sharing your results.
- Small changes deal with the unexpected better. If a small project is cancelled, the half-finished project may have cost only a day's work. A big project that is cancelled wastes weeks or months of effort.
- Small changes invite others to get involved. If you automate the OS installation's middle step in a way that works for desktops but not laptops, you've opened the door for other people to join in and fill in the missing parts.

Over time, small changes build to become big successes.

1.7 Summary

The remainder of this book is full of lofty and sometimes idealistic goals for an SA organization. This chapter looked at some high-impact changes that a site can make if it is drowning in problems.

There needs to be a way to manage work in progress, in particular requests from customers. Customers are the people we serve (often referred to as users). Using a request-tracking system to manage WIP means that the SAs spend less time tracking the requests and gives customers a better sense of the statuses of their requests. A request-tracking system improves SAs' ability to have good follow-through on customers' requests. Kanban is an alternative that is better suited for teams that have more project-oriented work, but supports customer requests by using a dedicated swimlane. Kanban creates a drumbeat, or cadence,

that creates a smooth flow of progress. It is a process that lets us measure progress and optimize it instead of creating a high-pressure push paradigm.

There should be a way to manage quick requests properly, so that customer expectations are met, while still leaving SAs with uninterrupted time for project work. Designating a person, a rotation, or a subteam of people to handle these requests permits other team members to focus on long-term project work.

The biggest time sinkhole must be eliminated. If a site doesn't have an automated way to install and configure machines, it will make all other tasks more difficult. Machines will be more difficult to support because each is slightly different. It will be difficult to deploy new services because all variations cannot be tested. It will be impossible to fix a machine because we don't know what a correct machine looks like. By automating the OS installation and configuration we begin all machines in the same state. By automating software updates we maintain the machines in a way that scales.

Another common time sinkhole is software deployment. The software delivery life cycle is the end-to-end process of taking software from source code, through testing, packaging, and beta, and finally to installation in production. In struggling organizations, these steps are done manually and become an all-consuming, painful process. Organizations that have automated the end-to-end process not only eliminate a lot of grief, but also enable the company to ship new releases more frequently with greater confidence. This confidence permits companies to be more aggressive about adding new features and fixing problems.

DevOps is a set of ideals and principles that apply to how companies manage and improve any complex IT process. Often thought of as a way to improve the SDLC, it is relevant for all IT processes.

Successful IT organizations focus their attention on optimizing bottlenecks in their systems. A bottleneck is where WIP accumulates. Improvements to a system anywhere other than at the bottleneck may be emotionally satisfying, but they do not improve the performance of the system.

No matter which kind of improvement you want to make, there will be organizational resistance. It is best to start with small changes rather than try to fix and change everything at once. Small changes receive less resistance, are easier to accomplish, and deliver results sooner. It is better to deliver small improvements today than big improvements tomorrow. Pick one achievable goal, complete it, and repeat. Over time you will build momentum, gain allies, and be able to make larger changes with more confidence.

Exercises

- 1.** What is WIP?
- 2.** How does a ticket system improve our ability to track WIP?
- 3.** How does Kanban work?
- 4.** Compare and contrast a ticket system and Kanban.
- 5.** How do you track WIP in your environment? What do you like or dislike about it? What you would like to improve?
- 6.** In your organization, how do you ensure that SAs follow through on requests?
- 7.** In your organization, if a piece of WIP has been ignored or progress has stalled, will this be noticed? If not, how could you fix this?
- 8.** Why is it important to automate OS installation?
- 9.** List the operating systems supported in your environment in order of popularity. Which automation is used to install each? Of those that aren't automated, which would benefit the most from it?
- 10.** For an OS that does not have an automated installation process, which product is available that will automate it?
- 11.** What's the biggest time sinkhole in your environment? Name two ways to eliminate it.
- 12.** What's the biggest bottleneck in your environment? What can be done to optimize or eliminate it?

Chapter 2. The Small Batches Principle

One of the themes you will see in this book is the small batches principle: It is better to do work in small batches than in big leaps. Small batches permit us to deliver results faster, with higher quality and less stress.

This chapter begins with an example that has nothing to do with system administration to demonstrate the general idea. Then it focuses on three IT-specific examples to show how the method applies and the benefits that follow.

The small batches principle is part of the DevOps methodology. It comes from the Lean Manufacturing movement, which is often called just-in-time (JIT) manufacturing. It can be applied to just about any kind of process that you do frequently. It also enables the minimum viable product (MVP) methodology, which involves launching a small version of a service to get early feedback that informs the decisions made later in the project.

2.1 The Carpenter Analogy

Imagine a carpenter who needs 50 two-by-fours, all the same length.

One could imagine cutting all 50 boards and then measuring them to verify that they are all the correct size. It would be very disappointing to discover that the blade shifted while making board 10, and boards 11 through 50 are now unusable. The carpenter would have to cut 40 new boards. How embarrassing!

A better method would be to verify the length after each board is cut. If the blade has shifted, the carpenter will detect the problem soon after it happened, and there would be less waste.

These two approaches demonstrate big batches versus small batches. In the big-batch world, the work is done in two large batches: The carpenter cuts all the boards, then inspects all the boards. In the small-batch world, there are many iterations of the entire process: cut and inspect, cut and inspect, cut and inspect

One benefit of the small-batch approach is less waste. Because an error or defect is caught immediately, the problem can be fixed before it affects other parts.

A less obvious benefit is latency. At the construction site there is a second team of carpenters who use the boards to build a house. The boards cannot be used until they are inspected. With the first method, the second team cannot

begin its work until all the boards are cut and at least one is inspected. The chances are high that the boards will be delivered in a big batch after they have all been inspected. With the small-batch method, the new boards are delivered without this delay.

The sections that follow relate the small batches principle to system administration and show many benefits beyond reduced waste and improved latency.

2.2 Fixing Hell Month

A company had a team of software developers who produced a new release every six months. When a release shipped, the operations team stopped everything and deployed the release into production. The process took three or four weeks and was very stressful for all involved. Scheduling the maintenance window required complex negotiation. Testing each release was complex and required all hands on deck. The actual software installation never worked on the first try. Once it was deployed, a number of high-priority bugs would be discovered, and each would be fixed by various “hot patches” that would follow.

Even though the deployment process was labor intensive, there was no attempt to automate it. The team had many rationalizations that justified this omission. The production infrastructure changed significantly between releases, making each release a moving target. It was believed that any automation would be useless by the next release because each release’s installation instructions were shockingly different. With each next release being so far away, there was always a more important “burning issue” that had to be worked on first. Thus, those who did want to automate the process were told to wait until tomorrow, and tomorrow never came. Lastly, everyone secretly hoped that maybe, just maybe, the next release cycle wouldn’t be so bad. Such optimism is a triumph of hope over experience.

Each release was a stressful, painful month for all involved. Soon it was known as Hell Month. To make matters worse, each release was usually late. This made it impossible for the operations team to plan ahead. In particular, it was difficult to schedule any vacation time, which just made everyone more stressed and unhappy.

Feeling compassion for the team’s woes, someone proposed that releases should be done less often, perhaps every 9 or 12 months. If something is painful, it is natural to want to do it less frequently.

To everyone’s surprise the operations team suggested going in the other

direction: monthly releases. This was a big-batch situation. To improve, the company didn't need bigger batches, it needed smaller ones.

People were shocked! Were they proposing that every month be Hell Month? No, by doing it more frequently, there would be pressure to automate the process. If something happens infrequently, there's less urgency to automate it, and we procrastinate. Also, there would be fewer changes to the infrastructure between each release. If an infrastructure change did break the release automation, it would be easier to fix the problem.

The change did not happen overnight. First the developers changed their methodology from mega-releases, with many new features, to small iterations, each with a few specific new features. This was a big change, and selling the idea to the team and management was a long process.

Meanwhile, the operations team automated the testing and deployment processes. The automation could take the latest code, test it, and deploy it into the beta-test area in less than an hour. Pushing code to production was still manual, but by reusing code for the beta rollouts it became increasingly less manual over time.

The result was that the beta area was updated multiple times a day. Since it was automated, there was little reason not to. This made the process continuous, instead of periodic. Each code change triggered the full testing suite, and problems were found in minutes rather than in months.

Pushes to the production area happened monthly because they required coordination among engineering, marketing, sales, customer support, and other groups. That said, all of these teams loved the transition from an unreliable hopefully every-six-months schedule to a reliable monthly schedule. Soon these teams started initiatives to attempt weekly releases, with hopes of moving to daily releases. In the new small-batch world, the following benefits were observed:

- **Features arrived faster.** Where in the past a new feature took up to six months to reach production, now it could go from idea to production in days.
- **Hell Month was eliminated.** After hundreds of trouble-free pushes to beta, pushing to production was easier than ever.
- **The operations team could focus on higher-priority projects.** The operations team was no longer directly involved in software releases other than fixing the automation, which was rare. This freed up the team for more important projects.

- **There were fewer impediments to fixing bugs.** The first step in fixing a bug is to identify which code change is responsible. Big-batch releases had hundreds or thousands of changes to sort through to identify the guilty party. With small batches, it was usually quite obvious where to find the bug.
- **Bugs were fixed in less time.** Fixing a bug in code that was written six months ago is much more difficult than if the code is still fresh in your mind. Small batches meant bugs were reported soon after the code was written, which meant developers could fix it more expertly in a shorter amount of time.
- **Developers experienced instant gratification.** Waiting six months to see the results of your efforts is demoralizing. Seeing your code help people shortly after it was written is addictive.
- **Everyone was less stressed.** Most importantly, the operations team could finally take long vacations, the kind that require advance planning and scheduling, thus giving them a way to reset and live healthier lives.

While these technical benefits were worthwhile, the business benefits were even more exciting:

- **Improved ability to compete:** Confidence in the ability to add features and fix bugs led to the company becoming more aggressive about new features and fine-tuning existing ones. Customers noticed and sales improved.
- **Fewer missed opportunities:** The sales team had been turning away business due to the company's inability to strike fast and take advantage of opportunities as they arrived. Now the company could enter markets it hadn't previously imagined.
- **A culture of automation and optimization:** Rapid releases removed common excuses not to automate. New automation brought consistency, repeatability, and better error checking, and required less manual labor. Plus, automation could run anytime, not just when the operations team was available.

The ability to do rapid releases is often called a DevOps strategy. In [Chapter 20](#), “[Service Launch: DevOps](#),” you’ll see similar strategies applied to third-party software.

The Inner and Outer Release Loops

You can think of this process as two nested loops. The inner loop is the

code changes done one at a time. The outer loop is the releases that move these changes to production.

2.3 Improving Emergency Failovers

Stack Overflow's main web site infrastructure is in a datacenter in New York City. If the datacenter fails or needs to be taken down for maintenance, duplicate equipment and software are running in Colorado. The duplicate in Colorado is a running and functional copy, except that it is in stand-by mode waiting to be activated. Database updates in NYC are replicated to Colorado. A planned switch to Colorado will result in no lost data. In the event of an unplanned failover—for example, as the result of a power outage—the system will lose an acceptably small quantity of updates.

The failover process is complex. Database masters need to be transitioned. Services need to be reconfigured. It takes a long time and requires skills from four different teams. Every time the process happens, it fails in new and exciting ways, requiring ad hoc solutions invented by whoever is doing the procedure.

In other words, the failover process is risky. When Tom was hired at Stack Overflow, his first thought was, “I hope I’m not on call when we have that kind of emergency.”

Drunk driving is risky, so we avoid doing it. Failovers are risky, so we should avoid them, too. Right?

Wrong. There is a difference between behavior and process. Risky *behaviors* are inherently risky; they cannot be made less risky. Drunk driving is a risky *behavior*. It cannot be done safely, only avoided. A failover is a risky *process*. A risky *process* can be made less risky by doing it more often.

The next time a failover was attempted at Stack Overflow, it took ten hours. The infrastructure in New York had diverged from Colorado significantly. Code that was supposed to seamlessly fail over had been tested only in isolation and failed when used in a real environment. Unexpected dependencies were discovered, in some cases creating Catch-22 situations that had to be resolved in the heat of the moment.

This ten-hour ordeal was the result of big batches. Because failovers happened rarely, there was an accumulation of infrastructure skew, dependencies, and stale code. There was also an accumulation of ignorance: New hires had never experienced the process; others had fallen out of practice.

To fix this problem the team decided to do more failovers. The batch size was

based on the number of accumulated changes and other things that led to problems during a failover. Rather than let the batch size grow and grow, the team decided to keep it small. Rather than waiting for the next real disaster to exercise the failover process, they would introduce simulated disasters.

The concept of activating the failover procedure on a system that was working perfectly might seem odd, but it is better to discover bugs and other problems in a controlled situation rather than during an emergency. Discovering a bug during an emergency at 4 AM is troublesome because those who can fix it may be unavailable—and if they are available, they're certainly unhappy to be awakened. In other words, it is better to discover a problem on Saturday at 10 AM when everyone is awake, available, and presumably sober.

If schoolchildren can do fire drills once a month, certainly system administrators can practice failovers a few times a year. The team began doing failover drills every two months until the process was perfected.

Each drill surfaced problems with code, documentation, and procedures. Each issue was filed as a bug and was fixed before the next drill. The next failover took five hours, then two hours, then eventually the drills could be done in an hour with no user-visible downtime.

The drills found infrastructure changes that had not been replicated in Colorado and code that didn't fail over properly. They identified new services that hadn't been engineered for smooth failover. They discovered a process that could be done by one particular engineer. If he was on vacation or unavailable, the company would be in trouble. He was a single point of failure.

Over the course of a year all these issues were fixed. Code was changed, better pretests were developed, and drills gave each member of the SRE (site reliability engineering) team a chance to learn the process. Eventually the overall process was simplified and easier to automate. The benefits Stack Overflow observed included

- **Fewer surprises:** More frequent the drills made the process smoother.
- **Reduced risk:** The procedure was more reliable because there were fewer hidden bugs waiting to bite.
- **Higher confidence:** The company had more confidence in the process, which meant the team could now focus on more important issues.
- **Speedier bug fixes:** The smaller accumulation of infrastructure and code changes meant each drill tested fewer changes. Bugs were easier to identify and faster to fix.

- **Less stressful debugging:** Bugs were more frequently fixed during business hours. Instead of having to find workarounds or implement fixes at odd hours when engineers were sleepy, they were worked on during the day when engineers were there to discuss and implement higher-quality fixes.
- **Better cross-training:** Practice makes perfect. Operations team members all had a turn at doing the process in an environment where they had help readily available. No person was a single point of failure.
- **Improved process documentation and automation:** The team improved documentation in real time as the drill was in progress. Automation was easier to write because the repetition helped the team see what could be automated and which pieces were most worth automating.
- **Revealed opportunities:** The drills were a big source of inspiration for big-picture projects that would radically improve operations.
- **Happier developers:** There was less chance of being woken up at odd hours.
- **Happier operations team:** The fear of failovers was reduced, leading to less stress. More people trained in the failover procedure meant less stress on the people who had previously been single points of failure.
- **Better morale:** Employees could schedule long vacations again.

Google's Forced Downtime

Google's internal lock service is called Chubby; an open source clone is called Zookeeper. Chubby's uptime was so perfect that engineers designing systems that relied on Chubby started writing code that assumed Chubby could not go down. This led to cascading failures when Chubby did have an outage.

To solve this problem, Google management decreed that if Chubby had zero downtime in a given month, it would be taken down intentionally for five minutes. This would assure that error handling code was exercised regularly.

Developers were given three months' warning, yet the first "purposeful outage" was postponed when a team came forward to beg for an extension. One was granted, but since then Chubby has been down for five minutes every month to exercise the failure-related code.

2.4 Launching Early and Often

An IT department needed a monitoring system. The number of servers had grown to the point where situational awareness was no longer possible by manual means. The lack of visibility into the company's own network meant that outages were often first reported by customers, and often after the outage had been going on for hours and sometimes days.

The system administration team had a big vision for what the new monitoring system would be like. All services and networks would be monitored, the monitoring system would run on a pair of big, beefy machines, and when problems were detected a sophisticated oncall schedule would be used to determine whom to notify.

Six months into the project they had no monitoring system. The team was caught in endless debates over every design decision: monitoring strategy, how to monitor certain services, how the pager rotation would be handled, and so on. The hardware cost alone was high enough to require multiple levels of approval.

Logically the monitoring system couldn't be built until the planning was done, but sadly it looked like the planning would never end. The more the plans were discussed, the more issues were raised that needed to be discussed. The longer the planning lasted, the less likely the project would come to fruition.

Fundamentally they were having a big-batch problem. They wanted to build the perfect monitoring system in one big batch. This is unrealistic.

The team adopted a new strategy: small batches. Rather than building the perfect system, they would build a small system and evolve it.

At each step they would be able to show it to their co-workers and customers to get feedback. They could validate assumptions for real, finally putting a stop to the endless debates the requirements documents were producing. By monitoring something—anything—they would learn the reality of what worked best.

Small systems are more flexible and malleable; therefore, experiments are easier. Some experiments would work well; others wouldn't. Because they would keep things small and flexible, it would be easy to throw away the mistakes.

This would enable the team to pivot. **Pivot** means to change direction based on recent results. It is better to pivot early in the development process than to realize well into it that you've built something that nobody likes.

Google calls this idea "launch early and often." Launch as early as possible,

even if that means leaving out most of the features and launching to only a few users. What you learn from the early launches informs the decisions later on and produces a better service in the end.

Launching early and often also gives you the opportunity to build operational infrastructure early. Some companies build a service for a year and then launch it, informing the operations team only a week prior. IT then has little time to develop operational practices such as backups, oncall playbooks, and so on. Therefore, those things are done badly. With the launch-early-and-often strategy, you gain operational experience early and you have enough time to do it right.

Fail Early and Often

The strategy of “launch early and often” is sometimes called “fail early and often.” Early launches are so experimental that it is highly likely they will fail. That is considered a good thing. Small failures are okay, as long as you learn from them and they lead to future success. If you never fail, you aren’t taking enough risks (the good kind) and you are missing opportunities to learn.

It is wasteful to discover that your base assumptions were wrong after months of development. By proving or disproving our assumptions as soon as possible, we have more success in the end.

Launching early and often is also known as the **minimum viable product (MVP)** strategy. As defined by Eric Ries, “The minimum viable product is that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort” ([Ries 2009](#)). In other words, rather than focusing on new functionality in each release, focus on testing an assumption in each release.

The team building the monitoring system adopted the launch-early-and-often strategy. They decided that each iteration, or small batch, would be one week long. At the end of the week they would release what was running in their beta environment to their production environment and ask for feedback from stakeholders.

For this strategy to work they had to pick very small chunks of work. Taking a cue from Jason Punyon and Kevin Montrose’s “Providence: Failure Is Always an Option” ([Punyon 2015](#)), they called this “What can get done by Friday?”–driven development.

Iteration 1 had the goal of monitoring a few servers to get feedback from

various stakeholders. The team installed an open source monitoring system on a virtual machine. This was in sharp contrast to their original plan of a system that would be highly scalable. Virtual machines have less I/O and network horsepower than physical machines. Hardware could not be ordered and delivered in a one-week time frame, however, so the first iteration used virtual machines. It was what could be done by Friday.

At the end of this iteration, the team didn't have their dream monitoring system, but they had more monitoring capability than ever before.

In this iteration they learned that Simple Network Management Protocol (SNMP) was disabled on most of the organization's networking equipment. They would have to coordinate with the network team if they were to collect network utilization and other statistics. It was better to learn this now than to have their major deployment scuttled by making this discovery during the final big deployment. To work around this, the team decided to focus on monitoring things they did control, such as servers and services. This gave the network team time to create and implement a project to enable SNMP in a secure and tested way.

Iterations 2 and 3 proceeded well, adding more machines and testing other configuration options and features.

During iteration 4, however, the team noticed that the other system administrators and managers hadn't been using the system much. This was worrisome. They paused to talk one on one with people to get some honest feedback.

What the team learned was that without the ability to have dashboards that displayed historical data, the system wasn't very useful to its users. In all the past debates this issue had never been raised. Most confessed they hadn't thought it would be important until they saw the system running; others hadn't raised the issue because they simply assumed all monitoring systems had dashboards.

It was time to pivot.

The software package that had been the team's second choice had very sophisticated dashboard capabilities. More importantly, dashboards could be configured and customized by individual users. Dashboards were self-service.

After much discussion, the team decided to pivot to the other software package.

In the next iteration, they set up the new software and created an equivalent set of configurations. This went very quickly because a lot of work from the

previous iterations could be reused: the decisions on what and how to monitor, the work completed with the network team, and so on.

By iteration 6, the entire team was actively using the new software. Managers were setting up dashboards to display key metrics that were important to them. People were enthusiastic about the new system.

Something interesting happened around this time: A major server crashed on Saturday morning. The monitoring system alerted the SA team, who were able to fix the problem before people arrived at the office on Monday. In the past there had been similar outages but repairs had not begun until the SAs arrived on Monday morning, well after most employees had arrived. This showed management, in a very tangible way, the value of the system.

Iteration 7 had the goal of writing a proposal to move the monitoring system to physical machines so that it would scale better. By this time the managers who would approve such a purchase were enthusiastically using the system; many had become quite expert at creating custom dashboards. The case was made to move the system to physical hardware for better scaling and performance, plus a duplicate set of hardware would be used for a hot spare site in another datacenter. The plan was approved.

In future iterations the system became more valuable to the organization as the team implemented features such as a more sophisticated oncall schedule, more monitored services, and so on. The benefits of small batches observed by the SA team included:

- **Testing assumptions early prevents wasted effort.** The ability to fail early and often means we can pivot. Problems can be fixed sooner rather than later.
- **Providing value earlier builds momentum.** People would rather have some features today than all the features tomorrow. Some monitoring is better than no monitoring. The naysayers see results and become advocates. Management has an easier time approving something that isn't hypothetical.
- **Experimentation is easier.** Often, people develop an emotional attachment to code. With small batches we can be more agile because we haven't yet grown attached to our past decisions.
- **MVP enables instant gratification.** The team saw the results of their work faster, which improved morale.
- **The team was less stressed.** There is no big, scary due date, just a constant flow of new features.

- **Big-batch debating is procrastination.** Much of the early debate had been about details and features that didn't matter or didn't get implemented.

The first few weeks were the hardest. The initial configuration required special skills. Once it was running, however, people with less technical skill or desire could add rules and make dashboards. In other words, by taking a lead and setting up the scaffolding, others can follow. This is an important point of technical leadership. Technical leadership means going first and making it easy for others to follow.

A benefit of using the MVP model is that the system is always working or in a shippable state. The system is always providing benefit, even if not all the features are delivered. Therefore, if more urgent projects take the team away, the system is still usable and running. If the original big-batch plan had continued, the appearance of a more urgent project might have left the system half-developed and unlaunched. The work done so far would have been for naught.

Another thing the team realized during this process was that not all launches were of equal value. Some launches were significant because of the time and work put into them, but included only internal changes and scaffolding. Other launches included features that were tangibly useful to the primary users of the system. Launches in this latter category were the only thing that mattered to management when they measured progress. It was most important to work toward goals that produced features that would be visibly helpful and meaningful to people outside of the team.

Thanksgiving

The U.S. Thanksgiving holiday involves a large feast. If you are not used to cooking a large meal for many people, this once-a-year event can be a stressful, scary time. Any mistakes are magnified by their visibility: All your relatives are there to see you fail. It is a big batch.

You can turn this into a small batch by trying new recipes in the weeks ahead of time, by attempting test runs of large items, or by having the event be potluck. These techniques reduce risk and stress for all involved.

2.5 Summary

Why are small batches better?

Small batches result in happier customers. Features get delivered sooner. Bugs

are fixed faster.

Small batches reduce risk. By testing assumptions, the prospect of future failure is reduced. More people get experience with procedures, which means their skills are improved.

Small batches reduce waste. They avoid endless debates and perfectionism that delay the team in getting started. Less time is spent implementing features that don't get used. In the event that higher-priority projects come up, the team has already delivered a usable system.

Small batches encourage experimentation. We can try new things—even crazy ideas, some of which turn into competition-killing features. We fear failure less because we can undo a small batch easily if the experiment fails. More importantly, we learned something that will help us make future improvements.

Small batches improve the ability to innovate. Because experimentation is encouraged, we test new ideas and keep only the good ones. We can take risks. We are less attached to old pieces that must be thrown away.

Small batches improve productivity. Bugs are fixed more quickly and the process of fixing them is accelerated because the code is fresher in our minds.

Small batches encourage automation. When something must happen often, excuses not to automate go away.

Small batches make system administrators happier. We get instant gratification and Hell Month disappears. It is just simply a better way to work.

The small batches principle is an important part of the DevOps methodology and applies whether you are directly involved in a development process, making a risky process go more smoothly, deploying an externally developed package, or cooking a large meal.

Exercises

1. What is the small batches principle?
2. Why are big batches more risky than small batches?
3. Why is it better to push a new software release into production monthly rather than every six months?
4. Pick a number of software projects you are or have been involved in. How frequently were new releases issued? Compare and contrast the projects' ability to address bugs and ship new features.
5. Is it better to fail over or take down a perfectly running system than to wait until it fails on its own?

- 6.** What is the difference between behavior and process?
- 7.** Why is it better to have a small improvement now than a large improvement a year from now?
- 8.** Describe the minimum viable product (MVP) strategy. What are the benefits of it versus a larger, multi-year project plan?
- 9.** List a number of risky behaviors that cannot be improved through practice. Why are they inherently risky?
- 10.** Which big-batch releases happen in your environment? Describe them in detail.
- 11.** Pick a project that you are involved in. How could it be restructured so that people benefit immediately instead of waiting for the entire project to be complete?

Chapter 3. Pets and Cattle

This chapter is about improving our efficiency by minimizing variation. We mass-produce our work by unifying like things so that they can be treated the same. As a result we have fewer variations to test, easier customer support, and less infrastructure to maintain. We scale ourselves. We can't eliminate all variation, but the more we can unify, the more efficient we can be. Managing the remaining variation is the topic of the next chapter. For now, let's focus on unification itself.

We can't spend hours custom-building every machine we install. Instead, we make our machines generic so that they can all be treated as similarly as possible. Likewise, we are more efficient when we treat related tasks the same way. For example, the process of onboarding new employees usually involves creating accounts and supplying hardware to the new hires. If we invent the process anew with each employee, it not only takes longer but also looks unprofessional as we stumble through improvising each step as the new hires wait. People appreciate a process that is fast, efficient, and well executed.

It is difficult to get better at a process when we never do the same thing more than once. Improvement comes from repetition; practice makes perfect. The more we can consolidate similar things so they can be treated the same, the more practice we get and the better we get at it.

3.1 The Pets and Cattle Analogy

The machines that we administer range from highly customized to entirely generic. The analogy commonly used is “pets and cattle.” Pets are the highly customized machines and cattle are the generic machines.

This analogy is generally attributed to Yale computer scientist David Gelernter, who used it in reference to filesystems. Gelernter wrote, “If you have three pet dogs, give them names. If you have 10,000 head of cattle, don't bother.”

The analogy gained in popularity when Joshua McKenty, co-founder of Piston Cloud, explained it this way in a press release ([McKenty 2013](#)):

The servers in today's datacenter are like puppies—they've got names and when they get sick, everything grinds to a halt while you nurse them back to health. . . . Piston Enterprise OpenStack is a system for managing your servers like cattle—you number them, and when they get

sick and you have to shoot them in the head, the herd can keep moving. It takes a family of three to care for a single puppy, but a few cowboys can drive tens of thousands of cows over great distances, all while drinking whiskey.

A pet is a unique creature. It is an animal that we love and take care of. We take responsibility for its health and well-being. There is a certain level of emotional attachment to it. We learn which food it likes and prepare special meals for it. We celebrate its birthdays and dress it up in cute outfits. If it gets injured, we are sad. When it is ill, we take it to the veterinarian and give it our full attention until it is healed. This individualized care can be expensive. However, since we have only one or two pets, the expense is justified.

Likewise, a machine can be a pet if it is highly customized and requires special procedures for maintaining it.

A herd of cattle is a group of many similar animals. If you have a herd of cows each one is treated the same. This permits us the benefits of mass-production. All cattle receive the same living conditions, the same food, the same medical treatment, the same everything. They all have the same personality, or at least are treated as if they do. There are no cute outfits. The use of mass-production techniques keeps maintenance costs low and improves profits at scale: Saving a dollar per cow can multiply to hundreds of thousands in total savings.

Likewise, machines can be considered cattle when they are similar enough that they can all be managed the same way. This can be done at different levels of abstraction. For example, perhaps the OS is treated generically even though the hardware may comprise any number of virtual or physical machine configurations. Or perhaps the machine hardware, OS, and applications are all the same, but the data they access is different. This is typical in a large web hosting farm, where the only difference is which specific web site is being served by each machine.

Preferably the systems we deal with are fungible resources: Any one unit can substitute for any other.

A related metaphor is the snowflake. A snowflake is even more unique than a pet. It is one of a kind. A system may have started out similar to others, but it was customized, modified, and eventually becomes unlike any other system. Or maybe it started out unique and had very little chance of being properly brought into line with the others. A snowflake requires special operational procedures. Rebooting it requires extra care. Upgrades require special testing. As Martin Fowler ([2012](#)) wrote, a snowflake is “good for a ski resort, bad for a datacenter.”

A snowflake server is a business risk because it is difficult to reproduce. If the hardware fails or the software becomes corrupted, it would be difficult to build a new machine that provides the same services. It also makes testing more difficult because you cannot guarantee that you have replicated the host in your testing environment. When a bug is found in production that can't be reproduced in the test environment, fixing it becomes much more difficult.

Alternative Analogies

There are other analogies that people use, especially in countries where cattle ranching is less common. One is the analogy of fine porcelain plates and paper plates. You take good care of fine porcelain plates because they are expensive and difficult to replace. In contrast, if a paper plate starts to lose structural integrity, you simply bolster it by putting another paper plate underneath it. If it becomes completely unusable, you replace it.

Another analogy is that modern system administration treats machines like blood cells, not limbs. Blood cells are constantly dying off and being replaced. Limbs, however, are difficult to replace and are protected.

3.2 Scaling

Cattle-like systems give us the ability to grow and shrink our system's scale. In cloud computing a typical architecture pattern has many web server replicas behind a load balancer. Suppose each machine can handle 500 simultaneous users. More replicas are added as more capacity is needed.

Cloud providers such as Amazon Elastic Compute Cloud (Amazon EC2), Google Cloud Platform, and Microsoft Azure have autoscale features where they will spin up and tear down additional replicas as demand requires. This kind of scaling is possible only when machines are cattle. If setting up each new machine required individual attention, the autoscale feature would not be possible.

In such systems we no longer are concerned with the uptime of a particular machine. If one machine fails, the autoscaler will build a new one. If a machine gets sick, we delete it and let the autoscaler do its job. Per-machine uptime was cool in the 1990s but now we measure total system health and availability.

Scale-out architectures are discussed further in [Section 16.6.3](#) and in Volume 2 of this book series.

3.3 Desktops as Cattle

The concept of generic, replaceable machines was first used in desktop environments, long before the cattle and pets analogy was coined. We already discussed the importance of unifying workstation configurations in [Chapter 1](#), “[Climbing Out of the Hole](#),” and we’ll discuss it in greater detail in [Chapter 8](#), “[OS Installation Strategies](#).”

The benefits of generic desktops are manifold. Users benefit from improved customer support, as SAs are no longer struggling to learn and adapt to an infinite number of variations. Repairs happen faster because the IT staff has a single vendor repair procedure to navigate.

Contrast this to an environment where each PC is fully customized. Fixing a software problem is difficult because any change may break something else. It is difficult to know what “working” means when there is no understanding of what is on the machine. Support for older operating systems depends on finding someone on the IT team who remembers that OS.

Creating an environment where cattle are the norm is the primary focus of chapters in [Part II](#), “[Workstation Fleet Management](#),” and [Part III](#), “[Servers](#).” [Chapter 11](#), “[Workstation Standardization](#),” focuses on taking a fleet of workstations that are pets and bringing about unification.

Resetting to a More Uniform State

One of the things that made Apple iPads such a success is that they reset the clock on variation.

PCs had become so customizable that variations had gotten out of control. One of the downsides of competition is that companies compete by differentiating their products, which means making them unique and different. Hardware vendors had many variations and choices, each trying to appeal to different customer segments. Each new market that Microsoft addressed resulted in adding customizability features to attract those users. As a result, by 2005 the complexity of supporting a fleet of Windows machines required a fleet of IT professionals.

Apple iPads took us back to having one particular configuration with curated applications. The uniformity made them more stable and consistent, which then permitted us to focus on the applications, not the infrastructure. Apple retains tight control over the iPad environment so that when the company repeats Microsoft’s mistake, it will play out much more slowly.

3.4 Server Hardware as Cattle

Server hardware and software in a datacenter is another situation where we have pets and cattle. At some companies each machine in the datacenter is specified to meet the exact needs of the applications it will run. It has the right amount of RAM and disk, and possibly even additional external storage peripherals or other hardware. Each machine may run a different operating system or OS release. This ensures that each application is maximally optimized to the best of the system administration team's ability.

However, these local optimizations cause inefficiencies at the macro scale. Each machine requires special maintenance procedures. Each operating system in use, and possibly each version of each operating system, requires individual attention. A security patch that must be tested on ten OS versions is a lot more work than one that has to be tested on only one or two versions. This kind of cost eventually outweighs the optimizations one can do for individual applications.

As a result, in large companies it often takes six months or more to deploy a new server in a datacenter. A consultant working at a U.S. bank said it takes 18 months from the initial request to having a working server in their datacenter. If you aren't sure why banks have such lousy interest rates and service, imagine if a phone app you wanted didn't start to run until a year after you bought it.

Contrast this to an environment that has a cattle strategy for its datacenter. Some companies standardize on two or three hardware variations and one or two OS releases. You might not receive the exact hardware you want, but you receive it quickly. Perfect is the enemy of good: Would you rather be up and running this week with hardware that is good enough, or wait a year and have the exact hardware you dreamed of, which is now obsolete?

Case Study: Google's Two Hardware Types

For many years Google standardized on two types of machines. Diskful machines maximized the amount of hard disk storage that could be packed into a single machine. Index machines (so called because they stored the search index) maximized the amount of RAM that could fit in a 1U configuration. Teams that requested machines in the datacenter could choose between one or the other and receive them within minutes because they were preloaded and ready for use.

This setup made handling future orders easier. The purchase department collected orders from all teams and tallied the number of diskful and

index machines requested. This was considerably easier than if each month department members had to manage requests for thousands of different bespoke configurations.

Software was designed to fit best with one model or the other. Most services were big enough that they required many (often thousands) machines, some of each type. For example, an application would be split to run the application's web frontend on index machines while storing application data on diskful machines. If the hardware being offered was 10 percent slower than the ideal machine, employees would simply request additional machines to compensate for the lagging performance.

This evolved into a pattern that was, actually, the opposite. Engineers didn't think in terms of spec'ing out the perfect machine. Instead, they designed applications such that scaling was done by adding a certain number of machines per unit of workload. They performed tests to see how many machines (of the type currently offered) would be required to process the number of users or the workload expected. Employees then could request that number of machines. They no longer thought of applications in terms of which machine would be best suited to a particular application, but rather how much generic capacity was required. As faster models were introduced into the datacenter, benchmarks would be run to develop new capacity planning models and the process would repeat.

Not every environment can standardize down to one machine type, but we can provide a few standard configurations (small, medium, and large) and guide people to them. We can minimize the number of vendors, so that there is one firmware upgrade process, one repair workflow, and so on.

Offering fixed sizes of virtual machines (VMs) results in less isolated or stranded capacity. For example, we can make the default VM size such that eight fit on a physical machine with no waste. We can offer larger sizes that are multiples of the default size. This means we are never left with a physical machine that has unused capacity that is too small for a new machine. It also makes it easier to plan future capacity and reorganize placement of existing VMs within a cluster.

By offering standardized sizes we enable an environment where we no longer look at machines individually, but rather treat them as scaling units to be used when sizing our deployments. This is a better fit for how distributed computing

applications are designed and how most applications will be built in the future.

We can also standardize at the software level. Each machine is delivered to the user with a standard OS installation and configuration. The defaults embody the best practices we wish all users would follow. Modifications made after that are the application administrator's responsibility. We'll discuss better ways to handle this responsibility in the next chapter.

The Power of Defaults

Defaults are powerful. If you announce an OS configuration change that all IT subteams are required to make, you'll get angry push-back from your loudest and most vocal co-workers. You will get very little participation. In fact, there may be enough push-back that you withdraw the request. Often a tyranny of a few loud complainers prevents the majority from receiving a beneficial change.

In contrast, if you make that change or setting part of the default configuration that is delivered with each new server (thanks to your automated OS install), you may be surprised at how little noise it generates. Most people will live with the change. The people who previously would have made noise will still complain, but now you can work with them to address their concerns. See the anecdote in [Section 7.3.1](#).

3.5 Pets Store State

Another way of describing pets is to note that they contain a lot of irreproducible state. Cattle are stateless, or contain only reproducible state.

State is, essentially, data or information. That information may be data files, configuration, or status. For example, when running MS Excel, the spreadsheet currently loaded is the state. In a video game, the player's score, position, and status are state. In a web-based application, there is the application itself plus the database that is used to store the user's data. That database is state.

The more state a machine holds, the more irreplaceable it is—that is, the more pet-like it is. Cattle are generic because we can rebuild one easily thanks to the fact that cattle contain no state, or only state that can be copied from elsewhere.

A web server that displays static content (web pages and images) is stateless if that static content is a copy from a master stored elsewhere. The web server can be wiped and reloaded, but as long as the content can be copied from the primary

source, the new server is functionally the same as the original.

But suppose a web application has a database. If the machine is wiped and reloaded, the database is lost. We can restore it from backups, but then we will have lost any new data accumulated since the last backup was done. This web application is stateful.

Configuration data is also state, but it can usually be regenerated. Which software packages were installed and how they were configured are state, even though the contents of the software packages themselves are not state; they come from a master repository. The state can be reproduced either manually or via automation.

Irreproducible configuration state can be particularly insidious. In this case the state is not a particular configuration file but rather how the system was made that makes it a snowflake server. We've seen important servers that could be rebuilt only by installing an old version of the software and then installing an upgrade package; installing the final version directly did not work. Unknown and unidentifiable state was being generated during the upgrade process that somehow was not reproduced via the direct installation. This is the kind of unexplained state that makes you want to cry.

Irreproducible Laptops

When Tom arrived at Ciberneta, the company depended on an application that had been installed on a set of laptops many years ago. By then, no one working there could figure out which combination of Windows release, patches, and installation packages would create a new laptop with a working version of the software. Each time one of the original laptops died, the company moved one step closer to insolvency.

The company was in the process of creating a replacement for the software. If the new software was ready before the last laptop died, the company would survive. If not, the company would literally not be able to perform the financial processing it did for customers. It would have to go out of business. One of the laptops was kept in a safe as a precaution. The others were used carefully and only when needed.

When there were only four working laptops remaining, VMware introduced a product that took a snapshot of a physical hard drive and created a virtual machine image (physical to virtual, or p2v). Luckily it worked and soon a virtual laptop could be run on any other machine. This reduced the risk of the replacement project being late, and probably saved

the company.

3.6 Isolating State

We can turn pets into cattle by isolating the state. Optimally this is done during the design process, but sometimes we find ourselves doing it after the fact.

Imagine a typical web application running entirely on a single machine. The machine includes the Apache HTTP server, the application software, a MariaDB database server, and the data that the database is storing. This is the architecture used by many small web-based applications.

The problem with this architecture is that the single machine stores both the software and the state. It is a pet. This situation is depicted in [Figure 3.1a](#).

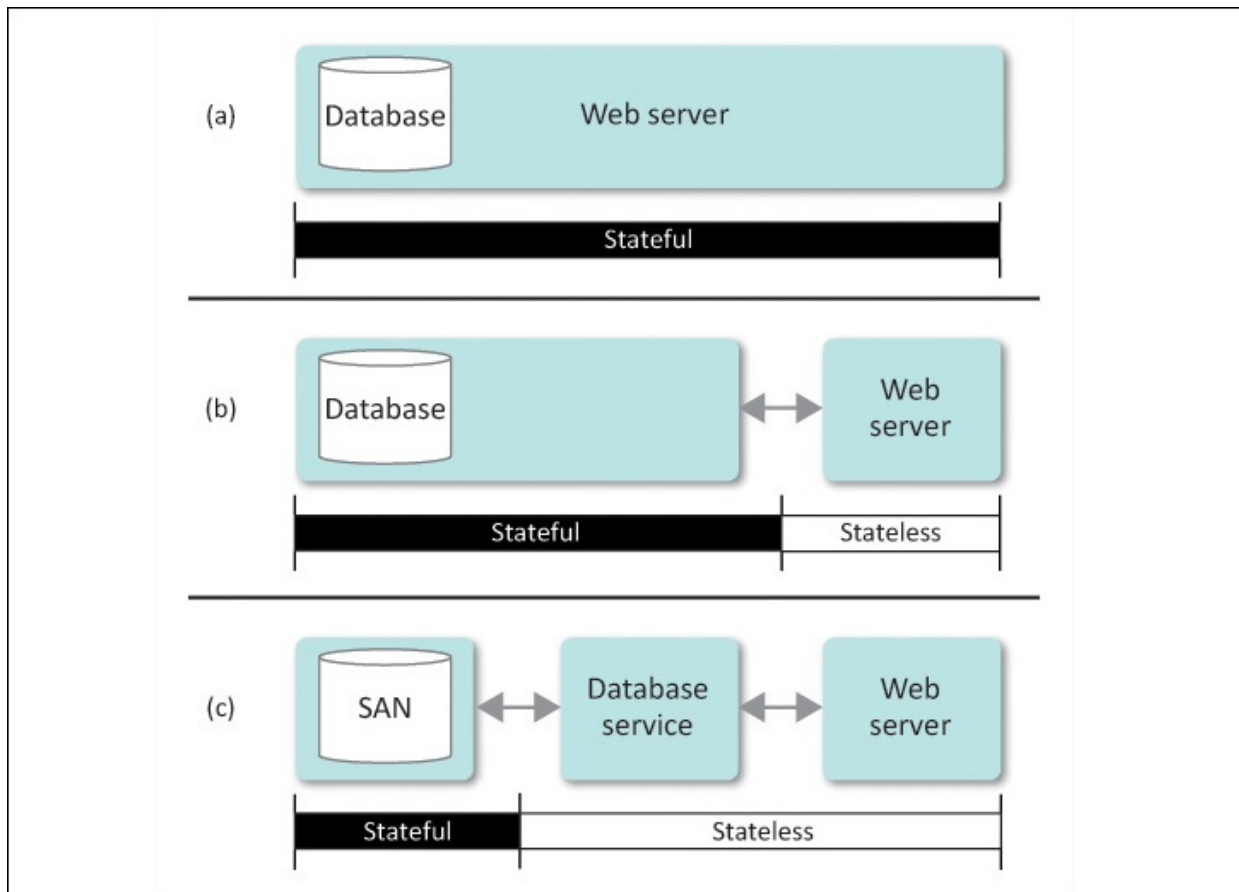


Figure 3.1: Evolving a web service to isolate state

We can improve the situation by separating out the database. As depicted in [Figure 3.1b](#), we can move the MariaDB database software and the data it stores to another machine. The web server is now cattle-like because it can be reproduced easily by simply installing the software and configuring it to point to

the database on the other machine. The database machine is a pet. However, having a cattle + pet situation is an improvement over having one big pet. If the cattle-like server becomes sick, we can easily replace it. The pet, since it has a single function, can be more easily backed up to prepare for an emergency. We can also lock out users so there is less chance of human-caused problems, and we can use more reliable (and more expensive) hardware. By identifying and isolating the state, we are putting all our eggs in one basket, but we can make it a very good basket—one to which we give special care and attention.

The state that remains is the data stored in the database. We can move this data to an external storage to further isolate the state. For example, rather than storing the data on local disk, we can allocate a data volume on our storage area network (SAN) server, as depicted in [Figure 3.1c](#). Now the database machine is stateless.

It can be wiped and reloaded without losing the data. It is simply configured to attach to the right SAN volume to access the state.

Many systems go through this kind of evolution. Sometimes these evolutions happen during the design stage, resulting in a design that isolates state or minimizes the number of places in which state is stored. For example, we might consolidate state into a single database instead of storing some in a SQL database, some in local files, and some in an external application service. At other times this kind of evolution happens after the fact. System administrators spend a lot of time reconfiguring and reengineering older systems to evolve them as needed, often because they were designed by predecessors who have not read this book. Lucky you.

This process is also called decoupling state. The all-in-one design tightly couples the application to the data. The last design decouples the data from the software entirely. This decoupling permits us to scale the service better. For example, the web server can be replicated to add more capacity.

Decoupling state makes it easier to scale systems. Many scaling techniques involve replicating services and dividing the workload among those replicas. When designing a system, it is generally easier to replicate components that are stateless. If we administer these components as cattle, we can easily generate and destroy them as demand increases and decreases. [Figure 3.2](#) is similar to [Figure 3.1c](#), but the web server component has been replicated to scale frontend capacity. A replicated database cache was added to off-load read-only queries, improving database performance. This kind of scaling is discussed further in [Chapter 18](#), “[Service Resiliency and Performance Patterns](#).”

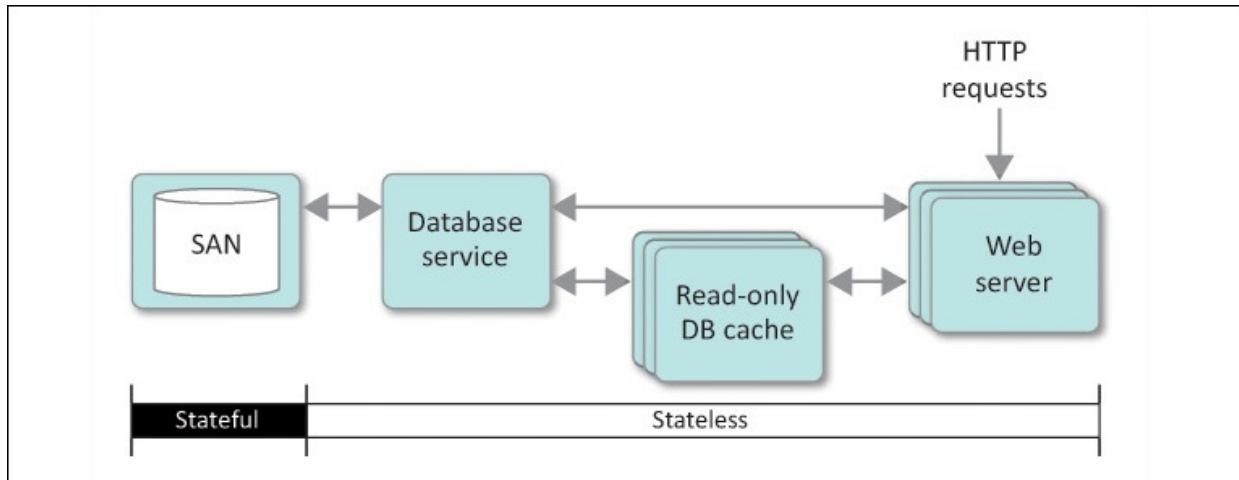


Figure 3.2: A scalable web application service

Blogs and State

State can also be moved to external services. Originally blog platforms were made up of software that generated each page on demand by reading data from locally stored files and an SQL database. This meant state was in three places (the software, the SQL server, and local files). Scaling such systems is very difficult.

In 2016, a new generation of blogging platforms arrived that required no server-side state. In this case, the site was a set of static files that could be uploaded to any web server—even ones without a database or the ability to execute code. Such platforms used client-side JavaScript for all interactive features.

Blog site generators like Hugo and Jekyll typically work as follows. The blog owner creates a Git file repository that stores everything related to the site: images, the text of blog posts, metadata that describes what the web site should look like, and so on. The site generator uses this information to generate the entire site as a set of static files. These files are uploaded to a web server. If a new blog post is created in the Git repository, the entire site is regenerated and uploaded again to the web host.

Highly stateful content such as user comments is handled by external services such as Disqus. While the comments appear to be dynamically updating on the site, they are really loading from the Disqus servers using HTML5 code that does not change. This eliminates most of the infrastructure the blog owner must maintain.

Because the files are static and require no server-side state, they can be served from nearly anywhere. This includes a directory on a file server, a Dropbox account, or a massive multiserver web hosting infrastructure.

3.7 Generic Processes

We can also make processes more generic to improve efficiency. For example, onboarding new employees is a complex process. In some companies each division or team has a different onboarding process. In some places engineers have a different onboarding process than non-engineers. Each of these processes is pet-like. It takes extra effort to reinvent each process again and again. Improvements made for one process may not propagate to the others. However, this situation often arises because different teams or departments do not communicate.

In contrast, some companies have a unified onboarding process. The common aspects such as paperwork and new employee training are done first. The variations required for different departments or roles are saved to the end. You would think this is a no-brainer and every company would do this, but you'd be surprised at how many companies, both large and small, have a pet-like onboarding process, or unified the process only after losing money due to a compliance failure that required the company to clean up its act.

Onboarding

Onboarding is the process by which a new employee is brought into the company. While it is not usually the responsibility of the IT team, much of the process involves IT: creating accounts; delivering the employee's computer, phone, and other technology; and so on. See [Chapter 12](#), "[Onboarding](#)."

Another example is the process for launching and updating applications in production. Large companies often have hundreds of internal and external applications. A retailer like Target has thousands of applications ranging from inventory management to shipping and logistics, forecasting, electronic data interchange (EDI), and the software that handles the surprisingly complex task of generating price labels.

In many organizations each such application has been built using different software technologies, languages, and frameworks. Some are written in Java; others in Go, Python, or PHP. One requires a particular web framework. Another

requires a particular version of an operating system. One requires a certain OS patch; another won't work on a machine *with* that patch. Some are delivered as an installable package; with others the developer emails a ZIP file to the system administrators.

As a result the process of deploying these applications in production is very complex. Each new software release requires the operations team to follow a unique or bespoke process. In some cases the process is full of new and different surprises each time, often based on which developer led that particular release. Joe sends ZIP files; Mary sends RAR files. Each variation requires additional work and additional knowledge, and adds complexity and risk to the production environment. Each variation makes automation more difficult.

In other words, each of these processes is a pet. So how can we turn them into cattle?

Around 2012 a number of organizations identified the need to unify these processes. Many new technologies appeared, one of which was the Docker Container format. It is a format for software distribution that also unifies how production environments deploy applications. This format not only includes all the files required for an application or service, but also includes a standard way to connect and control them. Docker Containers includes meta-information such as which TCP port the service runs on. As a consequence, in a service hosting environment nearly all applications can be deployed the same way. While not every application can work in the Docker Container system, enough can to greatly reduce the number of pets in the environment.

The Docker system includes a number of elements. The Docker Container image is an archive file (like ZIP or TAR) that includes all the files required for a particular service. A Dockerfile is a file that describes how to build an image in an automated fashion, thereby enabling a repeatable process for building images. A Docker compose file defines a complex application made up of many containers, and describes how they talk to each other.

[Listing 3.1](#) is a Dockerfile that describes how to create an image that includes the Apache HTTP server and related files. The `EXPOSE 80` statement indicates that the software this image runs needs exclusive access to TCP port 80.

Listing 3.1: A Dockerfile describing how to build a Docker image

[Click here to view code image](#)

```
RUN apt -get update && apt -get install -y apache2 \
    && apt -get clean && rm -rf varlib/apt/lists /*

ENV APACHE_RUN_USER www -data
ENV APACHE_RUN_GROUP www -data
ENV APACHE_LOG_DIR varlog/apache2

EXPOSE 80

CMD ["usr/sbin/apache2", "-D", "FOREGROUND "]
```

[Listing 3.2](#) shows a Docker compose file for an application that consists of two services: one that provides a web-based application and another that provides API access. Both require access to a MySQL database and a Redis cache.

Listing 3.2: A Docker compose file for a simple application

[Click here to view code image](#)

```
services:
  web:
    git_url: git@github.com:example/node -js -sample.git
    git_branch: test
    command: rackup -p 3000
    build_command: rake db:migrate
    deploy_command: rake db:migrate
    log_folder: usrsrc/app/log
    ports: ["3000:80:443" , "4000"]
    volumes: ["tmp:tmp/mnt_folder "]
    health: default
  api:
    image: quay.io/example/node
    command: node test.js
    ports: ["1337:8080"]
    requires: ["web"]
databases:
  - "mysql"
  - "redis"
```

With a standardized container format, all applications can be delivered to production in a form so sufficiently self-contained that IT doesn't need to have a different procedure for each application. While each one is wildly different internally, the process that IT follows to deploy, start, and stop the application is the same.

Containers can be used to build a beta environment. Ideally, the test environment will be as similar to the production environment as possible. Anytime a bug is found in production, it must be reproduced in the test environment to be investigated and fixed. Sometimes a bug can't be reproduced this way, and fixing it becomes much more difficult.

The reality is that at most companies the beta and production environments are very different: Each is built by a different group of people (developers and SAs) for their own purposes. A story we hear time and time again is that the developers who started the project wrote code that deploys to the beta environment. The SAs were not involved in the project at the time. When it came time to deploy the application into production, the SAs did it manually because the deployment code for the beta environment was unusable anywhere else. Later, if the SAs automated their process, they did it in a different language and made it specific to the production environment. Now two code bases are maintained, and changes to the process must be implemented in code twice. Or, more likely, the changes are silently made to the beta deploy code, and no one realizes it until the next production deployment breaks. This sounds like a silly company that is the exception, but it is how a surprisingly large number of teams operate.

Not only do containers unify the production environment and make it more cattle-like, but they also improve developer productivity. Developers can build a sandbox environment on their personal workstations by selecting the right combination of containers. They can create a mini-version of the production environment that they can use to develop against. Having all this on their laptops is better than sharing or waiting their turn to use a centrally administered test stage environment.

In June 2015 the Open Container Initiative (OCI) was formed to create a single industry-wide standard for container formats and runtimes. Docker, Inc., donated its container format and runtime to serve as the basis of this effort.

Containers are just one of many methods for unifying this process.

Shipping Containers

The concept of Docker Containers comes from the shipping industry. Before shipping containers were introduced, individual items were loaded and unloaded from ships, usually by hand. Each item had different dimensions and therefore had to be handled differently. An individual lamp needed to be carefully handled, while a large sack of wheat could be

tossed about.

That changed in April 1956, when Malcom McLeans organized the first shipment using standardized containers.

Standardized shipping containers revolutionized how products move around the world. Because each shipping container was the same shape and size, loading and unloading could be done much faster. Cranes and automation had to be built to handle only one shape, with a standardized maximum weight and lifting points.

A single container held many individual items, all with the same destination. Customs officials could approve all the items in a particular container and seal it, eliminating the need for customs checks at transit points as long as the seal remained unbroken.

Intermodal shipping was born. A single container would be loaded at a factory and remain as a unit whether it was on a truck, train, or ship. Standard shipping containers are accepted everywhere.

3.8 Moving Variations to the End

Operational science teaches us to move variations in a process to the end. Burger King restaurants make a generic hamburger, waiting until the last minute to add toppings such as ketchup, mustard, and pickles. Unsold inventory can be kept generic so that it can be quickly customized when the order is placed. Otherwise, a restaurant might end up with a surplus of burgers with pickles sitting unsold while Christina waits for her pickle-less order to be made from scratch.

Auto manufacturers also delay variation to the last possible moment. Option packages are added at the end, where demand is better understood. Unusual items like special audio systems or fancy tires are added by the dealer only after a particular customer requests them.

As long as the WIP stays generic, the process is simple and easier to streamline. You can mass-produce dozens of generic burgers with a single process and a single focus, improving it constantly to be more efficient. Once they are customized, everything becomes a special snowflake process. Our ability to improve the process is not impossible, though it is deterred.

This strategy also works in IT. Design systems and processes to keep WIP generic for as long as possible. Save variations until the end. This reduces the combinations of configurations and variables to be tested, makes it easier to verify completeness and accuracy, and makes it easier to improve the process.

We've already seen this in our discussion of the onboarding process, where common tasks were done first.

Another example relates to laptop distribution. Imagine a company where all new employees receive the same laptop, with the same OS, the same configuration, and the same applications. However, when a user logs in for the first time, specific applications are installed depending on whether the employee is an engineer, salesperson, or executive. After that customers can customize the workstation to their liking. This enables the entire laptop deployment process to be generic until the last possible moment.

Now imagine instead that such customizations were done at the start. If there was a burst of new engineers starting at the company, the IT department might find itself with no engineering laptops left but plenty of sales laptops. If the hardware was the same they could at least rework the laptops to be engineering laptops. This would double the effort expended on each laptop, but it would solve the immediate problem. If the hardware models were different, however, the engineers would have to wait for laptops since the units are not fungible resources. Alternatively, the engineers could be retrained to work as salespeople, but that would be silly since people are not fungible resources.

When things are different in software, we can treat them generically by choosing the right level of abstraction. Containers permit all services to be treated generically because no matter what is on the inside, the SAs can simply deal with them at generic touch points that are common for all.

Some software frameworks permit plug-ins or drivers to be written so that the framework deals with generic “things” but the differences are mediated by the plug-in.

3.9 Automation

Consistency makes it easier to automate a process. It is easier to write automation for cattle than for pets because there are fewer surprises and variations to be aware of and fewer permutations to test. Automation brings about opportunities for self-service system administration. Web sites and other tools can empower users to get their needs met without human intervention.

You can also look at this another way: Before we can improve things, we must make things consistent. Making improvements to something inconsistent is like wrestling a pig: It's messy and you probably won't win. Once things are consistent, we can make them better—optimize them—and we gain the freedom to experiment and try new things. Our experiments may fail, but if we do not try,

there is no way to improve. At least with each failure we learn something. This is not a rationalization that makes us feel better about our failures: The experiments that are a success are valuable because the system has been improved (optimized); the experiments we revert are learning experiences that guide us as we make future improvements.

You'll see this pattern of chaos⇒defined⇒repeatable⇒optimizing throughout this book. It is also the basis of "[The Three Ways of Operational Improvement](#)" described in [Section 12.3.5](#), and is the basis of the assessment levels in [Section 55.3.2](#).

3.10 Summary

Pets are machines that are irreproducible because they are highly customized over a long period of time with no record of how to exactly replicate the process. They must be managed individually. If a pet becomes damaged or corrupted, it must be carefully brought back into the desired state just as a doctor tends to a sick patient.

Cattle are machines that can be reproduced programmatically and are therefore disposable. If one of these cattle gets damaged or corrupted, it is wiped and rebuilt. To complete the analogy, when a single animal in a cattle drive is sick, it is killed so that the herd can keep moving.

Cattle-like systems make it easier to manage large numbers of machines. It is easier to mass-produce IT when machines are generic.

Desktops can be made cattle-like by starting them all the same via automation, and using directory services and other techniques to maintain their sameness. We can also reduce the number of vendors and models to make the repair processes more generic.

Servers have different challenges. The software each runs is usually very different. We can use containers and configuration management systems to automate the setup of these differences so that they can be reproduced by running the code again. More importantly, pet-like servers store irreproducible state: information that is not stored elsewhere (other than backups). We can design our services to separate out our state to specific machines so as to increase the number of cattle-like systems. State can be stored on a separate file server, database server, or external service.

We can also improve efficiency by making processes more cattle-like. A process should save any variations until the last possible moment. By keeping things generic at the start, we can mass-produce the start of the process.

Exercises

1. Explain the pets and cattle analogy for computers.
2. What is a snowflake server? Why are they a bad idea?
3. If a snowflake server is risky, how can we reduce risk through repetition?
4. How do cattle-like systems help us be more efficient?
5. How do cattle-like systems help us scale services?
6. According to this chapter, why do banks have lousy interest rates?
7. A laptop and a desktop PC are very different. In what way could we treat them both as cattle of the same herd?
8. What is state? What is irreproducible state?
9. Why is isolating state to particular machines a good thing?
10. How can beta and production environments end up being different? How can we make them as similar as possible?
11. How is mass-production aided by moving variations to the end?
12. Sometimes bad customer service is described as being treated like cattle. Yet, some of the best companies have practices that assure that everyone receives extremely high-quality service in an efficient and mass-produced way. These companies are also managing people like cattle. How are the latter companies able to achieve this without offending their customers?
13. Pick a service in your organization that stores a lot of state. Describe how it could be implemented using an architecture that isolates state.
14. What are the benefits of moving variations to the end of the process?
15. Pick a process in your organization that has a lot of variation. How can it be restructured to move the variation to the end? What benefits would be achieved by doing this?

Chapter 4. Infrastructure as Code

This chapter is about infrastructure as code (IaC), a system administration strategy where infrastructure is provisioned and managed through machine-processable definition files rather than manual labor. With this approach, we can manage our infrastructure using software engineering techniques such as source code control, unit tests, and continuous integration and delivery (CI/CD). IaC is a relatively new term, but it brings together a lot of ideas that have been present in system administration for decades.

In an IaC environment, we don't make changes to systems directly. Instead, we update the code and data that are used to create our environments. For example, instead of configuring many web servers individually, imagine if we kept a text-based configuration file that described which web sites were hosted on each machine. We could write a program that read that file and updated the hosts as appropriate. When this program was run, the hosts would be configured to match what was described in the configuration file. An existing web server might require no changes. A newly added definition would result in the automation spinning up a new web server, and possibly creating a new virtual machine for it to run on. Moving a web site from one server to another would be done by changing the configuration file and rerunning the program. It would remove the web site from one server and add it to the other. The same configuration data would also drive updates of web load balancers, security systems, database access, and so on.

When we administer systems this way, the code we use to control our infrastructure is not just part of our infrastructure, it *is* our infrastructure. It describes the network connections between machines, the machines themselves, and the applications that run on the machines. To create a new machine, we update our machine-processable description and let the automation create it. If we do not like that change, we can revert to a previous version of that machine-processable description. We fix our infrastructure as a developer fixes a bug: We make a change to code and test it in a test environment; if that is successful, we push the change to production.

The previous chapter was about improving efficiency by minimizing variation. IaC permits us to better keep systems unified and manage the remaining variations. We might treat our web servers as cattle, using code to build each one exactly the same way. However, the quantity of such cattle, how they connect to the network and to each other, and which web sites they serve

are variations that can be controlled by IaC. We can wrangle those variations best when we do it by code.

IaC is not a particular programming language or system. It is a strategy. We can implement this strategy by using either home-grown automation or (preferably) off-the-shelf systems such as CFEngine, Puppet, Chef, and other popular packages.

IaC can be used to control every layer of our infrastructure. This is easiest when we are starting a new company or a new project. However an “all in” strategy is not required if you have a preexisting or legacy environment. You can start by controlling one particular aspect and adding from there.

4.1 Programmable Infrastructure

In the old days infrastructure was not programmable. Setting up a machine required visiting the datacenter, manually unboxing a machine, lifting it into a rack, connecting cables, setting up disks, plugging the machine into the right local area network (LAN), and so on. Over time our infrastructure became more programmable. In the mid-1990s, virtual LANs (VLANs) made it possible to move a computer from one network to another through software rather than changing physical cables. In the late 1990s, storage area networks (SANs) permitted slicing and dicing a large pool of disk storage and making the chunks available to machines as virtual disks. Virtual machines became popular sometime after 2005, permitting the resources of one large machine to be used to create many smaller virtual machines.

All of this virtualization lead to programmability. Originally making changes involved clicking or typing into a user interface. Eventually each of these technologies added application program interfaces (APIs) that let us automate these processes using languages such as Java, Perl, Python, and any other that could speak the API protocol. Now it is possible to use API calls to create a virtual machine, allocate and attach a virtual disk, and connect it all to a particular virtual network. Manual labor is no longer required, nor are visits to the datacenter.

Cloud-based services took this one step further, providing access to virtual infrastructure controlled by APIs, with no requirement for local infrastructure. Cloud-hosted services such as Amazon AWS, Google Compute Platform, and Microsoft Azure went from being a niche in 2006 to an accepted business practice by 2016.

When installing a machine became an API call, we all became programmers.

4.2 Tracking Changes

SAs should be jealous of software engineers. They store all their source code in a version control system (VCS) such as Git, Mercurial, or SubVersion. As a result, they can view the history of a file and see when each line of a file was last edited, by whom, and why the change was made.

[Listing 4.1](#) is an example output of the `git annotate` command. It shows each line of a file, annotated with the change ID that most recently added or modified the line, who committed the change and when, plus the line number and text of the file.

Listing 4.1: Git annotate output

[Click here to view code image](#)

ChangeId:	(Author:	Date:	Line:)	Code:
eae564f2	(craigp	2015-08-06	172)	isRelayed :=
	r.Header.Get(relayHeader)			
67b67bd0	(mjibson	2015-04-20	173)	reader := &passthru{ReadCloser:
	r.Body}			
67b67bd0	(mjibson	2015-04-20	174)	r.Body = reader
67b67bd0	(mjibson	2015-04-20	175)	w := &relayWriter{
67b67bd0	(mjibson	2015-04-20	176)	ResponseWriter:
	responseWriter}			
ba83ae2e	(craigp	2015-09-16	177)	rp.TSDBProxy.ServeHTTP(w, r)
ade02709	(ipeters	2016-02-23	178)	if w.code/100 != 2 {
ade02709	(ipeters	2016-02-23	179)	verbose("got status
	%d", w.code)			
67b67bd0	(mjibson	2015-04-20	180)	return
67b67bd0	(mjibson	2015-04-20	181)	}
67b67bd0	(mjibson	2015-04-20	182)	verbose("relayed to tsdb")
a6d3769b	(craigp	2015-11-13	183)	collect.Add("puts.relayed",
a6d3769b	(craigp	2015-11-13	184)	opentsdb.TagSet{}, 1)

Each change is committed along with a comment. [Listing 4.2](#) is the comment submitted by `ipeters` about change `ade02709` that explains why he added lines 178 and 179.

Listing 4.2: Git commit comment

[Click here to view code image](#)

```
commit ade02709
Author: ipeters <ipeters@example.com >
Date: Tue Feb 23 16:12:39 2016 -0500
```

cmd/tsdbrelay: accept responses with 2xx code

opentsdb may return 204 or 200 on successful PUTs to *apiput*. 204 indicates success with no response body, but when the *details* parameter is specified, a 200 is returned along with a simple response document. *tsdbrelay* should relay all successful PUTs to *bosun*.

I am explicitly not changing *relayMetadata*, since *bosun* itself appears to only return a 204.

VCS systems permit us to see the entire history of a file. We can find out exactly what the file looked like yesterday, a month ago, or on December 2 of last year.

Contrast this to how we track the history of changes on a server. Yes, files have timestamps, so we know when a file was last modified. But we don't know what in the file was changed, and we don't know who made the change or why. Outside of reviewing backups, we have no history of changes.

For example, one day Tom's team was working with a set of ten web servers, all configured the same way—or so they thought. Someone noticed that one machine had a particular network buffer size parameter set differently than the others. The team had no idea why. Was this change an accident? Had someone made a typo on this one machine? Maybe someone had been experimenting with different buffer sizes to achieve better performance but forgot to clean up when done. Was this change intentional? What was the expected benefit? Was it working around a problem that no longer exists?

If the machine had died and needed to be rebuilt, would anyone have known to make this change? If this change was required for a particular application, we would have difficulty getting that application to work again.

Source code in a VCS doesn't have the same problem. Code doesn't appear in a VCS out of nowhere. Code can be commented. When comments are missing, we can use the VCS to see the note attached to a particular change, or see who made the change and speak with that person directly.

Another thing to consider is that software has a unique property that manually done work does not have: It is reusable. If you write a function that sorts a list of integers, you don't have to write a different function that sorts 10 integers and another that sorts 11 integers and another that sorts 12 integers. The same function can sort a list of any size. You can even design a sort function so that it does the right thing with integers, strings, or ZIP codes.

Now consider how computer systems and networks are traditionally managed. We find ourselves with a beta environment that is built one way and a production environment that is built another way. Meanwhile, our developers have created a third system that builds a development sandbox on their own machines.

It would be better to have a single function that accepts parameters to indicate specific variations. For example, the function that sets up an environment might include parameters that indicate if this is a production, beta, or sandbox environment. The fundamental logic is the same for all environments. What changes is the quantity of machines, as well as their names and locations. The production environment might have hundreds of replicas, configured to talk to the live production database. The beta environment might have fewer replicas, configured to talk to a separate database full of synthetic test data. The development environment might have only a single replica of each service type, talking to a tiny database, and all these services are running on a single machine, perhaps the developer's own laptop.

Parameterized code minimizes the procedural differences between the different environments. A change in the procedure for configuring a service does not have to be implemented three times. This reduces the possibility that the beta environment procedure becomes out of sync with the production procedure. It also reduces the likelihood of bugs that appear in production but cannot be reproduced elsewhere.

Imagine if our infrastructure was entirely managed as code. Every change would be made by updating machine-processable definition files that, when processed, created machines, deleted machines, configured systems, created files, started processes, and so on. We would gain the ability to track changes, know the history of our infrastructure, and identify who made which change. Our infrastructure would be parameterized so that we could build the same environment with different quantities, names, and locations. System administration could reap the benefits that software engineers have enjoyed for decades.

That is, in a nutshell, the basics of IaC.

4.3 Benefits of Infrastructure as Code

The three areas that benefit from IaC are cost, speed, and risk.

Cost is reduced because manual labor is reduced or eliminated. Automation is a workforce multiplier: It permits one person to do the work of many. Manual labor scales linearly: The more of a task that must be done, the more people who

must be hired. Without such automation, huge companies such as Facebook and Google would eventually have to hire every SA in the world.

Speed is enhanced. Not only can tasks be done faster, but they can also be done in parallel. They can be done without waiting for a person to be available to do the work. Waiting days for an SA to do a 5-minute task is a common impediment to a team's velocity.

Risk is reduced many ways. Security risk is reduced because we can prove compliance. That is, we can be assured that various knobs and controls are in the secure position. We can reduce the risk of errors and bugs by applying software engineering techniques. For example, the code can be first tested in a beta or testing environment. It is better to discover a bug in the testing environment and fix it there than to suffer the consequences of the bug making it into production.

IaC makes your environment reviewable by others. You can document the assumptions that go into it and the reasons you made various design decisions. This makes it easier for new team members to get involved with your project.

It becomes easier to audit your environment. Auditors can audit your code once, then audit any changes that follow. Without IaC, auditors have to review every inch of every machine every time.

Operations science has a principle that defects should not be passed forward. In an auto factory, this means that if a chassis is defective, you want to discover this problem as soon as possible. If you discover it right away, you can discard or recycle the chassis. Otherwise, you could end up with a \$45,000 vehicle that can't be sold. If the defective chassis is installed, all labor expended on the remaining manufacturing processes is a waste of company resources that could have been applied to vehicles that can be sold. (Of course, repairing the defective vehicle can recover some of the capital lost from making the defective car, but that requires even more labor.)

Likewise, in software, or IaC, we want to detect a bug as early as possible so that it can be fixed early (best case) or at least not let it reach production (worst case).

To determine if something is "bad," you need to have a definition of "good." The automotive world has a definition of what a good chassis is. Manufacturers know the expected weight, height, length, and shape. They can perform a visual inspection for dents.

Defining "good" for IaC is more difficult. We can perform a visual inspection of the code but the only way to know for sure that the code is "good" is to run it. Since we don't want to run the code in production until it is tested, we must run

the code in a test environment. There we can use a variety of tests:

- **Smoke tests:** These verify the most basic functionality. The name comes from the world of electronics, where a smoke test means plugging the device in. If it produces smoke, something is wrong. A smoke test might be to simply verify that a configuration file is syntactically correct by running the software to see if it crashes immediately.
- **Unit testing:** This is where you run parts of code—perhaps a particular function or module—with synthetic data for inputs and verify that the output is as expected. Each part is tested in isolation. For example, a function that formulates a new machine’s hostname based on its location and purpose can be fed combinations of locations and other data to verify the expected results.
- **Integration testing:** This is where you run the entire system to verify that all the parts work together. For example, you might create a DHCP server plus a few machines representing typical configurations. You would then verify that they all were able to receive their network configurations from the DHCP server.
- **Acceptance testing:** This verifies that a system meets the customer’s needs, often by running real-world data through the system and verifying the desired outcomes are realized.
- **Load testing:** This verifies the performance of a system. You might have a particular benchmark that is run to verify that the performance is within expected parameters. Discovering that a new release is significantly slower than previous releases is a simple way to prevent performance problems from reaching production.

We can apply CI/CD methods previously discussed in [Section 1.2.2](#) to system administration. Any change to the (infrastructure) code is checked into a VCS. This triggers automation that runs tests before making the desired changes.

[Figure 4.1](#) depicts this service delivery platform and shows how IaC lets us fully mirror the software delivery life cycle in our infrastructure. The top half shows traditional application software moving through a CI/CD system: being built, packaged, tested, approved, and installed in production. The bottom half shows IaC going through the same processes.

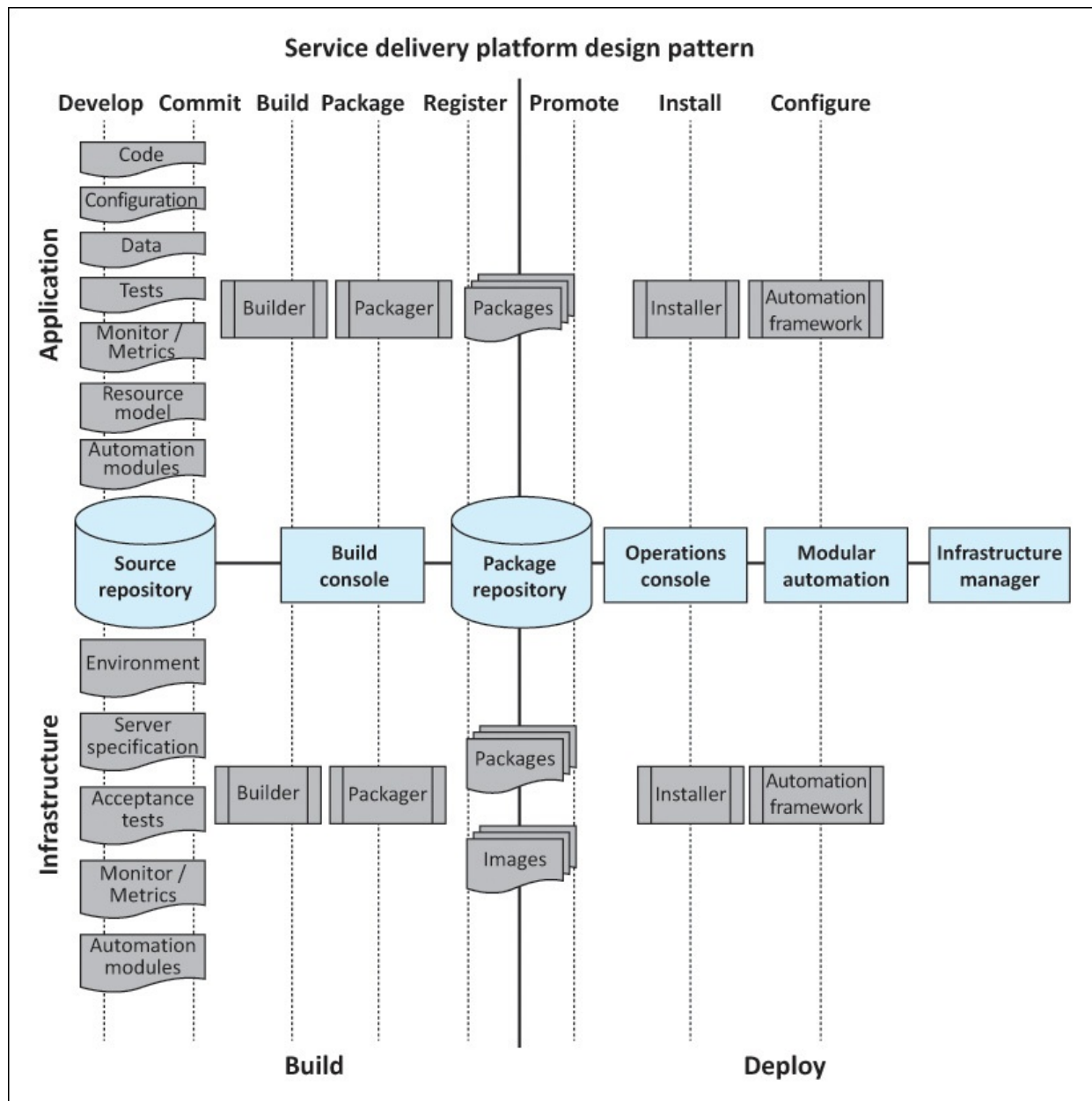


Figure 4.1: The parts of the modern service delivery platform design pattern.
(Reprinted with permission from Damon Edwards of DTO Solutions.)

Fail-Safe Tests

Another kind of test that IaC can do is a fail-safe test. Like a medical machine that automatically turns itself off if it is about to administer harmful amounts of radiation, a fail-safe test checks for situations that should not happen, or situations that could happen only due to a code bug.

Tom built an IaC system for managing complex DNS infrastructure at Stack Overflow. It included a fail-safe test that prevented certain IP

addresses from being exposed to the outside world. This could happen due to a typo, a bug, or a misused template. Even though none of those things should happen, the system fail-safe test worked and refused to do updates if they did happen while the system was running.

4.4 Principles of Infrastructure as Code

The principles of IaC are as follows:

- **Make all changes via definition files.** Configuration is defined in machine-readable data files and executable descriptions such as Puppet manifests, Chef recipes, or Dockerfiles. Do not log into a server to make changes unless you are experimenting in a lab in preparation to create such files.
- **Document systems and processes in code.** Rather than diagrams and human-written documentation, the IaC files should be the source of truth. Code is precise and consistently executed. Diagrams and other human-readable documentation are useful but should be generated from code.
- **Use a VCS for all files.** IaC code should be kept in a VCS. This includes source code, configuration files, and data files—in fact, anything changed manually. This way we know the history of changes, who made them, and why. This record can be audited and used when debugging problems, and for rapid rollback.
- **Apply CI/CD.** Run the tests and other processes with each change. This way we identify bugs as they are created. When one change breaks another part of the system, we find out immediately. It is easier to fix something while the code is fresh in our minds rather than days or weeks later when it hits production.
- **Apply the small batches principle.** A dozen small changes over a week is better than a big change at the end, even if both result in the same final code. Bugs are easier to find when they are isolated to a small change. Bugs are easier to fix when they are new, before more changes are layered on top. Small changes are easier to revert or undo than big changes.
- **Keep services available continuously.** Lean toward techniques for updating services in ways that do not require downtime. This way we can always (or more frequently) be making changes. Otherwise, it is difficult to apply the small batches principle.

4.5 Configuration Management Tools

Many different systems allow you to implement IaC. In the last chapter we discussed Docker and OCI containers. They provide ways to define how immutable elements (images) are constructed and brought together to create services. Systems such as Kubernetes, Mesos, and Docker use these elements to run services.

Another method is to use a configuration management (CM) system to either maintain existing systems in a more IaC-like manner or use them to build infrastructure from the ground up. At the time of writing the most popular CMs are Puppet, Chef, CFEngine, and Ansible; each is available in open source and there are also commercial versions.

Many sites create their own CM system. We recommend against this. It seems easy to write your own CM system at first, but eventually it grows and becomes a big headache to maintain. Using an off-the-shelf CM system permits you to leverage the features and bug fixes that come from the community of users: They have a support network (community or for-pay), and you can enhance your skills through books, web sites, and conferences. It is easier to hire someone with experience in an existing CM system than to train someone to understand one developed in-house.

Leveraging Community Support

CentOS 7 changed the way services were configured. The Puppet modules were updated to reflect this change by community members. When Stack Overflow upgraded to CentOS 7, machines that were configured using Puppet generally needed just to be reinstalled; Puppet rebuilt them with all the services configured the new way. Very little additional work was required other than testing. Machines that had been manually configured required a lot more effort. Being able to leverage the CentOS 7 support others had written was a win that would not have happened with a home-grown system.

4.5.1 Declarative Versus Imperative

CM systems are best known by their domain-specific programming language (DSL)—in other words, a language built for one specific purpose. The Puppet system has a language called Puppet, Chef extends Ruby to be its own language, and so on.

Most programming languages are either declarative or imperative. In an imperative language the code looks like a list of steps describing how to do something. In a declarative language you list what you want the final result to be, and the system figures out how to get there.

Most languages you are familiar with are imperative: Perl, Python, PowerShell, Bash, Ruby, Go, and Fortran. If you want to install a list of software packages on a machine, you write the instructions describing how that is done. [Listing 4.3](#) is an example written in pseudocode.

Listing 4.3: Pseudocode for imperative package installation

[Click here to view code image](#)

```
for package_name in [" apache", "openssh", "httpunit", "bosun "] do
  if packageIsInstalled(package_name) then
    print "Package " package " is installed. Skipping ."
  else
    if operating_system == "centos" then
      result = yum_install(packagename)
    else if operating_system == "debian" then
      result = apt_install(packagename)
    else
      print "ERROR: Unknown operating system ."
    endif
    if error then
      print "ERROR: Package " package " did not install !"
    else
      print "Package " package " installed successfully ."
    endfi
  endif
endfor
```

As you can see, the code handles different operating systems, different error conditions, and so on. It is rather complex.

The same thing in a hypothetical declarative language might look like the code in [Listing 4.4](#).

Listing 4.4: Pseudocode for declarative package installation

[Click here to view code image](#)

```
required_package (" apache ")
required_package (" openssh ")
```

```
required_package ("wine")  
required_package (" christmas ")
```

When [Listing 4.4](#) is run, the CM system will determine which packages are installed on the machine and install any that are missing. It knows the procedure for installing packages on different operating systems. Some CM systems know how to install packages correctly for more than twenty-five different operating systems and variations. If a new operating system is to be supported, that support is added to the CM system. Ideally you do not need to rewrite or update your code. That said, typically there may be some fine-tuning required for a new OS. For example, you may have to specify that a package has one name in a particular OS and a different name in others.

4.5.2 Idempotency

Another attribute of configuration management systems is that they are idempotent. This means that a CM system will not alter the system if the change is not required. Running the same code a second time should not make changes.

In the example shown in [Listing 4.4](#), the DSL would not install a package if it is already installed. This is idempotent behavior. Updating a configuration setting or Windows Registry key is idempotent; if it already stores that value, the system isn't changed. Appending a line to a file is *not* idempotent. Each time the operation runs, the file grows longer.

This sounds like the obvious way for automation to work but it is actually quite an innovation. It requires extra code to check whether something needs to be done before doing it.

Suppose you've written a program that generates the configuration file for a Network Time Protocol (NTP) server. Based on the subnet of the server and other information, it generates the configuration file. A non-idempotent system would simply generate that file and write it out every time the system is run. The result will be a file with the same contents, but the date stamp of the file will change every time. An idempotent system would generate the configuration in a temporary space, compare it to the configuration file found on the system, and rewrite the file only if the two do not match.

Eliminating the gratuitous file update is safer. Fewer disk writes means less wear and tear on the disk. Another benefit is that we actually know when changes happen. For example, if the CM system runs every hour, we will know when a substantive change happened instead of just knowing that the CM system ran and gratuitously rewrote the file.

Because we now know when an actual change happened, we can trigger other actions. Suppose our NTP server must be told to reread its configuration file anytime it changes. The service stops while the configuration file is reloaded. If we sent the reload signal to NTP every time CM ran, this would mean service would be constantly pausing. An idempotent system knows when an actual change happened and can notify the NTP server only when needed.

Because of this idempotency, we can run a CM program periodically. If there are changes to be made, the system makes the changes. If no changes are needed, the CM program makes no changes. If a person logged into the machine and undid a change, the CM system would change it back during the next run.

4.5.3 Guards and Statements

You can think of every change a CM system makes as being defined by two parts. The **guard** is code that determines “Does the change already exist?” The **statement** is code that attempts to make the change. As depicted in [Figure 4.2](#), first the guard is used to determine if there is work to be done. If there is, the statement is used to perform that work. The guard is then run again to verify that the change was successful. Notice that the guard is called twice rather than requiring a third snippet of code that determines if the change was done correctly.

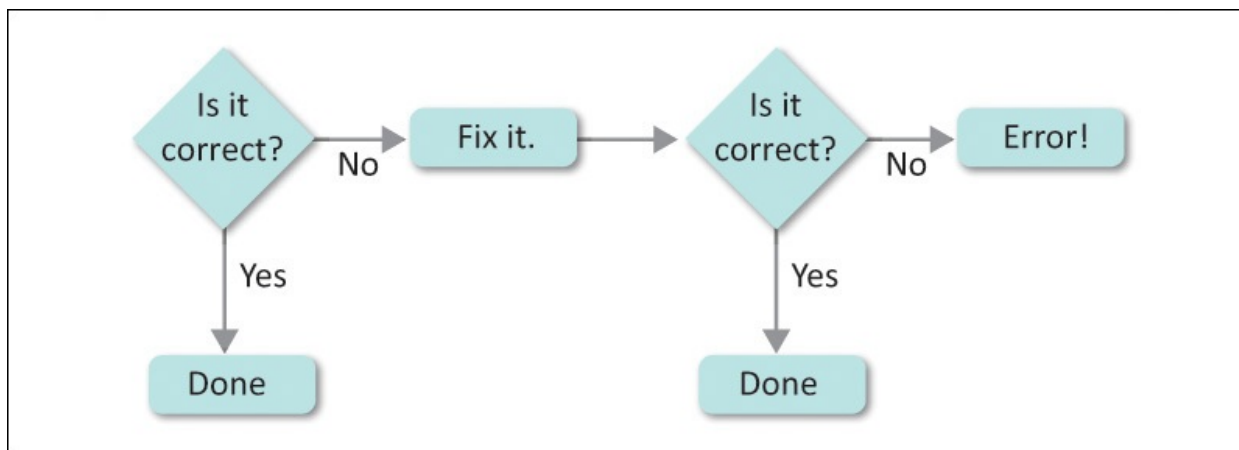


Figure 4.2: Idempotent changes

Dividing each change into these two separate functions is an internal detail of the CM system. You do not need to write this code unless you are extending the system.

Most CM systems have a “dry run” mode that lists what would be changed but does not actually make any changes. This permits us to verify that it is about to make the changes we expect and nothing more. With a CM system built with

guards, dry-run mode is easy to implement: Run the guards but none of the statements.

Origins

The guard/statement concept was first formally presented in a system administration context at USENIX LISA 2004 by Trey Harris in his talk, “A New Approach to Scripting.” He called it Procedural Test-Oriented Scripting (PTOS).

Harris attributed the term *guard* to computer scientist Edsger W. Dijkstra’s monograph entitled “Guarded Commands, Nondeterminacy and Formal Derivation of Programs,” published in *Communications of the ACM* in 1975.

4.6 Example Infrastructure as Code Systems

The best way to understand what IaC looks like is by looking at examples. These examples include small snippets of code in the Puppet language for managing various aspects of a machine or machines. They would be part of a larger CM or IaC system.

4.6.1 Configuring a DNS Client

[Listing 4.5](#) is an example of Puppet code that configures the NTP server on a host. On Windows systems this module uses the server names (line 2) to configure the built-in NTP client. On Unix-like systems it installs NTP client software, knowing that the package name is different for certain operating systems. It then generates the configuration file (`ntp.conf`) using a template that inserts the NTP servers at appropriate locations.

Listing 4.5: Puppet code to configure an NTP client

[Click here to view code image](#)

```
1 class {'ntpclient ':  
2   servers => ['ntp1.example.com ', 'ntp2.example.com '],  
3 }
```

4.6.2 A Simple Web Server

In this example we install the Apache HTTP server and configure it to serve a

site called `first.example.com` and an SSL-enabled site called `second.example.com`.

The HTML files for each site are in `homewebdata/SITENAME/www`. [Listing 4.6](#) shows this code.

Listing 4.6: Puppet code to configure Apache

[Click here to view code image](#)

```
1 class { 'apache ': }
2
3 apache::vhost { 'first.example.com ':
4     port      => '80',
5     docroot => 'homewebdata/first.example.com/www/',
6 }
7
8 apache::vhost { 'second.example.com ':
9     port      => '443',
10    docroot => 'homewebdata/second.example.com/www/',
11    ssl      => true ,
12    ssl_cert => 'etcssl/second.example.com.cert ',
13    ssl_key  => 'etcssl/second.example.com.key ',
14 }
```

Line 1 activates the Apache module that installs the software and creates a default configuration. Lines 3–6 define the first web site. Lines 8–14 define the second web site. Line 11 enables SSL, and lines 12 and 13 define where to find the related cryptographic keys.

As in the previous example, this module does the right thing depending on which OS is being used. Each OS may have a slightly different name for the Apache HTTP software package and may put the configuration files in a different place.

4.6.3 A Complex Web Application

In this next example we define a much more complex application. In the previous examples all changes were happening on the same machine. In this example we will demonstrate the concept of **application orchestration**, which is the ability to coordinate actions among and between different machines. This feature is part of the Enterprise edition of Puppet.

Suppose we want to create a blog service that requires three subservices: a web server for static files, an application (app) server that generates dynamic

content, and a database server that contains blog posts, comments, and other information. They are connected to one another as depicted in [Figure 4.3](#).

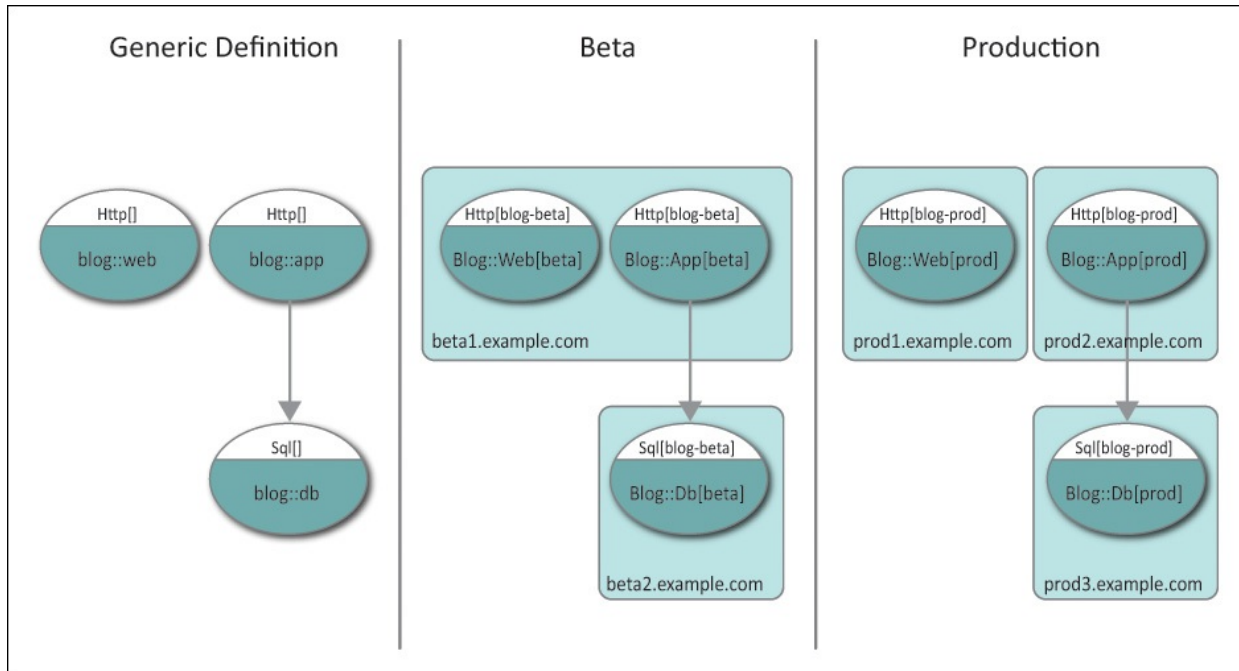


Figure 4.3: An application with database, web, and app servers

We also want to have beta and production versions of these blogs, but we don't want to have to define everything twice. Also, the beta version merges two of the subservices onto one machine to conserve resources. The production environment puts each subservice on its own machine for maximum performance. [Listing 4.7](#) shows how the Puppet `site` keyword is used to define the two environments.

Listing 4.7: Puppet code to define beta and production web servers

[Click here to view code image](#)

```

1  site {
2
3    blog { 'beta ':
4      db_username => 'betauser ',
5      db_password => 'thepassword ',
6      nodes => {
7        Node['beta1.example.com ' ] =>
8          [ Blog::Web['beta '], Blog::App['beta
9        ],
10       Node['beta2.example.com ' ] => Blog::Db['beta '],
11     }
12   }
13 }

```

```

12
13   blog { 'prod ':
14       db_username => 'produser ',
15       db_password => 'anotherpassword ',
16       nodes => {
17         Node['prod1.example.com '] => Blog::App['prod '],
18         Node['prod2.example.com '] => Blog::Web['prod '],
19         Node['prod3.example.com '] => Blog::Db['prod '],
20     }
21 }
22
23 }

```

In this case we've defined the beta environment to use two machines or **nodes**. Lines 7 and 8 define node `beta1` and specify that it should run the web server (`Blog::Web['beta']`) and the app server (`Blog::App['beta']`). Line 9 defines node `beta2` and specifies that it hosts the database server. The production environment is defined to use a different machine for each of the three components (lines 17–19).

Only what is different between the beta and production environments must be listed: credentials for access to the database and the location of the docroot. Everything else relies on the underlying module definitions and parameters. (In a real example we wouldn't list the password directly in the source code. Instead, we would use a feature that retrieves passwords from a secure secrets repository service.)

That is the high-level description. For it to work, we must define the blog application and subcomponents themselves. In Puppet, this is done with code like that shown in [Listing 4.8](#).

Listing 4.8: Puppet code to define a blog application

[Click here to view code image](#)

```

1  application blog (
2      db_username ,
3      db_password ,
4      docroot = 'varwww/html ',
5  ) {
6
7      blog::db { $name:
8          db_username => $db_username ,
9          db_password => $db_password ,
10         export      => Sql["blog -$name"],
11     }

```

```
12
13   blog::web { $name:
14       docroot => $docroot ,
15       export  => Http["blog -$name"],
16   }
17
18   blog::app { $name:
19       consume => Sql["blog -$name"],
20       export  => Http["blog -$name"],
21   }
22
23 }
```

This code specifies that to create an application called `blog` (line 1), one must specify three parameters (lines 2–4): the username and password used to access the database, and the location of the docroot (where static files are located). If the docroot is not specified, it will default to `var/www/html`.

In lines 7–11, `blog::db` is invoked to define the database. `blog::db` is given the username and password as parameters on lines 8–9. Line 10 is how we indicate that the database server exports, or makes available, the `Sql[]` service. The full name of the service will be either `Sql['blog-beta']` or `Sql['blog-prod']` depending on which name is used when invoking `blog` in lines 3 and 13 of [Listing 4.7](#). The code `"blog-$name"` means to concatenate `"blog-"` plus the name supplied in the site's `blog` command. Exporting a service gives it a unique name that subsystems can refer to.

In lines 13–16 of [Listing 4.8](#), `blog::web` is invoked to define the web server, taking the docroot as a parameter. It exports `Http[]` similarly to how `blog::db` exported `Sql[]`.

The last stanza (lines 18–21) is similar to the previous one but also declares that it consumes, or depends on, the `Sql[]` service exported previously. That is, the app server requires the database.

With all this information Puppet knows how to configure five machines, each with the appropriate services. Because the dependencies are explicitly listed, it knows not to start the app server until the database is started. It knows that the web server is the last one to be started. If any of those subservices die, Puppet will rebuild or restart them.

This is a very simple example. There are many enhancements one can make:

- Rather than specifying specific hostnames, we could tell Puppet to create virtual machines and run the services on them. The virtual machines might be in our own VM cluster or in a cloud-based service like Amazon AWS.

- If the application can be scaled up by adding replicas of the web server, we could add a parameter that specifies how many replicas to create, and let Puppet create as many VMs as needed.
- Similarly, we could scale the system down to fit on one machine. A developer who needs a private clone of the application to do development work could use this approach to run the entire system on his or her laptop.

4.7 Bringing Infrastructure as Code to Your Organization

It can be intimidating to get started with IaC, especially in a preexisting environment. The best strategy is to start small. Automate one thing on one machine to get comfortable. Then manage more aspects of the system over time and build toward an environment where all changes are made via the CM system.

The first step is to pick the CM system you'll be using. Popular options include Puppet and Chef. Most CM systems are open source but have a commercial version that includes support services. The open source version is often a group of tools that you must glue together. The commercial version usually packages them together, often with additional non-open source pieces, to provide a very complete environment. This environment usually encapsulates a source code repository (or access to your own), a CI/CD system, and other components.

The next step is to start managing a single file on a single machine. This assures that the basic infrastructure is up. On Unix, a good place to start is to manage *etc*motd since mistakes will not break the machine. Next manage this file on many machines. This verifies the ability to scale the system.

Once additional machines are working, begin managing other aspects of the machine. A common choice is the NTP client configuration because it is low risk. A badly configured NTP client won't usually cause an immediate outage. Another low-risk next step is to control a set of common packages that are useful on all machines, such as common utilities that most people install anyway, and that do no harm if installed but unused. Another good next step is to control common and security settings, but be sure you select uncontroversial ones.

A popular strategy is to have the OS installation system create a very minimal configuration and use a CM system to bring it from that state to fully configured. This eliminates any checklist of manual tasks that have to be done on new machines. It assures that all machines start the same and remain managed.

Now that the system is more mature, adopt CI/CD for the system. There are

many open source and commercial options. Your organization probably already has a CI/CD system you can use. If not, any of the popular ones will probably fulfill the needs of a typical SA team.

The CI/CD system can be enhanced to add automated testing. A simple test to add is running a syntax checker on the code and stopping if there are any errors. This quick smoke test prevents obviously bad code from reaching production. Most CM systems have some kind of testing framework that makes it easy to do deeper testing.

4.8 Infrastructure as Code for Enhanced Collaboration

IaC can improve your ability to collaborate. Since all system changes are kept in VCS, you can use a VCS collaboration feature known as a merge request (MR) (sometimes called a pull request [PR]). With an MR, rather than submitting a changed file to the VCS directly, someone else reviews the change and approves it first. The VCS manages a queue of change requests similar to how a ticket system manages requests from users.

Imagine you need to change the configuration of a load balancer. You edit the IaC files as needed and send the MR to a co-worker. The co-worker reviews the change, and perhaps has some corrections or suggestions. After a few iterations and revisions the change is approved and the VCS system permits you to submit the change.

This approach has a number of benefits. First, two heads are better than one. By having someone else review the change, you are more likely to find an error.

Second, it is a good way to mentor and train new SAs. Suppose you are new to the system. Having a more experienced person give feedback is an excellent way to learn.

Lastly, you can use this approach to make change requests self-service. It can be time consuming to create a web UI that permits people to make changes to a service. By comparison, it is easy to provide documentation and permit people to send an MR for approval.

For example, at Stack Overflow anyone in engineering can request a DNS update by editing the correct IaC file and sending an MR to the SRE team for approval. This lets the SRE team enforce naming standards and prevents bad changes without requiring team members to make every change themselves. It is a way of delegating work to others in a safe way. Documenting how to edit a file is much easier than writing a web UI. Thus, errors are prevented, SRE and developers can collaborate by passing suggestions back and forth, and updates

are delegated to developers in a way that still permits SRE to have the last say.

Merge requests are commonly used to update load balancer configurations, monitoring configurations, and alerting rules, and to perform other cross-team requests.

4.9 Downsides to Infrastructure as Code

There are some downsides to IaC.

First, the learning curve can be steep. Of course, that is true for nearly all technologies. Use one system rather than spread your knowledge over many. Use merge requests to help mentor new people.

Also, some critics of IaC point out that it requires SAs to be more developer, less SA. This is an unavoidable trend. Over time SA work has moved away from physical tasks and more toward coding. In the 1980s, a chief skill of a system administrator was being able to handcraft Ethernet cables. Now such skills are considered outdated and a waste of valuable SA time. More and more we view doing tasks manually in the same way. In particular, virtualization and APIs are leading to system administrators becoming developers.

You should encourage a culture of doing things with CM. New projects should be envisioned not as “do x-y-z” but as “automate x-y-z.”

4.10 Automation Myths

There are three myths about automation that we’d like to retire.

The first is that automation is somehow dangerous. This is usually phrased in the form of “What if I make a typo and erase the world?”

The reality is that both manual steps and automation are created by people, and people are fallible. The difference is that automation can be tested before being exposed to production. People often get lazy or overconfident and skip pre-checks or don’t check their work when they are done. If these checks are encoded in the automation, they will be done every time.

Automation brings about consistency. It can bring about consistent good or bad work. However, we can use testing and software engineering techniques to measure the quality and improve it over time. Any outages, small or large, caused by automation should result in new tests so that regressions, meaning the reoccurrence of previously fixed bugs, do not happen. Software gets stronger over time as more and more edge cases get fixed.

The second myth is that manual work is of higher quality. We’ve heard people

say they feel more comfortable making the changes themselves because “I check my work thoroughly.” We would rather have all that checking done by the automation.

We’ve also heard people claim that they check their work better than even their co-workers. Are they saying that work should stop when they are away or that lower-quality work is permitted when they are on vacation? If your verification steps are actually better, then being a good team member means you will automate the process so that everyone does it your way.

The last myth is that if you automate too much, you’ll lose your job. This is false. There is 100-times more IT work that needs to be done in your organization than you and your team can accomplish. If you automate a task, it simply means that one of the long-ignored projects will finally get some attention.

We’ve been in management positions where we’ve had to reduce headcount. In those situations do we let go the person who is great at automating tasks? No, we let go the person who doesn’t automate. The person who is good at automating tasks is a workforce-multiplier and therefore very valuable. That individual is the last to be let go.

The truth is that being good at automation is a skill that every manager desires. It sets you apart from all the other system administrators. In fact, we look at our role as constantly trying to automate ourselves out of a job. This frees us up to be assigned to the new and interesting projects, of which there are many.

4.11 Summary

Infrastructure as code (IaC) is the application of software engineering best practices to system administration. A machine-processable description of the infrastructure is maintained, and automation uses this description to create and delete machines, change configurations, start and stop services, and so on. By treating these descriptions as source code, we can use a version control system to track its history, revert back from bad changes, perform tests in isolation, and receive all the benefits of continuous integration and delivery. The description plus automation software is our infrastructure.

The principles of IaC are to make all changes via definition files, document systems and processes in code, use a version control system for all files, apply CI/CD to infrastructure, apply the small batches principle, and keep services available continuously.

One IaC strategy is to use immutable elements such as containers (OCI,

Docker) to build the infrastructure. IaC descriptions are used to build the elements in a repeatable way. Another strategy is to use a configuration management system (Puppet, Chef, and others) to build and maintain the infrastructure we desire in a repeatable fashion. Using these systems, rather than home-grown platforms, permits you to leverage the knowledge and experience of others.

IaC systems tend to be declarative languages. The code describes, or declares, what the end result should be. The system figures out how to get there. This is different than imperative languages like Python and C, which describe how to achieve a goal, step by step. Declarative languages are more concise and operate at a higher level of abstraction.

IaC systems are usually idempotent. They check for the need for a change, then make the change if required. Running an idempotent system a second time activates no changes, because the changes were made the first time.

IaC permits system administrators to collaborate more effectively. Infrastructure descriptions can include extensive comments that will prove helpful to future team members. Code can be shared within the team or publicly to reduce the amount of time wasted in reinventing the wheel. Proposed code changes can be shared among team members and reviewed and approved before they are committed, adding yet another quality-assurance step.

Automation is a workforce multiplier, enabling one person to do the work of many. This benefits organizations.

Exercises

1. What is meant by infrastructure as code (IaC)?
2. What are the benefits of automation over doing system administration work manually?
3. What are the benefits of IaC?
4. Why is it important to know the history of changes to a system?
5. What is meant by software being parameterized or parameterizable? How does this make software more reusable?
6. What are some key software engineering principles or techniques that have been adopted by system administrators to create IaC?
7. What are the principles of IaC?
8. What are immutable elements? How are they used?
9. What is idempotency? Why are most CM systems idempotent?

10. How do configuration management systems differ from other software that system administrators use to control our systems?
11. Which, if any, IaC systems are in use in your environment?
12. Describe a strategy to introduce, or increase the use of, IaC in your current environment?
13. What is a declarative language? How is it different from an imperative language?
14. Suppose you are going to cook dinner. You have no ingredients, so you must buy them. Then you must mix and cook them. Finally, you serve the food at a dinner table. Suppose the world was controlled by computers. How would you write a program to make dinner using an imaginary imperative language, and then in an imaginary declarative language?
15. Learn the basics of Puppet, Chef, or some other CM system. Use it to configure a simple web server.

Part II: Workstation Fleet Management

Chapter 5. Workstation Architecture

One of the most fundamental functions of an IT department is to manage the fleet of workstations used by the organization. This chapter is about the many technical and nontechnical decisions involved and how they determine the type and quality of the service that users experience.

Workstations are computers used by people. Whether it is a desktop PC in an office or a laptop computer carried from place to place, workstations are the computers people use to get work done. There are also mobile devices such as smartphones and tablets that are increasingly being used for mainstream work tasks. Contrast this to server computers, which are generally hidden from view in computer rooms providing services or performing computations remotely.

The experiences people have when using their workstations are determined by design decisions you make as the system administrator. Issues that are important to customers are

- **Locality:** Is the workstation available where and when it is needed? A workstation is not useful if it isn't where the user needs it.
- **Reliability:** Is the workstation reliable or does it lock up or crash frequently? Reliability starts with good hardware but increasingly depends on software and firmware updates being received in a timely manner. If it breaks, can it be repaired quickly?
- **Productivity:** Can the customer work with minimal obstruction? Is access to features and services wonderfully fluid and convenient, or painstakingly tedious and frustrating? How much time each day is spent doing productive work versus extraneous labor required by the system itself?
- **User agency:** Do the users have agency or control over their environment? Can they fulfill their creative potential or are they restricted to a predetermined set of features? Do they have control over which applications can be installed or is the configuration highly controlled?
- **Currentness:** How much lag time is there between when new features ship and when they are installed and ready to use on the machine?

What is important to system administrators is that the machines are easy to manage. To achieve this we limit variation between machines and automate OS installation, configuration, and updates. These goals can be the means to achieving the qualities that are important to customers. For example, if software

updates are automated frequently and quickly, the workstations will be more stable as these updates fix problems and introduce new features.

Conversely, if a fleet is not easy to manage, then the SA's time will be filled with emergencies and firefighting that result from manual processes and outdated software. There will be no time left over to create features and services that enhance user agency and productivity.

5.1 Fungibility

To the best of our ability, workstations should be a fungible resource: Any one unit should be able to substitute for any other. It should be possible for anyone with an account to log into any workstation. The applications the customers need should be immediately available. The files they need to access should be stored on network file servers, not local disk, so that they are accessible anywhere. The operating system and applications should be updated automatically with minimal or no user involvement. The user of a workstation should be able to customize it but not break it or subvert its security.

The ability to log into any workstation improves access locality. In a large desktop environment, it means a person can sit in front of any computer, log in, and begin to work. This is convenient when someone is working in a different part of the building, away from his or her laptop; when someone wants to give a presentation from a PC embedded in a podium; in a call center that needs to assign people to different cubicles depending on shifting demand; or if you forget your laptop at home and need a loaner for the day.

Computers break. If a person is tied to a particular machine, he or she is unable to work until the computer is fixed. This creates pressure to repair the machine quickly. Pressure leads to mistakes, which makes repairs take even longer. Recovering data from dead machines can be difficult and time consuming. If machines are generic, the customer can be handed a new machine and return to being productive. Alternatively, the customer can be handed a loaner machine for use until the original machine is repaired. Either way, the situation is better for both the customer and the person doing the repair.

When the software on a machine is corrupted, it is convenient to simply wipe and reload the machine from scratch. The more generic a machine is, the easier it is to do this.

Sadly, there are limits to how close we can get to this ideal of a 100 percent fungible system.

There will be variations in hardware. Within an organization some people

need laptops while others need desktops. Vendors stop selling older models as newer ones are introduced. Some customers require workstations with additional RAM, faster CPUs, and larger hard drives or displays. Managing these variations is made easier when they are abstracted away by the operating system. Some will require entirely different management methods and infrastructure.

There will be variations in login access. Ideally, anyone should be able to log into any machine, but security precautions may restrict that from being permitted. Information stored on the local machine might be accessible by others who log into the same machine. You may wish to prevent, for example, a workstation used by someone who manages confidential human resources information from being used by anyone else. This kind of variation can be managed by designing for the ideal of fungible machines, but layering on top policies that restrict logins on specific machines.

Preventing Access to Another User's Data

If a user sits down and logs into someone else's computer, will the user be able to access that person's files?

Unix was designed to be multiuser from the start and therefore has file permissions and other security controls as part of its foundation. It is very difficult for a non-root user to access the local files of another user if the file permissions are set correctly. However, sloppy configuration could leave those permissions wide open.

Microsoft Windows started out as a single-user operating system. Previously, if a second user logged in, that person would be able to access any local files left behind by the previous user. The situation has greatly improved over time as new security features, such as NTFS file permissions, have been added. However, to remain backward compatible with older software, these features are often disabled or not used.

Another variation from machine to machine is the applications and software installed. Not everyone needs every software package. It would be wasteful to pay for software licenses on machines where the software will not be used. It would also consume a lot of hard disk space. Updating and maintaining them all would be a waste of resources. Plus, it would reduce the security of the system: The most secure application is the one that isn't installed. Extra applications increase the surface area of attack, without adding any value. Luckily we can control this variation by having a "base install" that is standard for all machines

and limit the variation to which additional packages are on machines.

There will also be variations in the version of software on each machine. We wouldn't want to upgrade from version 1.0 to version 2.0 of a software package on every machine in our fleet at the same time. If the new software had problems, the entire company would suffer. It is better to upgrade machines in small groups, so that problems can be found and fixed before they are exposed to the entire company.

Lastly, having all data be stored remotely and accessed over a network requires network access. However, network access is not always available to laptops or other mobile devices, nor to workstations when used in a remote area or disaster areas.

Now that we have covered our ideal architecture, its limits, and its unavoidable variations, we are ready to take a deep dive into the individual elements that create this architecture.

The primary elements of a workstation architecture are the hardware itself, the operating system(s), network configuration, the accounts and authorization system, data storage, host security, and logging.

The following sections provide overviews of each. We will discuss some typical configurations in later chapters.

5.2 Hardware

Selecting the physical hardware is the most fundamental choice one can make. There are laptops and desktops, mobile devices and tablets, and other physical form factors to choose from.

We assume that readers of this book have basic knowledge of hardware concepts such as the difference between a laptop and a desktop, multiple vendors, and multiple product lines within each vendor. There is also the decision between physical workstations and virtual workstations, and whether the workstation is provided by the company or if some kind of bring your own device (BYOD) strategy is employed. These decisions will be discussed in greater detail in [Chapter 6](#), "[Workstation Hardware Strategies](#)."

5.3 Operating System

The operating system is the software layer between the applications and the hardware. It is the mediator that enables many applications to share the same computer.

As part of our workstation architecture, we can provide a single operating system or many. There are major OS groups, each with small variations within the group. The major groups are usually related to particular vendors: Microsoft Windows, Apple OS X, and Linux are major groups. Within each of those are smaller variations: Microsoft and Apple have different generations of OSs released over the years.

Linux has both different vendors and versions within those vendors. For example, RedHat, Ubuntu, Debian, CoreOS, and other vendors make Linux distributions (“distros”) by packaging the Linux kernel with different sets of applications, utilities, and enhancements. Each distro releases major new versions periodically, with minor updates being provided in between.

Similarly, Microsoft and Apple have server and workstation variations of each operating system. Server editions are bundled with additional features or tuned with different defaults.

The decision whether to support one or many operating system groups is similar to the discussion about supporting different hardware models. Minimizing variety results in a system that is easier to support. Standardizing on exactly one operating system lets you put “more wood behind fewer arrows.” That is, the support organization can become very good at all aspects of the one supported operating system.

To an organization that has historically supported only one operating system, adding support for a second one is a big leap. It doubles the support infrastructure: systems for automatically loading the OS, patching it, and so on. It also doubles the expertise the organization must maintain. One strategy is to hire people who know both systems. This can be expensive since people with expertise in two very different OSs are difficult to find and therefore can command higher salaries. Another strategy is to increase the team size to accommodate specialists in each OS. This approach also has additional cost and complexity.

Whether there is one standard OS or multiple OSs, there is also variation in minor releases. Does one support all versions of Windows, just the most recent, or a subset? A well-defined subset is best.

For example, one organization always had two officially supported OS version: the current one and the transition one. At one point Windows 7 was the currently supported OS. When Windows 8 was released, all corporate applications were validated, Windows 8 became the current supported OS, and Windows 7 was designated as the transition OS. Users could use Windows 7 but

were expected to convert to Windows 8 by a certain date. The project management office developed a plan to assure all Windows 7 machines were identified and converted. When Windows 8.1 was released, it was not deployed until the last Windows 7 machine was removed. This maintained the policy of only supporting two OS versions at any given time. This also motivated the system administrators to completely eliminate the use of Windows 7, since they were eager to deploy Windows 8.1. This strategy also helped set timetables for deprecating old OSs. If the old OS was eliminated faster than Microsoft released new operating systems, the SAs knew they were moving at a good pace. Eliminating the old release significantly faster had the benefit of giving them more time to plan for the next release. Eliminating the old release too slowly hurt their own plans, which provided additional incentive to stay on track.

5.4 Network Configuration

Workstations are generally connected to a network by either wired (Ethernet) or wireless (WiFi) technologies. These options will be discussed in more depth in [Chapter 23, “Network Architecture,”](#) and [Chapter 24, “Network Operations,”](#) which focus on networking.

Architecturally speaking, the decision to be made here is whether network configuration is hardcoded (stored on the machine itself) or dynamic (provided by the network). These configuration parameters include the machine’s IP address, subnet netmask, default gateway, DNS servers, and more.

5.4.1 Dynamic Configuration

In a **dynamic configuration**, the machine asks the network what its configuration parameters should be. More pedantically, it talks to a service on the network, which responds with those parameters. In Internet Protocol versions 4 and 6 (IPv4 and IPv6), this is the Dynamic Host Configuration Protocol (DHCP) service. How a machine can talk to the DHCP service over the network to find out how to talk to the network is a Catch-22 that makes the design of DHCP so interesting. IPv6 has the additional possibility of using a system called Neighbor Discovery (ND), which provides the minimum parameters required for an IPv6 host to configure itself in the absence of a DHCPv6 server. However, this should be considered a stop-gap option for sites that have not yet deployed a DHCPv6 server.

Dynamic configuration permits network configurations to be controlled centrally. This benefit cannot be understated. The ability to centrally control configurations is key to being able to efficiently manage large numbers of

machines. Otherwise, one would have to visit every machine for even the most minor change. Central control of machine configuration is further discussed in [Chapter 4](#), “[Infrastructure as Code](#).”

Even sites without sophisticated configuration management systems can achieve many of the same benefits for a host’s network parameters by using dynamic configuration. For example, there may be a need to change the netmask size of a particular subnet. When the DHCP server is reconfigured with the new netmask size for the particular subnet, each machine on that subnet will update itself as it periodically contacts the DHCP server to check for new settings, a process called renewing a lease.

5.4.2 Hardcoded Configuration

In a **hardcoded or static configuration**, the configuration parameters are stored on the machine itself. A hardcoded configuration has the benefit that it works whether the DHCP server is available or not. Servers generally use static configurations so as to minimize their external dependencies. For example, a file server’s IP address generally does not change. Having it depend on the DHCP server to boot is an unnecessary dependency. Of course, a DHCP server machine requires a static configuration because it can’t configure itself!

5.4.3 Hybrid Configuration

Servers can also have a **hybrid configuration**, where the network parameters are defined in local files, but the server periodically checks the network parameters via DHCP. This technique uses the DHCP INFORM request to get parameters such as DNS servers, NTP servers, and so on. If changes are detected, the system updates its live and stored configurations accordingly. This approach combines the ease of managing a large-scale installation with the benefit of removing the boot-time dependence on DHCP for critical servers.

5.4.4 Applicability

You may have noticed that workstations are squarely placed into the dynamic configuration category. Storing a static configuration on a host makes it less generic. It can plug into only the one subnet it was configured for.

By being dynamic, a laptop can float between WiFi connections in the office, home, or a coffee shop. Desktop PC users benefit from dynamic network configuration because it reduces the skill required to deploy or move their PC. If employees are being moved to a new office, they can move their own desktop

PCs as long as they are informed which network jack to plug into. If an office space reorganization requires moving many people, dynamic network configuration means the IT work can be done in advance instead of requiring precisely choreographed sequences of moves and reconfigurations.

Dynamic configuration is also less error prone. Expecting a user to manually type an IP address, subnet mask, and default gateway accurately is overly optimistic. Static configuration is for SAs. Dynamic configuration is for end users; it empowers customers to be self-sufficient in a safe way.

The Importance of DHCP

Before DHCP came into widespread use in the late 1990s, users would manually reconfigure their laptop each time they took it home and then again when returning to the office. If a desktop was being moved from one office to another, customers would open a ticket listing the port number in the new office, and request an IP address on that port's subnet.

Early in the authors' careers we processed many such network change tickets: carefully allocating an IP address from the host file; verifying that someone wasn't surreptitiously using it; and replying to the ticket with the IP address, the proper netmask, default gateway, and other settings. An important skill for a system administrator at the time was to be able to walk a customer through setting these parameters on a variety of operating systems and debug the typical problems that resulted from typos.

Occasionally customers would get confused and enter the gateway's IP address where they should have entered their host's IP. This would cause an outage for all users on that subnet as the router and the desktop fought over who could use that IP address. If you thought asking a barista for a WiFi network name and password was inconvenient, imagine what life was like before DHCP. If DHCP hadn't been invented shortly before WiFi, WiFi would have been a flop.

5.5 Accounts and Authorization

Workstations need a database of usernames, passwords, groups, and related account information. Architecturally speaking, the decision to be made here is whether this information is stored on the machine itself, is accessed from a central database over the network, or both.

All machines have a local account database. Unix has *etcpasswd* and *etcgroup*. Windows stores local accounts in the Security Accounts Manager (SAM).

When this information is made available as a network service, it is generally called a network directory. The most common example is Microsoft ActiveDirectory, though there are others such as Apple's OpenDirectory and RedHat Directory Server. All of these are based on the LDAP protocol for database-like access to this information and Kerberos for the secure handling of passwords and authentication. Such services will be discussed in greater detail in [Chapter 40](#), "[Nameservices](#)." Older systems such as Hesiod, NIS, and NIS+ have mostly gone away and are now likely to be found only in computer museums and history books.

What's So Light About LDAP?

The Lightweight Directory Access Protocol (LDAP) is anything but lightweight. Implementations require hundreds of thousands of lines of code and are very complex. Sadly, compared to the X.500 system's Directory Access Protocol (DAP), which it was modeled after, it is actually relatively thin.

With the use of a network directory, the same information is made available to all machines. The directory stores the username and password, and which machines the account is permitted to log into. Therefore someone can use the same username and password on any (permitted) machine. The administrator doesn't need to create the account on each machine individually, nor does the information need to be copied or synchronized to all machines after each change. If the user leaves the organization, the account is disabled in the directory, rather than requiring system administrators to lock the account on every single machine.

Before network directories, users had to change their passwords on every machine they had access to. This was acceptable when an organization had one or two large mainframes, but became unworkable when organizations switched to individual workstations.

Storing access policies in a directory means they are applied consistently everywhere. A policy might record that anyone with an account can log into some machines, but other machines are designated for use by particular people exclusively.

Having account information available to all machines is important even if the user doesn't have login permission for a specific machine. For example, listing the files on a file server shows the file name, file owner, and other information. Generating such a file list requires translating numeric user IDs into textual usernames. For example, the numeric ID 2034 could be the username `cjlear`. Even though `cjlear` may not have permission to log into your workstation, being able to list files and see that they are owned by `cjlear` is fundamental.

This brings us to an important point: There is a difference between identity, authentication, and authorization. **Identity** is storing information about a user. In the previous example, it is storing the fact that ID number 2034 is associated with the username `cjlear`, and vice versa. This information is made universally available to all subscribers of a network directory service so that it can be used to provide a consistent experience for all users on all machines. **Authentication** is using that information plus secrets (usually passwords) to prove that someone is who he or she claims to be. For example, a user entering a username and password, and the operating system verifying this information, is an authentication process. An authentication process must be carefully secured to prevent fraud. **Authorization** is what a user is permitted to do. For example, once a person is authenticated, whether or not that user is permitted to log into a particular machine is an authorization policy. Authorization comes after authentication.

While network directory services are important, local account databases still have their place. One benefit of a local account database is that the information is always available, whether or not the machine is connected to a network and even if the network directory server is down. For example, being able to log into a local administrator account—one whose credentials are stored in the local account database—is often required to fix a damaged or misconfigured machine that is unable to talk to the network. The ability to store different information in the local account database of each machine has benefits. For example, some sites set a unique password on the local administrator account of every machine. If a user requires administrator access, that individual can be told that particular password, instead of the password to an account in the network directory that has privileged access on all machines.

Having a local account database also permits a user to create a guest account for a co-worker or friend. This might be convenient, or a terrible breach of the organization's security policy.

When looking up information, systems generally check the local database first, then one or more network directories. They cache the results for

performance and for use if the machine is off the network. Offline access is often called roaming or a roaming profile and is particularly useful with laptops.

Home users without large IT infrastructures missed out on the benefits of a centralized network directory until around 2013. Now macOS will store passwords in the iCloud service, and Windows 8 and later permits logging in via a personal Microsoft Account (MSA). These services are convenient for home users but should be disabled by companies so as to not store security credentials outside their own control.

Importance of Network Directory Services

Whoever controls the directory controls the IT department. For a full decade, 1985 to 1995, Microsoft and Novell publicly dismissed the utility of directory services for a very good reason: They didn't make a directory service product. Novell and later Microsoft realized the strategic value of directory services. It was becoming an administrative nightmare, and increasing their development cost, to create product after product that had its own internal account directory. As Microsoft moved from making desktop applications to focusing on client-server products, it saw the light. Soon Novell Directory Services (NDS) and Microsoft ActiveDirectory were born and fought intensely—and sometimes unethically—until finally ActiveDirectory won.

5.6 Data Storage

The user of a workstation needs to store information, or state. Generally this is in the form of files and directories but also includes configuration and customizations that users make to their environment.

There are three fundamental ways to configure storage:

- **Local:** All data files are stored locally. The system is configured so that the user's files are stored on the local disk, which is also used to store the OS and any OS-related files and state. Local disks are often called direct attached storage (DAS).
- **Stateless:** No locally unique data. Users' files are stored remotely on a network server. Any information that is stored locally is a copy of data that is stored elsewhere. The local disk is used only for the OS, temporary files, and caches. Ideally, if the local disk was wiped, the only information that would be lost would be disposable temporary files and cached copies of files whose master copies are stored on the remote server.

- **Diskless:** No local disk storage. The operating system and user data are stored on a remote disk over a network using protocols such as iSCSI or NFS. If the machine dies, the same chunk of storage can be attached to a replacement machine. This setup is most commonly used in virtual machines and blade systems, discussed later. Often many diskless machines share a common read-only image of the base operating system, saving storage on the server. Changes (deltas) from the base image are stored on a separate network disk that is unique to the machine.

Local storage is generally the default for most operating systems. Local storage is simple to set up and always accessible. The primary drawback to local storage is that it is risky. Disks fail. Users accidentally delete data or need to view a file as it was many months ago. Therefore, such data needs to be backed up. This can be difficult, especially for machines with no or slow network access. Laptops are particularly difficult to back up because they are not always powered up. Data stored on a server is easier to back up and can be more cost-effectively protected through RAID and other techniques, which are discussed in [Chapter 43, “Data Storage.”](#)

Local storage is generally faster than network-based storage. Network storage has a reputation for being slow, although there are counter-examples. For example, if the workstations have slow disks and the file server has high-end storage technology, the result can be network storage that is faster than a local disk. The added cost required to successfully achieve this performance is amortized over the many workstations being served. If it permits less expensive workstations to be purchased, it can save money.

Traditional Disks or SSD?

Solid state drives (SSDs) are the best option for for laptops and desktops. As the price of SSDs drops, they will become standard for all workstations.

Traditional disks are made of mechanical parts that rotate. To read information the disk must spin until the read/write head is hovering over the proper location of the data. This wait exceeds the amount of time required to read the information many times over.

SSD devices use flash memory to provide storage. Flash memory has no moving parts and can access any information with minimal delay. Though they may be slower for certain operations, such as performing a continuous stream of writes, for workstation use, SSDs are generally a net

gain.

As of this writing, SSDs are more expensive than traditional hard drives. However, the price is falling. The performance improvement is well worth the additional cost, especially if one can put a price on the productivity-killing frustration of a slow machine. For laptops, SSDs are a win because they are more resilient to damage when dropped, plus laptop hard drives are often significantly slower than desktop hard drives, which accentuates the performance benefit of SSDs.

Early SSDs emulated old-fashioned hard drives by providing a SATA interface, thus making them plug-compatible replacements. This helped them gain acceptance, but added the cost of the additional connectors and electronics that provided the emulation. Cost was reduced by connecting the SSDs to the PCIe slots and shifting the emulation to software. Some vendors have integrated the SSD components into the motherboard, improving performance and reducing costs further.

By the time you read this, the price of an SSD may well have dropped to below “spinnny” disks, making the decision a no-brainer.

A big part of creating generic, fungible workstations is to make them stateless. This can be done a couple of ways:

- **Remote file storage:** The client accesses storage from a remote server as a “network disk” or other mechanism that makes the files appear as if they were local, but actually accesses them from another server behind the scenes. Examples include NFS, popular on Unix/Linux systems, and SMB, popular in Microsoft Windows.
- **Network-synced or cloud storage:** The user’s files are stored locally but copied to a network service as soon as possible. Examples of this include Dropbox, Microsoft OneDrive, and Apple’s iCloud Drive. Files are stored and manipulated locally but any changes are synced to the copy held on the network service. The files are also accessible via an API that enables mobile device and programmatic access, such as web applications that can save files directly to the network drive. A web user interface (UI) is generally also available.

Dataless configurations work well when the user has a large amount of data, more than could fit on the local machine. For example, the user may have terabytes of data on the server, but only a small hard disk on a laptop. The laptop may cache a tiny fraction of the full dataset to improve performance, but this

cache may be tiny enough to fit in RAM or it may use spare local disk space.

Network-synced storage requires enough local storage to hold all of the user's files. This generally means that the local storage capacity must be large, or the amount of data being stored is relatively small. If the user has large amounts of rapidly changing files, syncing to many devices may be unwieldy. Network-synced storage products generally have the ability to exclude certain directories from being synced, making it possible to have large amounts of data on the network service but a manageable subset that is synced to workstations. For example, one might have a large archive of videos that are not synced to the local machine. However, this creates a burden on users.

Another difference between these options is how they handle situations where the network service becomes inaccessible due to a lack of network connectivity or the network service itself being down. SMB will disconnect the network mount point. Software that had been accessing the data will report this as an error, much like how software reacts when a USB thumb drive is removed unexpectedly. An NFS client will wait forever until the server can be contacted again. The entire machine will hang until the server returns to service.

Network-synced storage keeps working when network access is lost. All files are stored and manipulated locally. Syncing happens after the fact. If the network service cannot be accessed, syncing simply pauses. This is called offline mode. Offline mode is particularly useful for users who are disconnected from the network for long stretches of time such as while traveling. When finally reconnected the system performs a "catch-up sync," bringing the network service up-to-date.

In both remote file storage and network-synced storage, the same files can be accessed from many machines. This is very convenient to users, as they see the same files everywhere. If a new file is created from one machine, it is soon visible to all the others.

Simultaneous access to the same file by multiple machines requires special coordination; otherwise, the data may become corrupted. Some systems default to uncoordinated simultaneous access. Updates to the file arrive and are applied in order. If two machines try to append to the end of the same file, the two new chunks of data may be appended in any order simply due to the randomness of which append request arrived first. If two machines are writing to the middle of the file, the result will be unpredictable.

Imagine, for example, a situation where one machine is running a program that will read a file, sort the contents, and write the new, sorted information back

to the file. Another program is appending data to the end of the file. Depending on the order in which the file server receives the various instructions, the newly appended data could end up at the end of the otherwise sorted file, appear at the beginning, or disappear altogether. The results are determined by chance or luck. If such uncoordinated updating is the only option, it is best to take measures to assure that simultaneous updates to the same file never happen. This can be as simple as doing all updates from a single machine.

Better simultaneous access requires coordination. There is usually some kind of protocol for one machine to gain exclusive access to a file so that it can perform updates unmolested by other machines. This is called a file lock. The file is locked by one machine, and now no other machine can access the file. The first machine updates the file, then unlocks it. Other machines now see the updates and can have their turn at manipulating the file.

SMB and other protocols automatically lock a file when it is being accessed. NFS requires software to specifically lock the file.

Network-synced storage generally has an API for file locking. However, offline mode creates special circumstances since, by definition, the API cannot be used while offline. The longer a machine is in offline mode, the greater the chance that another machine will have modified a file. When the offline machine reconnects and goes to sync the file, it may find itself syncing files that have changed out from under its own control. Some products provide sophisticated reconciliation mechanisms. However, the simplest method is just to detect that the situation has occurred, and move the updated file out of the way, perhaps to a name that includes the date and time of the collision. Since we are dealing with human users, they will see the multiple file names and reconcile things manually.

To make workstations fungible resources, use dataless configurations. Remote file storage is best for desktops with permanent network connections. It is also best when many machines need access to the same data, if the amount of data is too large to store locally, or if syncing would be unwieldy. Network-synced storage is best for laptops or situations where offline access is needed.

5.7 OS Updates

There must be a way to update the OS and application software on the machine. Software is never “done”; it can always be improved. As long as software is being actively supported by the vendor, there will always be updates, often called **patches**, that add new features, fix bugs, and close security holes.

Expecting software to never need updates is irrational. You must have a strategy

for how software updates will be deployed. Keeping machines up-to-date is part of good system hygiene.

In addition to fixing existing software, update mechanisms enable one to deploy new software. For example, the same mechanism may be used to deploy a new corporate application or tool on all machines.

We can install updates manually or automatically. Manually means a person visits each machine and gives the commands to install the updates, possibly rebooting the machine if that is required. Visiting machines can be done either physically or through remote access mechanisms such as Remote Desktop Protocol (RDP) or Secure Shell (SSH). Automatically means using a software system to perform the upgrades without system administration intervention. Automatic updates may happen on a schedule, such as weekly, or on demand, such as by a system administrator approving them for groups of machines.

Automated updates can download the updates in the background, and give the user the ability to schedule the installation and reboot, perhaps with a limit on how long users can postpone the updates.

Updates should be automated for a number of reasons. The first is scale: Over time you will acquire more machines and therefore any manual process becomes a bigger drag on your time. Second, automation assures completeness. If updates are important, then they are important when you are on vacation, too. Third, automating this process assures that the updates actually get deployed. Imagine visiting every employee in the company to manually update their machines. You might walk down the hall to visit each person. However, some people will be out or unavailable. You could track who you missed and make appointments to revisit them. If you are lucky, you'll eventually visit every machine. However, it is more likely that the people you visit will be unable to reboot at this time for some reason. You'll annoy them by trying to reschedule and find a time when you are both available. More likely they'll start to avoid you.

Even more likely, you will end up not installing these updates at all. Updates tend to be important but not urgent. There is always some other project that is more urgent and cannot be put off. By not installing these patches, systems become less secure and less reliable. A vicious cycle begins as updates are delayed, which increases the number of security issues and outages you have to deal with, which gives you less time to manually install patches, which causes more outages.

Lastly, updates should be automated because the work is boring and you deserve better.

Your workstation architecture should take the following considerations into account:

- Centrally control which patches are distributed when.
- Test updates before they are deployed.
- Distribute updates to a small group of users first, then to larger groups. Thus problems are detected early.
- Users should be able to delay an update, but perhaps by only a day or two.
- A dashboard should display which machines haven't been upgraded. Such machines should be tracked down to identify the cause of the delay.
- SAs should have the ability to stop all updates if a problem is detected.
- SAs should be able to rapidly deploy a specific update in an emergency, overriding user approval.

This topic will be covered in greater detail in [Chapter 7](#), “[Workstation Software Life Cycle](#).”

5.8 Security

Your workstation architecture has many security-related implications. We've already discussed accounts and authorization policy and the ability to distribute security updates. There is also the need to defend machines against data theft, malicious software, and changes to the network firewall configuration.

5.8.1 Theft

If a laptop is lost or stolen, the first priority is to make sure that whoever possesses the device cannot access the contents. We would also like to recover the device.

Laptop tracking software periodically announces the machine's IP address to a service that helps you track its location. In the event that the device is missing, the service can help you locate the machine. Such software often has a remote-wipe feature, which is a way to instruct the lost machine to erase its contents. Some tracking software will enable cameras and microphones and send periodic updates that help you track the device.

Another way to protect access to the information on the hard drive is to use full disk encryption (FDE). FDE encrypts the hard disk in a way that, without the decryption passphrase, makes the data on it basically unusable. At most companies FDE is standard practice for laptops and mobile devices; some companies do it for desktops, too. The downside is that disk access will be

slower and the user will have to enter the passphrase each time the system boots. There is also the issue that if the passphrase is lost or forgotten, the data on the device becomes inaccessible. There is no way to reverse-engineer the key. As a precaution, enterprise-grade FDE software securely stores copies of passphrases or decryption keys to a key escrow. A key escrow is a secure database designed specifically for such information. The key escrow function should be automated and transparent.

Recovering lost and stolen hardware is relatively rare. Organized thieves know to immediately power it off to evade tracking software. Most thieves will wipe the storage immediately and sell the hardware for fast cash. Most organizations assume the device will not be returned, so they activate the remote-wipe feature in case it ever does phone home, but otherwise replace the device and assume FDE will do its job. The users' accounts are disabled until passwords and security credentials can be reset. A police report is usually required if the device is insured.

5.8.2 Malware

Malware is software created to insert itself on a machine to subvert the security of the system. Often this takes the form of making use of resources or stealing information. Malware goes by many names: virus, Trojan, backdoors, spyware, and so on. Anti-malware is software that detects these programs and disables and reports their existence. Anti-malware software comes in two general categories:

- **Antivirus software/blacklisting:** Detects malicious software by watching for specific banned software, or detecting particularly bad behavior. It uses a blacklist of banned software and activity but permits all other software to run.
- **Application control software/whitelisting:** Uses a whitelist of permitted software and prevents all other software from running. This technique is relatively new, becoming generally accepted in 2012. The premise is that initially machines run in “learning” mode, which gathers information about which software is in use in the organization. After that, the software runs in “detect” mode, which reports any new software programs. Lastly there is a “prevent” mode, which not only detects new software, but also stops it from running. A centralized control panel is provided to approve new software.

Any kind of security defense software should have the following qualities:

- **Centralized control:** Security defense software should be configured and

controlled from a central point. The user may be able to make some adjustments but not disable the software.

- **Centralized reporting:** There should be a central dashboard that reports the statuses of all machines. This might include which viruses have been detected, when the machine received its most recent antivirus policy update, and so on. Knowing when the machine last checked in is key to knowing if the software was disabled.
- **Silent updating:** The software should update silently. It does not need to pop up a window to ask if the new antivirus blacklist should be downloaded and installed. That decision is made by system administrators centrally, not by the user.
- **Hidden from view:** The user should be able to determine that the software is activated, but an animated spinning status ball or pop-up windows aren't needed announce that updates were done. Such gimmicks slow down the machine and annoy users.
- **Negligible performance impact:** Anti-malware software can have a significant impact on the performance of the machine. Examining every block read from disk, or snooping every network packet received, can use a lot of CPU, RAM, and other resources. When selecting anti-malware software, benchmark various products to determine the resource impact.

A network firewall is a software-enforced policy that determines network access to and from a host. For example, the policy might be that the software running on the machine can send outbound network traffic, but the only inbound network traffic permitted is replies to the previously initiated outbound traffic plus certain other designated inbound traffic such as RDP access from specific authenticated sources.

Firewall configuration is particularly important for machines that leave the corporate network. There is an assumption that being on the corporate network is safe and being outside of it is dangerous, so the firewall configuration may be less strict. This is called perimeter-based security. Perimeter-based security has become less effective as corporate networks have become more porous.

Consumer Antivirus

While anti-malware products should be silent and nearly invisible, many are flashy and noisy. Why is this?

Anti-malware software sold to the consumer needs to be visible enough that users feel as though they're getting their money's worth. Imagine

what would happen if the product ran silently, protecting the user flawlessly, popping up only once a year to ask the user to renew for the low, low price of \$39.99. The company would go out of business. No one would renew given that the product appeared to have done nothing for the last 12 months.

Profitable anti-malware companies make sure their customers are constantly reminded that they're being protected. Each week their software pops up to say there is a new update downloaded and ask them to click "protect me" to activate it. Firewall products constantly ask them to acknowledge that a network attack has been defended, even if it was just an errant ping packet. Yes, the animated desktop tray icon consumes CPU bandwidth and drains laptop batteries, but that spinning 3D ball reassures the user and validates their purchase decision.

Would it have required less programming to simply do the update, drop the harmful packet, and not display any pop-ups? Certainly. But it would have reduced brand recognition.

All of this works because there is an information deficit. Bruce Schneier's blog post, "A Security Market for Lemons" ([Schneier 2007](#)), explains that typical consumers cannot judge the quality of a complex product such as security software, so they are vulnerable to these shenanigans.

However, you are a system administrator with technical knowledge and experience. It is your job to evaluate the software and determine what works best for your organization. You know that anti-malware software should rapidly update itself and be as unobtrusive to the users as possible. Your decision whether to renew the software will be based on the actionable data made visible by the dashboard, not due to the excitement generated by spinning animations.

5.9 Logging

A workstation architecture needs to make workstations observable. This is done by logging events and making this information accessible.

We cannot see everything going on inside a machine, and if we could, trying to watch every machine ourselves, 24 hours each day, does not scale. Instead, machines log various information as they go about their business. They log which programs ran, problems and errors, security policy violations, and more.

These log events need to be collected. Microsoft Windows calls this the event log, and Unix/Linux systems call it the system log, or syslog.

Logs are recorded to disk locally and can be accessed using various log viewing programs. Another strategy is to forward a copy of all log entries to a central machine for centralized storage, management, and viewing. This way we have global visibility from one seat.

Storing logs centrally enables more sophisticated analysis. One can observe a single person moving from machine to machine, or determine the earliest date a particular problem first occurred and correlate it to another event, such as a software upgrade. There are many log analysis tools such as Logstash and Splunk that store and interpret log information.

Large environments use a hub-and-spoke model. Each group of machines forwards a copy of their logs to a hub. There may be one hub per building or corporate division. Then all the hubs consolidate the logs and send them to a central repository.

5.10 Summary

Workstations are the computers people use, whether they be desktops, laptops, or mobile devices. The architectural decisions you make affect locality (where the machine can be used), reliability (how often it is available), productivity (whether users are able to efficiently get their jobs done), user agency (whether users can control their environments and invent new functionality), and currentness (how frequently new features and updates are applied).

Ideally a workstation should be a fungible resource, meaning any one unit can substitute for another. Users should be able to access their data from any workstation. If a workstation breaks, the person should be able to go to any other workstation. However, there are limits to this flexibility. If one user leaves confidential files behind, the next user may have access to them. It's best to design with fungibility in mind, but restrict user access as the reality of security limits require.

The elements of the architecture include the physical hardware model (laptop, desktop, vendor model), the operating system (Microsoft Windows, Apple macOS, or Linux; version and release), network configuration (static or dynamic), accounts and authorization (network based or locally administered), storage (data and other state stored on the machine or remotely), OS updates (how they are done and approved, and how frequent they are), security (defense against attacks as well as access restrictions and other issues), and logging (a

record of the history of what that machine did).

Exercises

1. This chapter's introduction lists issues that are important to customers, such as locality, reliability, and productivity. Which decisions affect those issues?
2. Consider your organization's workstation architecture. How have the decisions affected the issues listed in this chapter's introduction?
3. What are the benefits and downsides to fungible workstations?
4. What are the benefits of supporting many different operating systems? What are the benefits of supporting only a single operating system?
5. In your organization's environment, how many operating systems are supported? Include vendors and types plus variations within those.
6. List the different network configuration options and which situations they are most appropriate for.
7. In your organization's network, which network parameters are distributed via DHCP? How might you find out the exact settings that, for example, your laptop receives when connecting via WiFi?
8. This chapter warns against configuring a file server to use DHCP. What would happen if a file server was configured to use DHCP and, due to a power outage, the file server and DHCP server rebooted at the same time?
9. What is the difference between network identity, authentication, and authorization? Relate these to the process of logging into a virtual private network (VPN) system, using Facebook, or accessing a web-based payroll application.
10. What does it mean for a workstation to be stateless or dataless?
11. In your organization, which kind of storage is used for laptops? For desktops?
12. What is full disk encryption (FDE)? Does your workstation use it?
13. What are the benefits of having OS updates installed automatically versus manually? With or without user approval?
14. How are OS updates handled in your organization's environment?
15. What are the pros and cons of antivirus versus application whitelisting?
16. In the future, will antivirus or application whitelisting be more relevant?
17. Which kind of security defenses are deployed on the workstations in your

organization?

- 18.** Why do consumer antivirus products include cute animations and prompts?
- 19.** What kinds of information might one find in the workstation logs of a machine?
- 20.** How does your organization manage workstation logs?
- 21.** Why are logs called logs? Research the history of this terminology and why we “log in” to our computers.

Chapter 6. Workstation Hardware Strategies

This chapter discusses the three fundamental strategies related to workstation hardware. The first strategy is to provide physical machines such as laptops and desktops. We'll discuss how the product choices vendors make available affect the type, quality, and efficiency of service we can provide. The second strategy is virtual desktop infrastructure (VDI), or virtual machines accessed through various means. The last strategy is to not provide workstations at all. Bring your own device (BYOD) involves providing access to applications and services using the hardware your customers already have.

Most large companies have a mix of these strategies, as they migrate toward VDI and BYOD in an effort to reduce costs and embrace mobile computing.

6.1 Physical Workstations

In an environment where physical hardware is used for people's workstations, whether the user has a laptop or desktop is one of the most fundamental choices to be made. Other considerations include vendor selection and product line selection. We will look at each of these in more detail.

6.1.1 Laptop Versus Desktop

We assume that readers of this book have basic knowledge of hardware concepts such as the difference between a laptop and a desktop. However, some differences are not so obvious.

Desktops are generally more expandable. They have slots for add-on cards and video processor cards, and sockets for additional memory. They have internal space for additional storage. While both laptops and desktops have ports to permit external expansion (USB, Thunderbolt, eSATA), desktops have more surface area where designers can potentially place such ports. As a result, even "fixed-configuration" models (devices with no expansion slots) may be highly expandable.

Laptops are generally more expensive than comparable desktop models. A laptop's mobility requires it to be more rugged because people drop, bang, and otherwise mistreat these machines. Such ruggedization requires a more elaborate design and testing process and stronger, and therefore more expensive, parts. Maintenance contracts are more expensive due to the presumed higher repair rate. If a laptop has proprietary expansion slots, the peripherals that plug into

those slots will be more expensive due to reduced or eliminated competition. Power-saving features, which are not needed on a desktop, add cost. The need for mobility drives designers to favor power conservation and heat dissipation, leading to the selection of slower CPUs and I/O devices. Thus, laptops are often slower than desktops from the same generation.

However, a laptop's additional cost is often mitigated by other factors: Fixed-configuration devices have a lower manufacturing cost. Laptops are easier to support because people can bring their laptop to a helpdesk rather than requiring an IT technician to visit their desk. Laptops also work well for companies that want to promote dynamic seating arrangements and have pool workspaces, rather than fixed desks—in such an environment, each workspace just has a docking station and people bring their laptops. Lastly, the additional cost is justified by the fact that being able to bring a laptop somewhere has inherent value, particularly for people who do a lot of travel or oncall support.

Most companies that provide their employees with physical hardware usually offer both options. Some allow managers to choose for their employees; others have a stricter allocation based on the person's job requirements.

We predict desktops will be increasingly replaced by a VDI and thin clients. Because of the inherent value of laptop portability, we expect laptops to remain an option for a longer time than desktops. Eventually, however, most laptop users will transition to tablets, pads, and other mobile devices.

6.1.2 Vendor Selection

When purchasing hardware, there are many vendors to choose from. Organizations may standardize on one vendor, select a fixed set of vendors, or have, essentially, a policy of supporting all vendors. Minimize the number of vendors used to reduce complexity and support cost.

Standardizing on one vendor makes support easier and makes it easier to qualify for bulk purchase discounts. Having multiple vendors permits price competition but incurs a bigger support cost. However, the additional support cost may be offset by playing one vendor off another to get the best price. We have received much better pricing just by letting the vendor know we are receiving quotes from another vendor.

Organizations that support all vendors often do so as a result of not actually having a purchasing policy, having such a policy but not enforcing it, or purchasing each machine as an individual exercise in getting the best initial purchase price rather than having a disciplined, sustainable strategy that

manages the total cost of ownership. No matter how an organization got there, this is a red flag that the IT organization lacks planning, or discipline, or political sway. The result is haphazard and uneven support for everyone, with occasional excellent support for particular vendors due to unpredictable circumstances such as the fortuitous hiring of an SA with experience with that vendor's products.

Before starting a vendor selection process, document your criteria, evaluation, and rating methods. As you are evaluating each vendor, document how each vendor performs against each criterion, and how that rating was reached. Doing so helps ensure that you stay focused on your organization's needs, rather than being distracted by a good sales pitch.

6.1.3 Product Line Selection

Most vendors have multiple product lines. Generally you'll find those that emphasize lowest initial cost or purchase price, those that emphasize lowest total cost of ownership (TCO), and those that emphasize performance.

Depending on the size of the company, and the needs of the people within the company, one or more of the product lines may be applicable. For example, an engineering company may need to pay a premium for performance workstations for its engineers but focus on TCO for the rest of the company. For most companies, the product line that they should be looking at is the one that emphasizes lowest TCO.

Lowest Initial Cost

The purchase price is the cost the consumer pays to acquire the product initially and does not include the cost of operating the device. The product line that emphasizes the lowest initial cost or purchase price has a low profit margin on the initial sale because the vendor balances that with high-profit add-ons and maintenance contracts.

The lowest initial purchase price is achieved by sacrificing the features that would make the machine less expensive in the long term or would make managing many machines less expensive. For example, a fixed configuration makes the initial purchase price lower but future expansion may require replacing the entire machine, which is considerably more expensive than being able to simply add another hard drive.

This product line also frequently changes part selection. Machines manufactured on Monday may have a different video chip than machines made on Tuesday simply because one supplier changed its price. This lets the

manufacturer adjust to price fluctuations as fast as possible, lowering the sale price. Customers won't notice the part change if they own only one machine. However, anyone maintaining a fleet of machines will find it difficult and costly to support so many different video systems, for example.

Such systems are often more difficult or impossible to repair. To save money, replacement parts are stocked for only a year or two. Shady vendors may not stock any at all, saving even more money.

These product lines are often called "consumer" or "small business" because the downsides of such tradeoffs generally won't be felt by sites with few machines.

Total Cost of Ownership

Total cost of ownership refers to all costs related to the machine for its lifetime. This would include the initial purchase price, plus the estimated cost of service contracts, maintenance, and repair for the projected life of the device, usually three years.

A product line that emphasizes TCO does so by focusing on issues that would save money when an organization is maintaining a large fleet of machines. These product lines are often called the "enterprise line" or "business line."

A business with a large fleet can lower its TCO by minimizing the variations in hardware. This reduces training costs for the IT department, reduces the types of spare parts that must be maintained, and reduces the number of OS and driver combinations that must be tested. It is much less expensive to roll out an OS patch if it must be tested with one or two hardware variations rather than dozens. The enterprise line often has guarantees about how long a specific model will be available, with very little "parts fluctuation" during that time. Some vendors can provide a six-month warning before the introduction of new parts or the end of life of old parts.

Enterprise line machines are designed assuming that in-house IT departments may need to rapidly repair machines and return them to service. Disassembling a laptop to replace a damaged LCD screen can be a lengthy process, made more difficult by cramped designs that result from cutting corners. Taking the extra step to make a laptop easier to repair is costly. Adding mechanisms to make hard drives easier to replace increases the initial purchase price but reduces repair cost. This is less important for consumer machines that get repaired at a local computer repair shop that charges hourly rates; in fact, the repair shop benefits from lengthy disassembly/assembly processes.

Enterprise line machines tend to support older hardware longer. A consumer line laptop might have one “video out” port, showcasing the latest video connector technology. That is insufficient for business users who need to contend with video conference systems that have VGA inputs, and probably will for decades. The enterprise line will have the added expense of including both video ports, both the old and new serial ports, the old and new everything.

Performance

The product line that emphasizes **performance** is often called the “engineering” or “power” line. It includes features required by engineering applications such as computer-aided design (CAD) and computer-aided engineering (CAE), which require high-end graphics, vector processing, and heavy-duty computation.

These models offer the fastest, newest CPUs, are expandable to large amounts of RAM, and have more storage options. Vendors charge a premium for the latest CPUs and technologies because customers can justify the additional cost. If engineers can create finer meshes that give more accurate results in their fluid dynamics and structural simulations, the result is a product that is more efficient and less susceptible to fatigue failures. As a result, the products gain a better reputation and increased market share worth potentially millions of dollars. An extra thousand dollars in workstation cost is tiny by comparison.

6.2 Virtual Desktop Infrastructure

Another workstation architecture option is virtual desktop infrastructure (VDI). The point of VDI is to turn workstations into a centrally managed network service to reduce administrative and hardware costs.

With VDI, the user’s applications run on a virtual machine (VM) in a VM server cluster elsewhere on the network. The user interacts with the VM through a “thin client.” The thin client looks like a very tiny desktop PC. It has a main unit that a monitor, keyboard, and mouse plug into. The main unit has no hard disk. It runs a specialized operating system that simply connects to the virtual machine and, essentially, becomes that VM’s monitor, keyboard, and mouse.

The benefits of VDIs are that they offer savings due to reduced hardware costs, along with increased ease of management. The challenge in deploying VDIs is dealing with users’ needs to access nonstandard applications. There are also some variations on the thin-client VDI model, which will be described later in this section.

In highly regulated environments that need to adhere to standards such as the

Health Insurance Portability and Accountability Act (HIPAA), VDIs present an opportunity for finer-grained control over access to information. They also prevent the placing of sensitive information on unencrypted removable media by making sure that all data interactions are done on the server side. This is another advantage of VDIs beyond cost and ease of management.

6.2.1 Reduced Costs

The thin client hardware is inexpensive since it is designed for the single purpose of speaking a remote access protocol such as Windows Terminal Services, VMWare Horizon, or Citrix. Any hardware not required to do that is eliminated. Further savings come from the fact that the monitors, keyboards, and mice they connect to are standard commodity parts, the same ones used on full PCs.

The VDI system is easier to manage because the thin clients are generic. Anyone can log into one and, based on who the user is, the VM is allocated for that person. VDI systems are often used by cost-conscious users, especially where mass deployment means the savings are multiplied many times over. The thin clients themselves cost much less than a full workstation and generally do not need to be upgraded as often.

6.2.2 Ease of Maintenance

The VM server infrastructure can be upgraded more cost-effectively because it does not require the physical labor of visiting each thin client. One can upgrade or enlarge the VM server infrastructure and all the thin clients benefit.

For example, suppose initially each VM is allocated one virtual CPU. A year later, new applications arrive that would benefit from multiple CPUs. Additional CPUs can be added to the VM infrastructure and allocated among the VMs. Now the new applications are running faster without buying new thin clients. Such hardware upgrades also benefit from economies of scale.

Another example is that if 100 PCs needed an additional 1 TB of storage, that would require 100 individual hard drives. The labor to visit and install each hard drive would cost more than the hard drives themselves. If 100 VMs required the same storage, that could be achieved by allocating one hundred 1 TB virtual disks from a large SAN built with more cost-effective large disks. This could be done with a PowerShell script.

6.2.3 Persistent or Nonpersistent?

The first thing that an SA needs to consider when deploying VDI systems is whether the VDIs will be persistent. This decision has a significant impact on both the user experience and the cost and manageability of the service as a whole.

With **nonpersistent VDIs**, the virtual desktop is created from scratch using a standard golden image each time it is needed. Nonpersistent VDIs do not permit customization.

With **persistent VDIs**, there is a one-to-one mapping between users and virtual desktops, and each user's desktop is created from a separate disk image. The user's settings are saved at the end of each session, and appear each time at login, just like with a physical workstation.

In environments where users have very standard needs, nonpersistent VDIs will be the best choice. In environments where there are power users, or many users need nonstandard applications, persistent VDIs will be the better choice. Some environments may use a mixture of both approaches.

Nonpersistent VDIs

With nonpersistent VDIs, the VMs are also generic. Each time they boot, they start with a fresh copy of the base operating system image, often called the golden image. All user files reside on remote storage. If the user reboots, the VM is recreated from the newest golden image. There is no patching or software upgrade management required for the individual machines. Software upgrades are done to the golden image, and then the VMs are rebooted to receive the fresh update. If the user accidentally corrupts the OS, or a virus makes its way past the anti-malware software, a reboot returns the machine to a pristine state.

Nonpersistent VDI systems are often used in call centers or in any situation where the needs of the users are very standardized. They are also useful for information kiosks and other situations where local support is minimal and it is convenient to be able to reboot the machine to bring it back to its original state. The advantages of nonpersistent VDIs are as follows:

- It is easy for administrators to patch and update the image—it needs to be done only once, and it applies to every machine at the next reboot.
- It minimizes storage and backup requirements for the OS image.
- It simplifies deploying company-wide applications to all end users.
- It improves security since users can't alter desktop settings or install their own applications. Also, rebooting a compromised desktop brings it back to a clean state.

The disadvantages of nonpersistent VDIs are as follows:

- Not all software supports use in a nonpersistent VDI.
- It reduces personalization. With nonpersistent VDI, users cannot easily personalize their desktop.
- It reduces application flexibility. End users cannot simply decide that they need a new application and request it through a standard ordering process. Applications are either installed on the golden image or not. Thus the application is available either for all users or for none. The golden image should not be cluttered with lots of applications that most people do not need.
- It is a difficult transition for users. End users are accustomed to being able to personalize their machines and order additional software. The lack of these features reduces user acceptance of a VDI solution.
- It increases complexity. Custom applications that are required by some users but not others usually require application virtualization or user environment virtualization, which adds complexity. In addition, not all applications lend themselves to being virtualized.
- It requires a new way of thinking for support personnel. Supporting nonpersistent VDIs requires retraining of the end-user support staff, as it is an inherently different environment from what they are used to.

Persistent VDIs

Persistent VDIs allow for more personalization. However, they also require more storage and backups than nonpersistent VDIs, because each desktop runs from a separate disk image that needs to be stored and backed up.

With a one-to-one persistent VDI, the user's settings are saved and appear each time at login. These types of desktops allow for more personalization, but they require more storage and backup than nonpersistent desktops. The advantages of persistent VDIs are as follows:

- They ease customization. End users can personalize persistent desktops, and install custom software that is not needed by all users and perhaps has expensive licensing.
- They provide an easier transition for users. The customization aspect of persistent desktops tends to help users embrace VDIs more easily.
- Support is similar to desktops. A persistent VDI is basically the same as a physical desktop from an end-user support perspective, which makes it

easier for existing support staff to transition to supporting VDI users.

- The one-to-one user to disk image mapping requires less desktop re-engineering than switching to a single golden image with additional application virtualization.

The disadvantages of persistent VDIs are as follows:

- Larger storage and backup capacities are required. All the individual, customized disk images require more storage capacity than a single golden image for a nonpersistent VDI.
- There is more support overhead due to image management. Similar to physical desktops, all the diverse OS images need to be individually upgraded and patched. Therefore persistent VDIs do not benefit from the same reduction in support overhead that nonpersistent VDIs have.
- There are fewer security benefits. As with physical devices, a compromised persistent VDI remains compromised across reboots, because the (compromised) image persists.

Variations

In addition to providing a complete workstation experience, VDIs can be used to provide individual applications. While the application looks as if it is installed and running on the user's VM, the application is actually running on another VM, with the windows it creates being displayed alongside the user's other windows. Generally the user won't be able to see any difference. This is one way that custom applications are provided to users of nonpersistent VDIs.

One variation of a VDI is called a **VDI thick client**. This is a thin client emulator that runs on a normal PC. Imagine a user already has a PC, but needs to run a single application that is available from the VDI application service. The user runs a thick client on his or her workstation to access the application, eliminating the need to go through any installation procedure, use any local storage, and so on. In some systems this client runs in the user's browser, so it requires no additional software, albeit with some functionality being impaired.

Thick clients give users access to applications that are otherwise incompatible with their workstations. For example, one can access applications that are incompatible with the workstation's OS or that conflict with other installed applications.

There are also security benefits. For example, it may be difficult to get approval to install applications on machines. However, if the thick client app has been approved, the user can then access any VDI-provided applications safely.

The VDI thick client can generally be trusted because it runs in a protected container, generally the Java run-time environment, which has been vetted and approved by the organization's security group.

Case Study: VDIs for Untrusted Software

While at a major consulting firm, an SA was deployed to work at a client's location. The client provided the workstation to be used, but restricted which software could be installed. She couldn't install the timesheet software required to bill the company for her time worked. However, the client workstations supported a VDI thick client. She could securely access the VDI-provided timesheet software without violating the client's software restriction policy. The client was willing to approve this use case because it trusted the security of the VDI thick client.

6.3 Bring Your Own Device

Bring your own device (BYOD) is a service model where users supply their own hardware device rather than use one provided by their employer. For example, people may already own a mobile device such as an Android phone or Apple iPad and want to run corporate applications on them.

This is a radical departure from the way IT departments have operated in the past. Typically an IT department evaluates products and tells its users what they will use. Adopting a strategy where users provide equipment and IT must figure out how to support it is quite an adjustment.

6.3.1 Strategies

In a **BYOD-only strategy**, the IT organization supplies no hardware and requires users to supply their own device. BYOD-only users are usually a specific subset of users, such as people involved in a particular program or short-term project. For example, college students contracted for a six-week campaign may only need access to email and a task-specific application, and typically already have a mobile device. BYOD-only enables them to be productive without adding significant burden to the IT organization.

In a **BYOD mixed model strategy**, the IT organization supplies workstations but also permits BYOD. This is often a productivity enhancement, or a way to provide mobile access. Companies may already support corporate-owned mobile devices for certain people (typically salespeople and executives) and find that support for everyone else's personal devices carries only a small additional cost.

Also, many companies find that if they do not support some kind of BYOD, users will find makeshift solutions that give them the mobile access they desire. Sadly, these workarounds are never as secure as a properly supported BYOD service. Thus, providing BYOD services improves security.

A **BYOD-lite strategy** enables a small number of specific applications to be accessed, possibly only from registered devices made by approved vendors. For example, an organization might provide access to a core application such as email, calendar, and a corporate phone book, but nothing else.

BYOD is inevitable, and companies need to explicitly define a strategy for supporting it. For most companies, the BYOD mixed model is the right approach. Employees use company resources, such as a VDI thin client or physical workstation, when on site, and can access most applications from their mobile devices at other times. For some industries, such as health care and banking, where data privacy and regulation are significant themes, the BYOD-lite strategy may be more appropriate, and the end users may be more understanding about the need for more limited access.

6.3.2 Pros and Cons

BYOD benefits users because they are typically comfortable with their current device and find it bothersome to have to carry a second device for work.

BYOD benefits the company by improving productivity for its employees without having to accept full responsibility for hardware support. Savings come from not supplying hardware, but more so from not having to support it. Support costs often outweigh the hardware purchase price, especially since mobile devices are often so inexpensive. BYOD also benefits the company because when employees have easy access to their work when mobile, they will be more available and will work more without even realizing it.

However, the support cost increases due to the added complexity inherent in supporting a wide variety of models and devices. Supporting multiple brands and models is difficult and may require expanding the number of helpdesk support engineers needed. It can be less expensive to purchase a supported device for a person than to have a helpdesk support engineer spend a week figuring out how to make one odd-ball device work.

This complexity can be mitigated by adopting a number of classes of support. First-class models are fully supported. If an application doesn't work on them, it is treated as a bug—the same as if the application didn't work on the company-issued workstation. Second-class models work, but have documented known

issues or certain applications are exempt from support. Third-class devices receive support that is “best effort” or “self-support.” The helpdesk personnel are limited in how much time they will spend trying to fix an issue.

6.3.3 Security

BYOD raises many security issues. Is the device itself secured from malware, keyloggers, and password sniffers? What if the device is stolen? Mobile device management (MDM) software permits an organization to secure and control the BYOD devices in the organization. Typical features include the ability to deny access unless approved anti-malware software is in use, the ability to perform end-to-end encryption on all communication, and the ability to remotely wipe the device (or just the corporate-owned data) in the event of theft or if the employee leaves the company. MDM software can also reject access if various security settings are not enabled. For example, it may require an unlock PIN or a recent security certificate.

MDM systems often include some kind of VDI thick client, which permits application access as described earlier.

BYOD can be a source of data leakage. Not only is the device physically out of control of the company, but the software onboard is, too. For this reason, it is important to assess data leakage risks. Define and apply a process for risk identification, assessment, and management. For example, define which types of data can be on mobile devices, which can't, and which can be only on devices with an MDM.

6.3.4 Additional Costs

Mobile devices with cellular voice and data plans may incur large bills. Often companies reimburse employees for their cellular phone bill, with monthly limits. Requiring employees to fill out a reimbursement form and scan their mobile bill every month is tedious at best, and a waste of employee time at worst. It is usually better in the long run to simply pay a fixed amount via people's paychecks.

BYOD plans should also consider the impact on the network. Typically all mobile devices connect to the company over the Internet via a VPN. If the current VPN system is nearly overloaded, adding hundreds of new mobile devices will send it over the edge. More bandwidth and additional VPN hardware may be required.

6.3.5 Usability

Lastly, but most importantly, the end-user experience is everything. For a BYOD system to be accepted by the end users, it must be convenient. Usability gates acceptance more than any other factor. Historically, people tolerated corporate applications with badly planned user interfaces and ugly color schemes because they were told (and paid) to use them. IT departments didn't have to care as much. Consumer devices, by comparison, have a much bigger emphasis on style and design. For example, the designers of the iPhone devoted a lot of attention and effort to how the different parts work together as a whole, and strive to get right all the little details known as "fit and finish." This raises users' expectations for quality and usability. Unlike in the past, when IT departments told customers what they would be using, in a BYOD world employees have more power to walk away if they don't like what has been offered.

Mobile devices are the future of IT, and BYOD is a big part of that. As mobile devices get more powerful and better connected, they will subsume more and more workstation use cases. With thoughtful planning and disciplined security policies, BYOD can be a strategic part of how services are provided to end users.

MDM Usability Disaster

Many BYOD deployments have been ruined by clunky systems requiring users to enter passwords repeatedly. The failed projects leave a wake of ill feelings between employees and their IT department.

With one particular MDM, a person using Facebook who wishes to pause to check work email must log out of Facebook, log out of the "personal zone" of the device, log into the "work zone," and log into email. To return to Facebook, this process must be repeated in reverse. The only thing more shocking than this MDM's terrible workflow was the fact that companies purchased the product.

If anything is going to prevent the acceptance of BYOD, it is, sadly, MDM vendors themselves, many of which don't seem to understand that usability gates acceptance.

6.4 Summary

There are three major strategies for providing workstations to users.

The traditional strategy is to provide traditional laptops and/or desktops. The selection considerations for workstation hardware go beyond which vendor and

whether it is a desktop or laptop. Vendors usually have multiple product lines. The consumer or small business line emphasizes the initial purchase price. The enterprise or business line emphasizes improving the best TCO and management costs. The performance line emphasizes high-performance options.

A VDI strategy uses virtual machines as workstations. Users then access the virtual machines via thin clients (hardware that provides a display, keyboard, and mouse) or thick clients (software that runs on an existing workstation or mobile device).

A bring your own device (BYOD) strategy leverages people's personal mobile devices but gives them access to enterprise applications and resources. This is done by secure (SSL) access to web sites, VPNs, or VDI thick client software.

Whatever the strategy, the quality of the resulting user experience can be described in terms of the key issues described in the previous chapter: locality, reliability, productivity, user agency, and currentness.

Exercises

1. What are the three major strategies for providing workstations to users?
2. Laptops can go anywhere. Why do we buy desktops at all?
3. I sit at my desk all day, and so do all my co-workers. Why would I ever benefit from switching to a laptop?
4. How do vendors typically differentiate their product lines?
5. Some product lines focus on lowest initial cost, while others focus on total cost of ownership. Compare and contrast the two types of product lines.
6. List the different workstation vendors that are used in your environment. Do you consider this to be a lot of vendors? What would be the benefits and problems with increasing the number of vendors? Decreasing the number of vendors?
7. Identify three features that improve total cost of ownership. How would we know at purchase time if the extra cost is worth it? How would we later measure whether the goal was achieved?
8. Suppose you work for a company with 1,000 employees and your CEO wants to save money by purchasing workstations from the consumer line because the initial purchase price is so appealing. How would you talk your CEO out of this bad decision?
9. Standardizing on one vendor makes it easier to qualify for bulk purchase discounts. Why is this?

10. What is VDI and how does it work? What is the user's experience like?
11. What is BYOD and when is it appropriate to adopt this strategy?
12. Which strategy or strategies are used in your current organization? What would be the benefits to switching to the other strategies? Which of these benefits would apply to you or a co-worker you know?
13. Consider the issues identified in [Chapter 5](#), "[Workstation Architecture](#)" (locality, reliability, and so on). For each of the three strategies identified in this chapter, how do they help or hinder your ability to positively affect each issue?
14. In your organization, which strategy is used for your workstation? What are the pros and cons?
15. Given all the various options described in this chapter and [Chapter 5](#), "[Workstation Architecture](#)," design a desktop architecture that would be appropriate for a company of 20 employees.
16. Repeat Exercise 15, but assume the company has 1,000 employees.
17. Repeat Exercise 15, but assume the company has a call center with 1,000 employees sitting inside the office and 2,000 employees working from home.

Chapter 7. Workstation Software Life Cycle

This chapter is an overview of installing and maintaining the software that runs on our computers. The first piece of software a computer needs is its operating system (OS). The OS must be installed and then configured. Next, applications are installed. Both the OS and the application software will eventually need to be updated, or **patched**, as new releases add features, fix bugs, or plug newly discovered security holes. Configuration settings need to be managed over time due to changing requirements. Files and configurations may become corrupted and need to be repaired. Sometimes we need to wipe and reinstall the machine with the same or a new operating system. Lastly, when a machine is decommissioned, the OS and its files must be erased prior to disposal. All of these steps can be done manually or through automation. As you may have guessed, we favor automation.

Each new machine should start out in a “known good state.” Eventually the OS and configuration will deteriorate on their own. If each machine doesn’t start out the same, it is equivalent to giving the deterioration process a head start.

Active management of the OS and machine life cycle is an important part of the job of a system administrator. Some sites choose passive management: The OS that came with the machine is supported until the machine dies. This is a support nightmare, yet very common. In such environments making the smallest change across the fleet is a huge amount of work. Without automation these tasks will be delayed or skipped all together, leading to more problems.

7.1 Life of a Machine

A machine has a birth, life, and death. This life cycle is examined in Rémy Evard’s paper “An Analysis of Unix System Configuration” ([Evard 1997](#)), from which we borrow the diagram shown in [Figure 7.1](#).

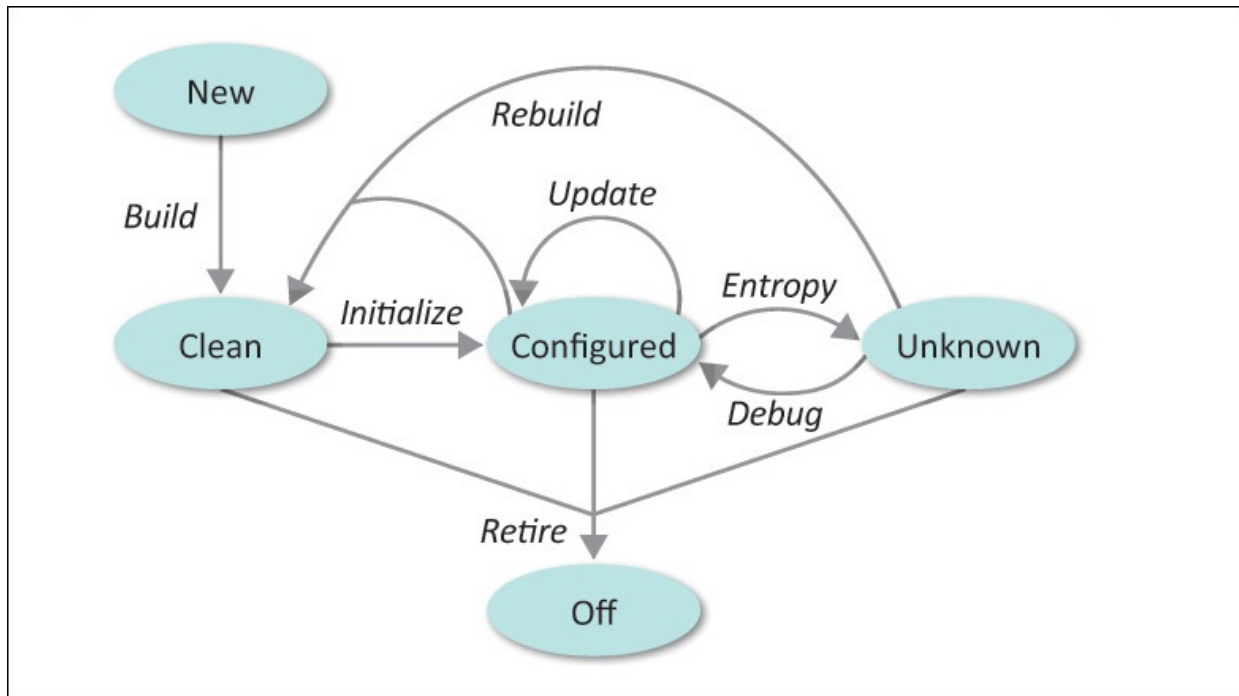


Figure 7.1: Evard's life cycle of a machine and its OS

The diagram depicts five states: new, clean, configured, unknown, and off:

- **New:** A completely new, unconfigured, machine
- **Clean:** A machine on which the OS has been installed but no localizations performed
- **Configured:** A correctly configured and operational environment
- **Unknown:** A computer that has been misconfigured or has become outdated
- **Off:** A machine that has been retired and powered off

There are many ways to get from one life-cycle state to another. At most sites, the machine *build* and *initialize* processes are usually one step; they result in the OS being loaded and brought into a usable state. *Entropy* is deterioration or changes we don't want. This leaves the computer in an unknown state, which is fixed by a *debug* process. *Updates* happen over time, often in the form of patches and security updates. Sometimes, it makes sense to wipe and reload a machine because it is time for a major OS upgrade, the system needs to be recreated for a new purpose, or severe entropy has plainly made it the only resort. The *rebuild* process happens, and the machine is wiped and reloaded to bring it back to the clean state and then initialized to bring it to the configured state.

These various processes are repeated as the months and years roll on. Finally,

the machine becomes obsolete and is *retired*. Alternatively, it may die a tragic death or, as the model describes, be put into the off state.

What can we learn from this diagram? First, it is important to acknowledge that the various states and transitions exist. Our team must have a process for each of the transitions. We plan for installation time, accept that things will break and require repair, and so on. We don't act as if each repair is a surprise; instead, we set up a repair process and maybe an entire repair department, if the volume warrants it. All these things require planning, staffing, and other resources.

Second, we notice that although there are many states, the computer is usable only in the configured state. We want to maximize the amount of time spent in that state. Most of the other processes deal with bringing the computer to the configured state or returning it to that state. Therefore, these setup and recovery processes should be fast, efficient, and automated.

To extend the time spent in the configured state, we must ensure that the OS degrades as slowly as possible. Design decisions of the OS vendor have the biggest impact here. Some OSs create a subdirectory for each application and install all related files there, thus consolidating the installation's impact on entropy. Other OSs scatter the application's files all over the place: main executable in one place, libraries in another, data files elsewhere, and so on. Doing this makes it difficult to discern which files are part of which package. Microsoft Windows is known for problems in this area. In contrast, because Unix provides strict permissions on directories, user-installed applications can't degrade the integrity of the OS.

As discussed in [Chapter 5](#), "[Workstation Architecture](#)," decisions made by the SA play a big part in this and can strengthen or weaken the integrity of the OS. Is there a well-defined place for third-party applications to be installed? Has the user been given `root` or Administrator access? Has the SA developed a way for users to do certain administrative tasks without having the supreme power of `root`? SAs must find a balance between giving users full access and restricting them. This balance affects the rate at which the OS will decay. A wise person once said, "To err is human; to really screw up requires the `root` password."

Manual installation is error prone. When mistakes are made during installation, the host will begin life with a head start into the decay cycle. If installation is completely automated, however, new workstations will be deployed correctly.

Reinstallation—the rebuild process—is similar to installation, except that one

may potentially have to carry forward old data and applications. As discussed in [Section 5.6](#), reinstallation is easier if no data is stored locally on the machine, or any data stored locally is a copy of data stored elsewhere. This topic will be covered in more detail in [Chapter 33](#), “[Server Upgrades](#).”

Finally, this model acknowledges that machines are eventually retired. Machines don’t last forever. Various tasks are associated with retiring a machine. As in the case of reinstallation, some data and applications must be carried forward to the replacement machine or stored on tape for future reference; otherwise, they will be lost in the sands of time.

To do a good job at OS management, we must have procedures in place to handle each of these states. These procedures may be ad hoc and improvised while a site is small, but a professional system administrator has formal processes that are embodied in automation.

7.2 OS Installation

The OS installation process erases any existing operating system and installs a new one, bringing the OS to Evard’s “clean” state.

Installation is best achieved through automation. A fully automated installation process assures that each machine is configured correctly. Doing it manually is prone to errors and, to be blunt, is pretty boring. You have better things to do than stand around answering prompts.

There is a lot to say about OS installation and many strongly held opinions on this subject. [Chapter 8](#), “[OS Installation Strategies](#),” goes into greater detail about OS installation and automation.

7.3 OS Configuration

After installation, many subsystems and components need to be configured. The specific settings may change over time, so we need a way to both make the initial configuration changes and update them.

There are many ways to create and distribute such configurations. This chapter discusses configuration management (CM) systems, group policy objects (GPOs), and some tricks one can do with DHCP. This section ends with a common technique that we recommend against.

7.3.1 Configuration Management Systems

As described in [Chapter 4](#), “[Infrastructure as Code](#),” CM systems are specialized programming languages that permit you to describe the details of what a

properly configured system should look like, called the **desired state** of the machine. The CM system reads this description and makes the changes required so that the machine conforms to the desired state. If the system already conforms, no changes are made. If only a few things are out of line, only the minimum changes required are made. Therefore you can use these systems to configure a newly installed machine, or run them periodically to correct entropy as it sets in, shepherding machines that drift into the *unknown* state back to *configured*. Later, if the configuration needs to change, updating the CM code results in the corresponding changes being made on the machine.

CM systems generally provide a class system similar to object-oriented languages. Thus, you might define one class that describes the base configuration for CentOS and another that describes Debian. Other classes might describe how to configure an Apache HTTP web server, an ISC DHCP server, or another application server. By specifying that a particular host runs CentOS and is an Apache HTTP web server, the CM system will do the right thing. Another host may be defined as a Debian host that is also a DHCP server.

Such systems permit subclasses based on other classes. For example, one might define a blog server as the web server class plus additional changes such as blogging software. This way code is reused to build more sophisticated systems.

The class that defines the configuration for a CentOS machine might include that all CentOS machines are configured to point to a particular NTP server for time synchronization. If we need them to point to a different NTP server, simply changing that class will roll out the change to all CentOS machines. This enables us to make mass changes with very little effort.

CM systems are the preferred method of distributing changes because of the fine degree of control they provide.

Giving Users Control Over Settings

Google developed a highly automated system for macOS installation, configuration, and updating. It enforces numerous security settings. Some of these settings are unpopular to a small minority of users. Google developed a system for users to loosen certain security settings on a per-machine basis. The system provides software “knobs” or switches that users can adjust. For example, due to many security concerns, Firewire interfaces are disabled by default, but there is a knob that will reenable them. Without this knob, people who wanted to attach Firewire

peripherals would find ways to subvert the security settings, possibly to the detriment to the overall security of the machine. By providing a knob, users get what they want, and the security team can manage the risk by tracking the exceptional machines. If a large number of people undo a particular security setting, this is useful information. The security team knows which users to interview to better understand their needs and identify ways to achieve the users' goals in a more secure way.

7.3.2 Microsoft Group Policy Objects

Microsoft Windows systems use GPOs for making changes or setting policies on machines and users. While not as sophisticated as full configuration management systems, GPOs are integrated into Windows in a way that makes them very easy to implement.

A GPO defines some permission or setting to a definition stored in Active-Directory. A GPO can apply to groups of machines or users. Suppose there is a group of users who should have a slightly more restricted environment. There is a GPO setting that disables changes to the taskbar and the Start menu; one might apply this setting to users in that group. One can also apply a GPO to particular machines. For example, there is a GPO setting that disables the shutdown menu item for non-administrative users. This GPO setting might be applied to any shared machine, as we do not want a user to accidentally shut down a machine while others are using it. GPOs can also define printer access policies, patch update schedules, and many other settings.

While GPOs are generally entered manually using the Microsoft Windows GUI-based administration tools, they can also be created and distributed via CM systems, thus achieving the best of both worlds.

7.3.3 DHCP Configuration

Many network settings can be controlled via DHCP. As described in [Section 5.4](#), this is often more useful for workstations than servers. For example, DHCP can be used to set which DNS server a machine uses. If you are decommissioning one DNS server and replacing it with another, configure the new DNS server's IP address in the DHCP server's configuration. The new configuration will be distributed to all new clients. Existing clients will receive the new information when they renew their lease. Depending on how often clients renew their leases, it may take hours, days, or weeks to reach all machines. Therefore both the old and new DNS servers would need to remain active until the last DHCP client has

renewed its lease and seen the change.

DHCP has many benefits. First, it can control machines to which you don't otherwise have administrative access. Most OSs default to using DHCP to control NIC configuration, DNS settings, and other network-related options. Therefore even a guest machine that you otherwise have no control over will abide by the settings you provide it from your DHCP server.

DHCP has downsides as well. It controls only a few predetermined settings, almost all related to networking. Not every machine on your network will use DHCP, so you'll have to implement any change twice: once for the DHCP hosts and once for whatever controls the others. The quality of DHCP client implementation varies, too. Workstations and servers usually have well-tested, full-featured implementations, while other devices have implementations that are buggy and minimalistic. Some embedded devices ignore changed settings when their lease is renewed, frustrating attempts to make global changes via DHCP. Many voice over IP (VoIP) phones have persnickety DHCP implementations that freak out if they are sent any optional settings that don't pertain to them.

However, the biggest problem with distributing changes via DHCP is the lack of positive feedback. It is difficult to determine which clients have received an update, which haven't, and which have simply gone away. You need to wait for the full length of the lease time to be sure that all dynamic clients have received the update. However, DHCP also allows for static leases, which have either no lease expiration date or a very long lease time. Machines with a static DHCP configuration often will receive the new settings only upon reboot.

7.3.4 Package Installation

Some sites distribute configuration changes via software packages. How this works is that software packages usually include a post-install script—a program that runs after installation to perform various cleanup tasks. A common technique is to create a package that installs no files but includes a post-install script. This script makes the desired change. The package is distributed to any machine that requires the desired change.

We do not recommend using this technique because if the change gets undone due to entropy, it will not be corrected because the package is already installed. There is no mechanism to check that the change has been undone and needs to be redone. Soon you are releasing new versions of packages just to force the change to happen again. It becomes a mess. To avoid this chaos, you should use any of the previously discussed techniques instead.

7.4 Updating the System Software and Applications

Wouldn't it be nice if an SA's job was finished once the OS and applications were loaded? Sadly, as time goes by, new bugs and security holes are discovered and new applications become available.

All these tasks require software updates. Someone has to take care of them, and that someone is you. Don't worry, though: You don't have to spend all your time doing updates. As with installation, updates can and should be automated.

Software-update systems should be general enough to be able to deploy new applications, to update existing applications, and to patch the OS. For example, Microsoft WSUS can distribute all three. The entire Linux operating system is a group of many individual packages. Linux applications use the same package format. A system for distributing and installing updated Linux packages can update both the operating system and applications.

The ability to roll out software packages via automated means is key to a well-run environment.

Case Study: Installing a New Print System

An SA was hired by a site that needed a new print system. The new system was specified, designed, and tested very quickly. However, the consultant spent weeks on the menial task of installing the new client software on each workstation, because the site had no automated method for rolling out software updates.

Later, the consultant was hired to install a similar system at another site. This site had an excellent—and documented—software-update system. Mass changes could be made easily. The client software was packaged and distributed quickly.

At the first site, the cost of building a new print system was mostly the labor required to deploy the software. At the second site, the cost was mostly just the expense to develop the new print service. The first site thought it was saving money by not implementing a method to automate software roll-outs. Instead, the organization spent large amounts of money every time new software needed to be deployed. This site didn't have the foresight to realize that in the future, it would have other software to deploy. The second site saved money in the long term by investing some money up front.

7.4.1 Updates Versus Installations

Automating software updates is similar to automating the initial installation but is also different in many important ways:

- **The host is in a usable state.** Updates are done to machines that are in good running condition, whereas the initial-load process has extra work to do, such as partitioning disks and deducing network parameters. In fact, initial loading must work on a host that is in a disabled state, such as with a completely blank hard drive.
- **The host is already in use.** Updates involve a machine that has been in use for a while; therefore, the customer assumes that it will be usable when the update is done. You can't mess up the machine! By contrast, when an initial OS load fails, you can wipe the disk and start from scratch.
- **No physical access is required.** Updates shouldn't require a physical visit, which is disruptive to customers; also, coordinating visits is expensive. Missed appointments, customers on vacation, and machines in locked offices all lead to the nightmare of rescheduling appointments. Physical visits can't be automated.
- **The host may not be in a "known state."** As a result, the automation must be more careful, because the OS may have decayed since its initial installation. During the initial load, the state of the machine is more controlled.
- **The host is in the native environment.** Update systems must be able to perform the job on the native network of the host. They cannot flood the network or disturb the other hosts on the network. An initial load process may be done in a particular location where special equipment may be available. For example, large sites commonly have a special **install room**, or **staging area** with a high-capacity network, where machines are prepared before delivery to the new owner's office.
- **The host may have "live" users.** Some updates can't be installed while a machine is in use. Microsoft's System Management Service solves this problem by installing packages after a user has entered his or her username and password to log in but before he or she gets access to the machine. The AutoPatch system used at Bell Labs sends email to a customer two days before and lets the customer postpone the update a few days by creating a file with a particular name in `/tmp`.
- **The host may be unavailable.** It is increasingly likely that a host may not always be on the network when the update system wants to push updates to

it. This is especially true for laptops and remote employees who access the network by VPN only occasionally. Either the update system must retry and hope to catch the machine when it is available, or the machine must poll the server for updates.

- **The host may have a dual-boot configuration.** In this age of dual-boot hosts, update systems that reach out to desktops must be careful to verify that they have reached the expected OS. A dual-boot PC with Windows on one partition and Linux on another may run for months in Linux, missing out on updates for the Windows partition. Update systems for both the Linux and Windows systems must be smart enough to handle this situation.

7.4.2 Update Methods

There are a number of strategies to distribute software updates. One can choose not to do updates at all, to do them manually, or to automate them.

No Updates

The first strategy is *to simply never do updates*. In many ways this is the easiest option, but in other ways it is the most difficult. Initially it requires the least amount of work. However, over time the lack of updates will result in security issues and other problems that create more work than was initially saved. (Some executive managers feel that delaying spending is better than saving money, so the vastly larger expense of dealing with security problems is rationalized. Such rationalizations will eventually be regretted.)

What would make this strategy work is if all new machines were created fully patched and up-to-date, and then somehow we create a situation where we have only new machines. This sounds impossible, especially considering that new OS and application updates arrive frequently. However, there are two situations where we do exactly that.

As discussed in [Section 6.2](#), non-persistent VDI systems provide workstation environments through virtual machines that are wiped and reloaded from scratch every time the user logs out or reboots. If users log out every day, their machine is perpetually new, and updates appear on their new machine automatically.

The other situation where this strategy is viable is in the server world. Large services are actually made up of many smaller subservices, called micro-services, which run in different virtual machines. Each micro-service can be upgraded individually. Rather than upgrading the software in a virtual machine,

any new software release results in a new virtual machine image being created. The old VM is shut down and the new VM image is spun up in its place. Because these images are not upgraded, they are often referred to as immutable clones, though a more accurate description would be prefabricated or disposable infrastructure.

This strategy reduces duplicate effort. When a new package must be deployed, it is simply updated in one place: the OS installation or VM image creation automation. There is no OS update mechanism that must be updated as well, so there is less work to do.

Another benefit is that this strategy reduces the potential variations within the environment. Compare two machines: The first machine was installed today with today's set of packages. The second machine was installed last week with last week's set of packages but was upgraded today to include all the same version packages as the first machine. In theory these two machines are the same, but in practice they are not. There is the potential for minor but significant differences that can cause bugs and other problems.

The second machine is a new variation of machine that adds to the testing burden and support complexity of all the systems that surround it. In some environments there are dozens of platforms for each release. Using prefabricated infrastructure reduces the amount of testing labor by half. For organizations that previously did not test both variations, this results in testing that is of consistently higher quality.

User-Initiated Updates

User-initiated updates mean the user of the machine is required to update the system. Most users simply never think to update their software unless they are compelled to do so. In other words, systems will not be updated until a user feels the need for a new feature. Security updates and any update that is invisible to the user will not be installed.

While this is a very common strategy, it is not recommended. Customers should not be responsible for keeping their machines updated. It isn't their job. That is the job and expertise of system administrators. It may feel like less work on your part, but the security and other problems that result will create more work for you in the long run.

Automated Updates with User Approval

It's almost always best to automate updates. On servers, these updates need to be coordinated so that they do not cause service interruptions.

In the case of replicated services, the automation can patch and reboot at any time, though sometimes a special process is required. For example, if a service is replicated on two machines, the service may have to be disabled on one machine while it is being upgraded and then reenabled. The complexity of what needs to be done ranges from nothing at all to highly sophisticated orchestrations that move state and processing around.

In the case of unreplicated services, we can automate special cases. For example, updates that do not require a reboot might be possible at any time. Other updates may require the automation to wait until a particular service window of time where outages are permitted.

Software updates on workstations are an entirely different matter. If the update happens while customers are using their machines, they could lose their work.

User-prompted updates are an attempt to address these challenges. When updates are required, the user is asked to permit the updates to begin. This improves the likelihood that the updates will happen. However, users consider these prompts an annoyance and generally opt to postpone updates forever.

In response, many update management systems can mark updates as required by a certain date, with rare emergency updates having a shorter fuse. Users may be able to postpone the update for a few days, but then their false hope that they can postpone the update forever is dashed. Users probably dislike this most of all.

When using such a system, be generous in the duration people may postpone an update. They know best when an update will least disturb their work.

System administrators therefore have a responsibility to use such systems wisely. Following is some basic advice we've learned the hard way:

- Be generous in the amount of postponement permitted. If an update is needed, it is reasonable to expect customers to be able to do it within a week. If someone goes a week without a break from his or her computer, that person probably needs a break anyway.
- Test all updates. If you don't have time to test each update, at least delay the update for a few days from when it is originally published by the vendor to when the user is prompted to install it. Most updates that are discovered to be problematic are withdrawn by the vendor within five days. One important aspect of testing is to make sure that there is sufficient disk space available, especially in boot partitions.
- Emergency updates should be extremely rare, occurring perhaps once a year and then only with communication to all employees that the risk of

not deploying it justified the short fuse.

- Minimize the update surface area. Install only the absolute minimum software required on a machine. This reduces the probability that updates will be required. It is frustrating to be asked to reboot to update software that you don't use.

7.5 Rolling Out Changes . . . Carefully

An automated update system has the potential to cause massive damage. You *must* have a process around it to make sure that risk is managed. The process needs to be well defined and repeatable, and you should attempt to improve it after each use based on what was learned from the previous experience. You can avoid many disasters if you follow this system.

An automated patch system is similar to a clinical trial of an experimental anti-influenza drug. You wouldn't give an untested drug to thousands of people before you'd tested it on small groups of informed volunteers. Likewise, you shouldn't perform an automated patch roll-out to the whole company until you're sure that it won't do serious damage.

- Create a well-defined release candidate that will be distributed to all hosts. Nominate it for distribution. The nomination begins a buy-in phase to get it approved by all stakeholders. This practice prevents overly enthusiastic SAs from distributing trivial, non-business-critical software packages on a whim.
- Roll out the change to a small group first, then larger and larger groups until it is distributed to all. Each repetition with another group is called an **iteration**.
- Establish success criteria for an iteration. Make sure the success criteria are achieved for each group before the deployment to the next group begins. For example, if there are no failures, each succeeding group might be about 50 percent larger than the previous group. If there is a single failure, the group size returns to a single host and starts growing again.

The ramifications of a failed patch process are different from those of a failed OS load. A user probably won't even know whether an OS failed to load, because the host usually hasn't been delivered yet. However, a host that is being patched is usually at the person's desk. A patch that fails and leaves the machine in an unusable condition is much more visible and frustrating.

You can reduce the risk of a failed patch by using the **one, some, many technique**:

- **One:** First, patch one machine. This machine may belong to you, so there is incentive to get it right. If the patch fails, improve the process until it works for a single machine without fail.
- **Some:** Next, try the patch on a few other machines. If possible, you should test your automated patch process on all the other SAs' workstations before you inflict it on users—SAs are a little more understanding. Then test it on a few friendly customers outside the SA group.
- **Many:** As you test your system and gain confidence that it won't melt someone's hard drive, slowly, slowly move to larger and larger groups of risk-averse customers.

In very large environments we can apply this on a larger scale. Categorize customers by how risk averse they are. Upgrade them in groups, beginning with the least averse and ending with the most averse. This way the most risk-averse users don't have to suffer from any bugs found and fixed earlier in the process.

For example, suppose you have four customer groups, perhaps taking the form of four different departments within your company. Each has a different tolerance for pain. The engineering team prefers to stay on the cutting edge and accepts the potential risk, even craving the responsibility to find and report bugs before anyone else. The marketing department tolerates less risk and may not have the technical skills to effectively report problems. The executives have the least risk, plus we don't want them to see anything new until after all the bugs have been worked out. This makes us look our best, as perception of competence affects their budgeting decisions. The sales department says they want the newest features first but the truth is that they have no appreciation for the fact that this means they'll see the most bugs. The fact that they complain to the executives first, and file bugs second, means they've squandered any right to self-determination as far as where they rank for new services.

Thus, in this example the big changes are first rolled out to your own machine, then the remainder of the SA team, then engineering, then marketing, then sales, then the executives.

Rather than grouping by employee type, one may group by division, group by hallway, or even choose groups randomly. Some sites pick a random 1 percent to go first, then proceed with groups of 10 to 15 percent until a change is rolled out globally.

Don't Erase Your University

In May 2014, Emory University accidentally sent an update that wasn't

just an update, but an entirely new image. As a result, all Windows machines, including laptops, desktops, and servers, started erasing their disks and reinstalling Windows from scratch. As soon as the accident was discovered, the SCCM server was powered off. Sadly, by that time the SCCM server had wiped itself. Recovery took weeks.

This is an extreme example but a good one. Automation should roll out changes to small groups over many days. Good automation makes it possible to include safety checks and other precautions against unintended results. And, of course, this is an important reminder that backups, kept off-site, are a necessary insurance plan against the biggest IT nightmares.

7.6 Disposal

The last stage of life for a machine is disposal. Tasks related to disposal fall mainly in three categories: accounting, technical, and physical.

Accounting tasks deal with things the accountants of the company are concerned with: capital inventory and taxes. Technical tasks are work that the system administrators do, including securely erasing storage. Physical tasks deal with the actual disposal of the item. The tasks will be different at different companies, but here are areas to cover:

- Accounting:
 - Remove the machine from capital inventory.
 - Write off any calculated depreciation.
 - Remove the machine from any hardware maintenance contracts.
 - Deallocate or transfer any software licenses.
- Technical (decommissioning):
 - Move or decommission any remaining services.
 - Transfer any data to other hosts or long-term backup media such as tape.
 - If licenses are being deallocated or transferred, complete the technical work required to do so.
 - Remove the machine from monitoring systems.
 - Remove or overwrite any passwords or encryption certificates, especially in any onboard management subsystems (iLO, DRAC, IPMI).
 - Power off the machine, or perform the virtual machine equivalent.

- Remove the machine from the configuration management database (CMDB), DNS, and any other databases or configuration files.
- Technical (data security):
 - Physically disconnect the machine from the network, or perform the virtual machine equivalent.
 - Reset any onboard management systems to the factory defaults.
 - Securely erase all disk, SSD, or other storage.
- Physical:
 - Disconnect any remaining cables.
 - Remove the machine from the rack.
 - Collect cables and other parts for the repair depot.
 - Sell, donate, or send the machine to an environmentally certified electronics recycling program.

7.6.1 Accounting

Accounting-related tasks involve eliminating any ongoing capital and operational expenses. These tasks are different in each country the company operates in. The only way to know all the tasks is to ask. Work with the appropriate department to develop a checklist of tasks, or a procedure to follow. The procedure may be simply that the machine's asset tag number is emailed to the accounting office, or it may be a more involved procedure involving updating databases, verifying contracts, and so on.

Sometimes decommissioned equipment goes back into the company stores to be reused or to be used for replacement parts. Make sure that you have a documented process for returning equipment to the company stores, including the accounting aspects.

7.6.2 Technical: Decommissioning

Decommissioning-related technical tasks need to be done by the system administration team. Again, these steps should be documented in a checklist so that they are done consistently. The checklist should note the order in which tasks must be done. For example, removing the machine from monitoring systems should probably be done earlier in the process so that spurious alerts aren't generated. The decommissioning tasks take the machine out of service. The data security tasks prevent accidental disclosure of proprietary, secure, or personal information.

Make sure that the CMDB is appropriately updated. If the hardware is being returned to the company stores for later use, it should be flagged as “in stock”; if it is being disposed of, it should be marked as “disposed.” Both cases should have separate workflows listing all the other tasks that need to be performed. In mature companies, the change in status in the CMDB will kick off a series of automated cleanup tasks.

7.6.3 Technical: Data Security

We can’t stress enough how important it is to properly erase disks and other storage before disposing of them. We constantly read reports of someone buying a used server on eBay only to discover that the machine is filled with credit card data, medical records, personal information, or secret government documents. If you think “nobody could possibly want the data on this machine,” think again.

Erasing disks and other storage is critical and, sadly, more difficult than one would expect. The best way to erase a disk is to use a utility that overwrites every disk block multiple times, each time with a different pattern. It can take a long time to do this but it is worth it.

Simply deleting the files does not actually erase the data. This process generally marks allocation tables to indicate that those disk blocks are available for reuse. The disk blocks still contain the data and can be pieced back together like a big puzzle. Anyone who has ever used an undelete program to recover an accidentally deleted file is glad it works this way. Even formatting a disk does not erase the data itself. To save time it overwrites the allocation table, again leaving the data blocks unmodified.

One would think that doing something as invasive as writing zeros to every block of the disk would erase the data. However, one can still recover data overwritten this way given the right laboratory equipment. Because zeros are a very consistent pattern, one can still read the echoes of the data that had been previously stored. Therefore disk erase systems generally make several passes over the disk, writing random data each time.

Derek’s Boot and Nuke (DBAN) is one such utility. It boots a mini operating system that runs out of a RAM disk. Because it does not depend on the underlying storage, it can wipe all storage. It can even be configured to not require a keyboard and monitor. After a series of beeps, the erasing begins.

Erasing the disks this way is insufficient for high-security situations. Disks have onboard intelligence. Often they set aside allocations of space for special purposes. These blocks are not accessible by normal means, yet they are large

enough to hold the remnants of your old data, often megabytes worth. Because of this, the gold standard for secure disk disposal is to crush the disk, then grind it into a powder.

7.6.4 Physical

A physical decommission involves disconnecting all cables, removing the device from the rack, and then removing all the cables. While this may sound simple, companies with mature SA processes will track and schedule this work as a high-risk activity. Removing devices and cables can result in unexpected equipment outages. Pulling out a cable that is in a big bundle of cables typically involves some amount of pulling, which can result in other cables being inadvertently pulled loose from equipment that is not being decommissioned.

Physical Decommission Problem

At one company, technical decommission of a network appliance had been completed, and the physical decommission was scheduled for a weekend. During the physical decommission, alerts for a different network appliance were received.

The datacenter technician had removed the wrong appliance, because when the newer appliance was installed, it had been erroneously labeled with the name of the other appliance. In addition, the inventory system had not been updated with rack locations when these devices were installed. As a consequence, the technician needed to rely on the labels. Unfortunately, he had failed to also check the asset tag (which was correct).

Because of the redundancy built into the service, there was no actual customer-visible outage. The labeling error was corrected, and everyone was reminded of the importance of having accurate labels, an accurate inventory system, and a process for double-checking all available information before performing a physical decommission.

Sometimes decommissioned equipment goes back into the company stores to be reused or to be used for replacement parts. However, eventually it will need to be disposed of. It can be sold, donated, or given to an electronics recycling company.

Selling the item can be labor intensive, especially when one considers the process may involve phone calls and visits from potential buyers, monitoring

eBay auctions, packing and shipping the device, and other logistics. The process is fun the first time, but soon you'll find it a distraction from more important technical tasks. It is a better use of your time to sell the item to a junk dealer that specializes in electronics, or to arrange for clerical support employees in your organization to coordinate it. If you set them up with the right information about the equipment and asking price, along with a process, they should be able to handle 90 percent of the tasks involved, coming to you only to handle exceptions. Such employees don't need to know what something does to be able to sell it.

Donating items or permitting employees to bring them home can be a good deed, but can create problems if they expect they can come to you for support. Clearly communicate up front that the items are delivered "as-is" with no support now or in the future.

Electronic disposal companies may accept the items for free or for a fee. It is your responsibility as a system administrator to make sure that the company follows all environmental laws regarding electronics disposal and is audited periodically. Electronics contain hazardous materials that are toxic to our environment.

7.7 Summary

This chapter reviewed the processes involved in maintaining the OSs of desktop computers. Desktops, unlike servers, are usually deployed in large quantities, each with basically the same configuration.

All computers have a life cycle that begins with the OS being loaded and ends when the machine is powered off for the last time. During that interval, the software on the system degrades as a result of entropy, is upgraded, and is reloaded from scratch as the cycle begins again. Ideally, all hosts of a particular type begin with the same configuration and should be upgraded in concert, though slowly to detect problems before they affect the entire fleet.

Some phases of the life cycle are more useful to customers than others. We seek to increase the time spent in the more useful phases and shorten the time spent in the less useful phases.

Three processes create the basis for everything else covered in this chapter: The initial loading of the OS should be automated, software updates should be automated, and network configuration should be centrally administered via a system such as DHCP. These three objectives are critical to economical management. Doing these basics right makes everything that follows run

smoothly.

There are many strategies for updating the OS of a workstation. We can simply not do updates, and either suffer the consequences or wipe and reload machines frequently with pre-patched software. We can expect users to do the updates, which is irresponsible for professional system administrators. We can automate the updates, with or without some kind of lever that permits users to postpone an update.

When rolling out updates, the “one, some, many” technique lets us defend against unexpected problems. We upgrade a few machines at a time, looking for issues. Eventually we upgrade larger and larger groups.

Disposing of a machine is complex. Technical issues involve copying any data off the machine to a safe location, deallocating licenses, and making sure nothing depends on it. For security reasons, the machine’s storage must be erased. Finally, the machine must be disposed of in an environmentally sound manner.

Exercises

1. What are the five states of the life cycle of a machine’s OS?
2. Which states enable the user to be productive? How can we maximize the amount of time in those states?
3. What is a configuration management (CM) system?
4. How are OS updates different from OS installations?
5. What are the strategies for performing software updates?
6. What are the pros and cons of permitting users to delay or postpone software updates?
7. In your organization, which strategies are used for performing software updates?
8. Sometimes a software update breaks existing software or causes some other problem. How can you guard against this in your organization?
9. What are the most important steps in disposing of a machine?
10. When it is time to dispose of an old machine, why can’t we just toss it into the garbage bin behind the office?
11. How does your organization manage the disposal of hardware?
12. An anecdote in [Section 7.4](#) describes a site that repeatedly spent money deploying software manually rather than investing once in deployment

automation. Examine your own site or a site you recently visited, and list at least three instances in which similar investments have not been made. For each, list why the investment hasn't been made. What do your answers tell you?

- 13.** Purchase a used machine or hard drive from eBay or Craigslist and see what personal data you find on it.

Chapter 8. OS Installation Strategies

This chapter is about installing the operating system of a machine and preparing it for use. There are many strategies for installing an OS, from fully manually to fully automated. How well we do this task influences everything else about the machine's life. It also affects all other aspects of our IT department. It is a fundamental building block of our environment.

Do a good job of OS installation, and users are ready and able to do their jobs. They require less support from us, which saves us time. Do a bad job, and the user is unproductive. The support required becomes a larger and larger burden.

We have visited many IT departments, and the ones that were a disaster (the polite term is “low-performing”) invariably had no OS installation strategy. Some machines were manually loaded and configured, others retained the vendor-provided OS, and in others nobody was sure how the OS had been installed at all.

The lack of automation meant that setting up each new machine took days of SA time plus all the time consumed by helpdesk requests that would have been unnecessary if the machine had been configured properly from the start. In fact, most of these organizations didn't even have a definition of what constituted being properly configured.

Every high-performing IT department we've seen has OS installation and configuration highly automated. This saves time because it is faster, but also because once the automated process starts the SA can walk away and do something else. If customers can initiate the automation, it disappears as a task for system administrators entirely.

The low-performing IT departments all complained of being overworked. They were drowning in urgent requests that prevented them from taking on big projects. Our recommendation always is to get control over the OS installation process. It will free up time for everything else.

Such automation doesn't create itself. If it doesn't exist in your IT organization, creating it may be the most significant thing you ever do.

8.1 Consistency Is More Important Than Perfection

The measure of any OS installation strategy is whether the resulting machines are configured consistently and correctly.

If OS installation is done incorrectly our customers will suffer. However, suffering also comes from inconsistency. If a customer has multiple machines, each unexpected difference is an unpleasant surprise. Each surprise looks sloppy and makes your IT team look bad. Consistency makes your team look much more professional.

The Tale of Two Coffee Shops

Imagine two coffee shops across the street from each other.

Mark's House of Consistently Mediocre Coffee serves coffee that is okay but nothing special. Still, customers know they can rely on getting coffee that tastes exactly the same every day.

At Ian's Inconsistent Coffee Shop, each cup of coffee might taste great or might taste terrible. One cup might be superior in every way to Mark's coffee, but the next cup you'll take a sip and spit it out. Gross.

It is unsurprising that customers gravitate toward Mark's shop. Even though people may potentially get better coffee at Ian's shop, they go to Mark's and avoid negative surprises.

Restaurants have understood this principle for a long time. A burger joint may not sell the best coffee, but it will have more success if it sells coffee that has the same consistent flavor every time. It might not be the best coffee, but coffee snobs probably don't drink coffee there anyway. The people who do buy coffee there will get a decent cup of coffee and, in fact, the place may develop a following of people who happen to like that particular flavor. Customers will adjust the coffee by adding sugar and milk, and it is comforting to know the exact same amount of sugar and milk will be needed every time.

People would rather receive machines that are consistently configured than perfectly configured. Perfection is impossible to achieve, especially considering that many decisions are simply a matter of personal taste. Consistency is achievable, however, and is a prerequisite for incremental improvement over time.

Customers may not like a particular setting but if they find all new machines have that setting, they can adjust it to their liking. If it is sometimes set one way and at other times set the other way, it will be a disconcerting surprise each time.

Inconsistency also creates inefficiencies for your team. It is more work for you

to support customers when machines are inconsistently installed and configured. Support calls are hampered by confusion. Support, therefore, takes longer and becomes more complex and expensive to provide.

A small misconfiguration can cause a big problem. An application may refuse to start or act oddly. If you are lucky, the problem will be found quickly so it can be fixed as an installation issue. If it isn't noticed for a long time, debugging will be more difficult since it will take longer to determine the source of the problem.

Manual OS and application installation will always result in some kind of inconsistency. No amount of SA training will help. People will forget things, misremember steps, or change steps intentionally to match their personal biases.

Inconsistent Windows Installation at Bell Labs

In the mid-1990s Tom worked in a department at Bell Labs that did its own computer support. Microsoft Windows was new and the installation process was manual, inefficient, and inconsistent. Tom found that PC system administrators spent about 25 percent of their time fixing problems that were a result of mistakes made at the time of installation. Customers usually weren't productive on new machines until they had spent several days, and often as much as a week, going back and forth with the helpdesk to resolve issues. This was frustrating to the SAs and customers alike! It also made a bad first impression. Every new employee's first encounter with an SA happened because his or her machine didn't work.

At the time there were two technicians who did all Windows installations. When Tom inquired, they were adamant that while the installation process was manual, it was exactly the same for all machines.

Tom walked the two technicians to a whiteboard and asked them to each list the steps in the installation process and any tweaks or settings that were done. As the two technicians discussed the process, the list grew and grew. Soon they saw many settings that were set differently depending on who was doing the installation, often based on the installer's personal preference. One installed a few extra utility applications that "everyone probably wants," while the other removed certain Windows utilities that "people probably shouldn't use." One even disabled certain features in MS Office that he didn't like and set some defaults to his own personal taste.

These two people who had just moments earlier claimed they followed the same process were now in a heated debate over many of the steps.

That's when the lightbulb went off in their heads and they realized they were delivering inconsistently configured machines to their users.

They agreed that the best way to resolve these conflicts would be to automate the process and *let the code be the record of their decisions*.

At the time Microsoft did not officially support automated installation of Windows, but provided hints and teases of how it might be done. Nonetheless, the team automated the installation process, filling in the gaps with their own inventive solutions, overcoming obstacles and uncooperative third-party drivers ([Fulmer & Levine 1998](#)).

Within a month the installation process was completely automated. Customers had a better experience with new machines. As a bonus, now they could wipe and reinstall the OS without requesting assistance. They were happier and more productive. The SAs were also a lot happier. The new process avoided all the mistakes that can happen during manual installation. As a result, the SAs had an easier time supporting the users and spent less time debugging problems caused by inconsistencies and more time doing more interesting projects.

As when building a house on a foundation of rock versus sand, we need a good foundation for our machines. In other words, we serve our customers well when we do these things well, and it is a disservice to our customers when we do these things badly or inconsistently. We also do a disservice to ourselves when we don't automate these basic fundamentals because we have made our job more difficult.

Perfection is unachievable because it is undefinable. However, if we produce consistent results, we can make improvements in small increments. The system evolves as we reconsider past decisions and new information and the system gets closer and closer to perfection as we learn how to define it. If a configuration setting is consistent but unpopular, we can have a discussion about improving the situation or developing ways to give users control over that setting. Desired changes and improvements can be filed in a bug-tracking system, triaged, and acted upon in an orderly manner. The installation process can be managed like a software project.

Segal's Law

A person with a watch knows what time it is. A person with two watches is never sure.

The ultimate result of such automation is to make the process self-service. People can install or reinstall the OS on their own, which means they can do it on demand and without taking your time. As a result, you are no longer in the business of installing the OS for people. You are in the business of maintaining the automation that does the process. It is like going from being a worker on an auto assembly line to being the person who does maintenance on the auto assembly line robots. The latter is a much better position to be in.

While creating such automation may take time and money, it is an investment that pays for itself many times over. Customers' productivity is improved by the lack of configuration problems and the ability to do self-service reinstalls without waiting for an SA to do the task for them. It also provides a path for them to request features to further improve their productivity. It improves SA productivity by eliminating work related to fixing problems created by inconsistencies. It eliminates the need to perform installations for a user, during which most of the SA's time is spent standing around waiting. Lastly it improves SA morale: Doing installations manually is really boring!

Friction-Free Laptop Reinstallation

Christine once worked at a site that stored all configuration-related information about a machine in a database (a CMDB, discussed later). The site also kept all user data on the D: drive in desktops and laptops, in addition to having a network home drive. Finally, it used **roaming profiles**, where the users' data and configurations were automatically synchronized across the network at logout and login.

Because of this preparation, her laptop could be rebuilt in a fully automated way.

One day the hard disk in Christine's laptop died. The drive was replaced, and the machine was reconnected to the network. It booted from the network, and after about four hours, with no further interaction required, the machine was restored to its previous state—custom applications and local data included.

While she was waiting for her own machine to be rebuilt, Christine was able to log in to another machine, which automatically installed any additional applications that she needed and her local data so that she could still work productively. Any changes in her roaming profile, including local data, were automatically synchronized to the newly rebuilt laptop

when she logged off the loaner machine and logged in to her own machine again.

This is what a well-developed, mature workstation management system looks like.

8.2 Installation Strategies

Installation of the OS and initial applications can be done by automation, cloning, or manually. The best approach is using automation. However, even if your site uses cloning or manual installation, there are ways to improve on the process to produce more consistent results.

8.2.1 Automation

All modern OS vendors provide a mechanism to automate OS installation. Microsoft Windows has Microsoft Deployment Toolkit (MDT) and other options. RedHat Linux has KickStart, Debian has Debconf, and so on. Solaris has JumpStart. There are also third-party products like KACE and Fully Automatic Installation (FAI).

All of these have the same basic components: They have a way of booting over the network from an installation server. What they boot is a mini-OS that runs out of a RAM disk. Because it does not depend on the underlying storage, it can wipe, partition, and otherwise prepare the machine. Configuration information for the network interfaces, storage, software subsystems to be installed, and so on are received from the installation server. Once the process is complete, the machine is rebooted and the newly installed OS takes over.

An automated installation process does not require a long installation checklist, but some documentation is needed. It should include information about how to start the process, which often involves special keystrokes during a computer's boot process. This documentation is also a place to record any manual steps required after the automation has finished.

Multiple OS Automation

All the vendors' automated OS install processes start with booting an image acquired over the network. This process, which is called PXE booting, relies on DHCP. When any device is turned on and told to start the network boot/install process, it requests DHCP options 66 (boot server hostname) and 67 (boot file name) from the DHCP server. This is how the

client knows which OS image to load and which server to download it from.

DHCP servers can be configured to provide a single answer for all machines on the network, or to provide a custom answer based on Ethernet MAC address, or a mixture of both. However, the MAC address-based approach relies on pre-configuration of the DHCP server. Depending on the DHCP solution, this may require advanced knowledge of which network the client will boot from, or may require duplicating that configuration on multiple DHCP servers. The more scalable solution is to set the options at the network level. Even better, most DHCP servers will support setting those parameters in a single place to apply on all networks configured on that DHCP server.

However, if you provide the same boot image to all devices, how can you support network installation of different OSs? The answer lies in configuration files on the boot server. All boot servers require a per-machine configuration file to set host-specific options, such as the hostname. By initially providing a basic boot loader OS to the device, plus a configuration file that tells it which image to load next, you can use the same DHCP configuration for all operating systems.

8.2.2 Cloning

Some sites use cloning to create new machines. One machine is installed and configured as desired, and a snapshot of that machine's disk is saved somewhere. Subsequently, any new machine is installed by copying that image to its hard disk. The original machine is known as the **golden host** and the snapshot is called the **golden image**.

A small industry is devoted to helping companies with this process and providing specialized cloning hardware and software. An early product was called Ghost. Clonezilla and FOG (Free and Open Ghost) are now popular open source alternatives.

Disadvantages of Cloning

Automating the OS installation process is typically a better solution than cloning for several reasons. First, if the hardware of the new machine is significantly different from that of the old machine, you have to make a separate master image. It doesn't take long before you end up with many master images—which means you are stuck in a situation where any change must be made to each

master image.

In addition, cloning hides history. A clone image may have years of minor changes. There is no record of why each change happened or how. The result is an image that cannot be rebuilt from original source materials. If a bad change is made, the only way to revert is to go back to the last known good image. The specific changes made since that revision are unknown or ill defined. It is likely that you will forget one of the changes, or do them differently, creating confusing variations that make support difficult. In the worst-case scenario, there is no archive of previous images and you have to start from scratch; the resulting image is unlikely to be exactly the same as any of the ones previously created.

Another problem occurs when a new release of the operating system arrives. It is likely to be difficult to reproduce the same changes in the new operating system. As SAs we take pride in being expert enough to know all the settings to set and all the defaults to change, but we are also human and may forget steps or do them differently than the time before. Or perhaps the person who previously did the work has since left the company, so it is being created by a completely different person who has his or her own ideas and standards.

Automated Image Creation

Many of these disadvantages can be mitigated or eliminated by automating the creation of the golden image in a way that can be reproduced easily. Packer (<https://www.packer.io/intro>) and Vagrant (<https://www.vagrantup.com/>) are two such systems that enable this kind of automated image creation. Both have configuration languages that describe the steps required to build an image. The process can then be repeated at any time. If changes to the image are needed, they are added to the configuration file and a new image is built from scratch.

The configuration file includes directives that make it easy to specify a process. For example, the following process might be used: Start with version 7.1 of the vendor install image; add additional files (with a list of source locations and destinations); and install product keys, instructions to modify particular configuration files, specifications for configuring network interfaces depending on their quantity and speed, and so on. This configuration file can be stored in a source code repository, which makes it possible to track changes easily.

Most importantly, when a new version of the operating system arrives, creating the new image may be as simple as updating the initial vendor image listed in the configuration file. If the new OS release is significantly different,

such as the move from Windows 2012 R2 to Windows 10, it may require creating a new configuration file from scratch. However, you have the previous configuration file as a reference for which types of changes need to be replicated.

Hybrid Cloning and Automation

Some sites use a hybrid of cloning and automation. The golden image is used to establish a minimal OS install and the remaining configuration is achieved through other automation, such as the software distribution system that runs periodically on all machines to distribute applications and patches.

This has the benefit that the golden image rarely changes and therefore is easier to maintain.

Use Clones to Speed Installation

A company had a fully automated installation process that was very elaborate and complete. Unfortunately, it was also slow, taking nearly an hour to run. Users of the virtualization environment requested faster installations so they could more easily create temporary machines.

The SAs shifted all the generic installation tasks to the early part of the automation. Tasks that were specific to the machine were shifted to the end. The virtualization manager's storage API was used to take a snapshot of the machine after the generic tasks were complete. The installation software was modified to use the snapshot as a golden image, then use the automation to complete the machine-specific tasks that remained. The new installation time was less than 5 minutes.

The older, slower installation process was still used for physical machines and when preparing new golden images for the virtualization cluster.

8.2.3 Manual

The most basic, and least preferable, way to install and configure an operating system is manually. Insert the installation DVD, boot, and answer a few questions, and the process wipes the computer's hard disk and installs the operating system. Typically the questions relate to network interface configuration, hard disk partitioning, network directory configuration, and which optional software subsystems should be installed. The actual installation may take minutes or hours.

Once this process is complete, the system is in the Evard model's clean state, as described in [Section 7.1](#), and must be configured before it is actually usable. Manual configuration may include installing applications, enabling or disabling various features and options, and so on. There may be dozens or hundreds of steps depending on the organization's requirements.

As discussed previously, manual installation results in inconsistently configured machines. However, there are some instances when manual installation is your only choice. One such situation would be when an operating system is new and the automation has not yet been created. Ideally, in this case you are manually installing the OS as a way of gaining the experience required to automate the process. Another situation is if the site is so small that new machines are installed too infrequently to justify creating such automation. All too often, however, such sites have a single system administrator who is overburdened in ways that would be alleviated by such automation.

Lastly, another situation is when the machine is that of a remote employee with limited bandwidth. Usually automated OS installation systems do not work via remote access networks due to the large amount of bandwidth required to transfer the entire operating system to the machine, or because the initiation process requires a specially configured network. Luckily, solutions to these problems have been developed.

Some systems can generate custom installation media that is burned onto a DVD or written to a USB stick. This can be shipped to the remote worker or made available for download. The installation process is entirely self-contained, or at least self-contained for the initial stages, at which point the system can download the remaining parts.

Other systems provide generic boot media that initiates the installation process, at which point the user inserts the vendor installation DVD. As each file is installed, first an attempt is made to copy the file from the DVD. If the file is not found, it is downloaded over the network. The result is a faster installation that requires less bandwidth.

If your only choice is manual installation, the process can be improved by maintaining a simple document with a bullet list of steps, known as an **installation checklist**. This helps improve consistency even if you are the only person doing the process.

Again, as SAs we pride ourselves on being expert enough to know all the settings to set and all the defaults to change. However, as discussed previously, doing these steps manually is not best. A written list of the necessary steps—

even the most rudimentary checklist—can be a great tool. It enables us to perform well even if we are bored and distracted, or sloppy due to lack of sleep. Best of all, a well-written checklist lets us delegate the process to others. You may be the only person with enough experience to create the list of steps. However, if you write these steps down, other people can be trained to follow them.

Do Small Sites Need Automation?

If your site has only a few hosts that are using a particular OS and hardware combination, it is difficult to justify creating extensive automation. Of course, as the site grows, the benefits of automation become more obvious. Sadly, we often recognize the need for automation only when we have become so overloaded that we don't have the time to create it.

If your site is small you should start investigating and experimenting with automation anyway. Allocate a small amount of time for this task each week, and work on it whenever you get an opportunity (such as when you are waiting for a manual OS install to complete). If you complete it before the need becomes critical, then great. If not, at least you will have made some headway, and completing the automation process will not be such a daunting task. Remember: Installation automation has the additional benefit of reduced support overhead due to the delivery of consistent machines. You will be glad that you have done it, and you will have extended your knowledge and your marketability.

8.3 Test-Driven Configuration Development

Whether you maintain an automated OS install or a set of clone images, you should adopt a formal methodology for tracking which changes are made.

Each change made to the installation process should be documented and the following three aspects should be recorded:

- **The need:** A nontechnical description of the change, or why it is needed
- **The change:** Technical work to achieve the change
- **Tests:** One or more tests that demonstrate the change was successful

For example, a need might be that the host must be added to a Windows ActiveDirectory domain. The change might be the command that connects it to the directory. The test is to do an ActiveDirectory lookup, or to verify that a user

can log into the machine using the appropriate domain name and password.

Another example might be that an updated video driver is required because the default driver does not work with newer hardware. The change would be placing the driver's files in a particular directory so that it is auto-discovered during installation. The test would be that on such hardware, after rebooting, the driver is activated and reports the proper version number. A weaker test would be to simply verify that the files reached the destination directory. This is a weaker test because it doesn't directly test the desired outcome. It is possible that the file might be copied correctly but the OS does not activate it due to it being in the wrong place or other unforeseen circumstances. Avoid such weak tests whenever possible. Test the functionality, not the change.

Formally documenting tests for each need also permits test-driven development (TDD) to be used. In this case, the need and the test are developed first. One then executes the tests, which should fail because the associated change hasn't been made. Next you make the change and re-execute the test, which should now pass. If it does not pass, investigate and revise the change. Once the test starts passing, you know you are done. Document the successful change. Sometimes one discovers that the test itself was underspecified or incorrect. In that case the test is revised and the entire process starts again from scratch. By working this way we build up a library of tests that can be reused anytime we want to verify the functionality of our installation.

Following this methodology makes the installation process more reliable because each new image can be tested and those tests reflect all the documented needs. Any new image should undergo all the tests accumulated so far. If the tests are automated, this is very easy. Some organizations have not only automated the creation of images, but their automation finishes by immediately running the entire battery of tests. This prevents **regressions**, or setbacks, from reaching customers. The moment a regression appears, it is identified and fixed. The location of the regression is easier to identify since it must be due to the most recent change. The end result is that it becomes easier to consistently provide a high-quality service.

The order in which tests are run is important. In a perfect world, you would know which test is going to fail so that it can be run first, saving you from having to wait for all the successful tests to complete. Sadly, we don't live in that world. However, we can adjust the order of tests so that quick tests that are likely to fail run first. The most expensive tests should run last. If there are any manual tests, they should be done after all automated tests have completed successfully. This conserves your time. Reserve your highly valuable time for testing releases

that have proved themselves worthy.

Test-driven development works equally well with automated OS installation and cloning. However, it is particularly important with manually built clones because it assures that the needs are documented. With manually built clones, documentation will exist only if you take a disciplined approach to creating it. With automated OS installs your source code is your documentation; your infrastructure is code.

TDD has one particular downside: Designing tests can be extremely time-consuming. For that reason, some sites use TDD only for the most important or fragile changes.

8.4 Automating in Steps

The secret to automating anything is to start by doing the task yourself. You can't automate something if you can't do it manually. Do the steps manually and take good notes along the way. Use your notes as a guide for creating the automation you desire. Automate the easy items first, track what's left to be done, then repeat.

Start small. Automate just one or two steps. A good first goal is to boot off a network install server instead of installation media. Booting over the network in this way is called PXE booting (PXE is pronounced "pixie"). Get to the point where the machine PXE boots, contacts the installation server, and performs the installation over the network instead of requiring the DVD or other installation media. The process may prompt you to manually answer questions to guide the installation, but that's okay for now.

Once this works, a good next step is to eliminate the need to manually answer such questions. Provide a configuration file with the answers. The machine reads this file and uses those answers instead of prompting the user at the keyboard. This is often known as an **answer file**.

Tip: Frontload Installation Questions

If an installation needs to ask any questions at all, it should ask them all at the beginning of the process. The goal is to be able to answer any questions, such as which operating system to install, and then have the process complete without any other interaction. Otherwise, you will have to keep running back to the machine to see if it has finally gotten to that prompt that appears in the middle of the installation. Or, worse, you may be tempted to just stand there doing nothing as the process runs. That's

bad time management.

While it is strangely satisfying to watch automated tasks run, it is much more satisfying to be able to walk away and work on something else, confident that the installation will complete on its own.

Other good next steps are to include additional software packages to be installed, configuration settings, BIOS settings, firmware updates, RAID controller configuration, and so on.

Automate the easy steps first. This gives you experience that helps you automate the more difficult steps. If you get stuck and there are a few manual steps left over, track the need to automate them in your ticket system or bug tracking system. This gives them visibility with your managers, so they know to allocate time for you to work on them, and also helps you not forget about them.

Make Brokenness Visible

Tom once observed a PC technician use an automated installation process that was slightly broken, requiring manual intervention at certain steps. When asked if there were bugs filed for those steps, or if the brokenness was in any way documented, the technician replied, “Oh, everyone knows about that problem.”

When Tom asked the technician’s boss about these issues, the boss was surprised. He hadn’t done an installation himself in months and had no visibility to the existence of problems. “Had I known,” he reported, “I would have delayed other projects so that he could have focused on fixing those problems rather than wasting time working around them.”

Don’t muddle though brokenness. It isn’t heroic to suffer in silence. Process improvements should be filed right along feature requests and bugs. Amplify their visibility so they get the attention they deserve.

Do not wait until the system is fully automated before using it. The more you use the system early on, the faster you will spot areas for improvement. You benefit from using the working parts over the old manual process even if the new process works for just a single model of machine, or requires manually configuring RAID before starting the process, or still prompts for answers to some questions, or requires manual post-installation steps. If the only improvement so far is that you don’t need to carry around installation media, that’s a worthwhile improvement.

Over time you will improve the system, making it more fully automated, faster, and more resilient to failures due to strange edge cases and errors. This is called **incremental development**. Rather than assuming you can start the project and be done, embrace that this is an ongoing project that evolves and improves over time. Benefit from what has been implemented sooner rather than waiting for it to be perfect.

The last 1 percent can take longer to automate than the initial 99 percent. It is better to automate the easy 99 percent and have a written checklist to direct people through the remaining 1 percent than to delay the use of such a system.

Tip: Don't Let Stop-Gap Measures Become Permanent

There's nothing more permanent than a temporary solution. Give the temporary workaround enough visibility that it isn't forgotten. File a bug or ticket at the moment you create the temporary workaround. Assuming you will remember to do it later means it will be forgotten.

Ultimately, however, we want the installation process to be fully automated. There should be no prompts, other than to ask which operating system is to be installed or to ask "Are you sure?" before the disk is erased. With virtual machines and other situations, even those prompts can be eliminated if the process is started by a command-line tool or web portal that collects such information.

When you think that you've finished automating something, have someone unfamiliar with your work attempt to use it. Start the person off with how to find the documentation but otherwise do not help. If the person gets stuck, you've found an area for improvement. Repeat this process until your cat could use the system.

Automate the Little Things, Too

Tom was mentoring a new SA who was setting up Solaris JumpStart. The SA gave him a demo, which showed the OS load happening just as expected. After it was done, the SA showed how executing a simple script finished the configuration. Tom congratulated him on the achievement but politely asked the SA to integrate that last step into the JumpStart process. Only after four rounds of this was the new JumpStart system completely automated.

An important lesson here is that the SA hadn't made a mistake, but had

not actually fully automated the process. It's easy to forget that executing that simple script at the end of the installation is a manual step detracting from your automated process. It's also important to remember that when you're automating something, especially for the first time, you often need to fiddle with things to get it right.

After the system is fully automated in a way the system administrators can use, you can take it a step further and enable normal users to wipe and reinstall their own machines. This can be dangerous, since they could accidentally erase irreplaceable data. Even so, developers and other technical customers who know what they're doing may benefit from being able to install their OS in a self-service manner. Making such customers self-sufficient also benefits you.

Self-service OS installation has another benefit. You'll always get more people to follow a policy if the easy path is the one that institutes the policy. If you have had trouble getting customers to conform to certain installation and security standards, make sure the self-service automation implements or enables those policies. You'd be amazed at how many customers are suddenly willing to conform to a policy they previously rejected because you've made machine installation so easy. In fact, your job is not to put up roadblocks to the bad behavior, but to smooth the path that leads to the good behavior.

Machine Says, "I'm Done!"

Consider automating as many post-installation tasks as possible. There may be a number of steps that one doesn't traditionally think of as part of the installation that can be automated—for example, the machine adding itself to the inventory system, or playing a loud "beep" so you know it is finished.

At Bell Labs one of Tom's co-workers modified the Solaris JumpStart system to send email to the helpdesk when the installation was complete. The email was sent from the newly installed machine, thereby verifying that the machine was operational. The email that it generated included the hostname, type of hardware, and other information that the helpdesk needed to add the machine to its inventory.

On a busy day, it can be difficult to remember to return to a host to make sure that the installation completed successfully. With this system, SAs did not have to waste time checking on the machine. Instead, they could make a note in their to-do list to check on the machine if email

hadn't been received by a certain time.

While it would have been preferable for the machine to add itself to the inventory, this was a good first step.

8.5 When Not to Automate

A lack of automation can be justified if there is only one host of a particular OS/hardware combination, if the cost of automation is larger than the time savings, or if the vendor has done the world a disservice by making it impossible (or unsupported) to automate the procedure.

The most basic stop-gap measure is to have a well-documented process, so that it can be repeated the same way every time. The documentation can be in the form of notes taken when building the first system, so that the various prompts can be answered the same way.

You'll find that over time these notes lead to automation. An individual step is rewritten as a script. Then another one. Then these scripts are chained together. Soon you have the basis for a fully automated solution.

8.6 Vendor Support of OS Installation

Vendors should make it easy for their operating system install procedure to be automated.

When vendors try to sell us new products, always ask them whether and how installation can be automated. Reject vendors that have no appreciation for deployment issues. Increasingly, vendors understand that the inability to rapidly deploy their products affects the customers' ability to rapidly purchase their products. Some need to be reminded of this.

The only thing worse than not being able to automate the installation of an OS is not being able to install it at all.

An OS That Had to Be Vendor Loaded

Once upon a time, Tom was experimenting with a Unix system from a Japanese company that was just getting into the workstation business. The vendor shipped the unit preloaded with its customized version of Unix. Unfortunately, one day the machine got irrecoverably mangled while developers were porting applications to it. Tom contacted the vendor, whose response was to send a new hard disk preloaded with the OS—all the way from Japan! Even though the old hard disk was fine and could be

reformatted and reused, the vendor hadn't established a method for users to reload the OS on their own, or even restore it from backups.

Luckily for Tom, this workstation wasn't used for critical services. Imagine if it had been, though, and Tom suddenly found his network unusable, or, worse yet, payroll couldn't be processed until the machine was working! Employees would not have been amused if they'd had to live without their paychecks until a hard drive arrived from Japan.

If this machine had been a critical one, keeping a preloaded replacement hard disk on hand would have been prudent. A set of written directions on how to physically install it and bring the system back to a usable state would also have been a good idea.

The moral of this story is that if you *must* use a vendor-loaded OS, it's better to find out right after it arrives, rather than during a disaster, whether you can restore it from scratch.

The previous anecdote describes a situation in 1994, which was a different era compared to today's world. However, history repeats itself. PC vendors preload the OS and often include special applications, add-ons, and drivers. Always verify that add-ons are included in the OS reload disks provided with the system. Sometimes, the applications won't be missed, because they are free tools that aren't worth what is paid for them. At other times, they may be critical device drivers. This is particularly important for laptops, which often require drivers that are not included with the generic OS installation media. Tom ran into this problem while writing the first edition of this book. After reloading Windows on his laptop, he had to add Ethernet drivers. The drivers couldn't be brought to the laptop over the network, because he had no Ethernet drivers. Instead, he had to use a different computer to download the drivers to an external storage device, which he could then plug into his laptop. Without a second computer, there would have been a difficult Catch-22 situation.

This issue has become less severe over time as laptops are now less likely to depend on vendor-specific drivers. Microsoft has also responded to pressure and has made it easier for vendors to have their drivers included on the Microsoft installation media. Although the situation has improved for low-level device drivers, vendors have tried to differentiate themselves by including application software unique to particular models. Doing that defeats attempts to make one image that can work on all platforms. Starting the machine in a known state that you have defined and tested is always best.

8.7 Should You Trust the Vendor's Installation?

Computers usually come with the OS preloaded. Knowing this, you might think that you don't need to bother with reloading an OS that someone has already loaded for you. We disagree. In fact, new machines received from the vendor should have their OS reinstalled. It will make life easier in the long run.

Reloading the OS from scratch is better for several reasons. First, automating the entire loading process from scratch is often easier than layering applications and configurations on top of the vendor's OS install. Second, vendors change their preloaded OS configurations for their own purposes, with no notice to anyone. Loading from scratch gives you a *known state* on every machine. Using the preinstalled OS leads to deviation from your standard configuration. Eventually, such deviation can lead to problems.

Another reason to avoid using a preloaded OS is that eventually hosts need an OS reload. The hard disk might crash and be replaced by a blank one, or you might have a policy of reloading a workstation's OS whenever it moves from one customer to another. When some of your machines are running preloaded OSs and others are running locally installed OSs, you have two platforms to support. This adds to the support burden.

Lastly, you *don't* want to discover, smack in the middle of an emergency, that you can't load and install a host without the vendor's help. It is better to learn this when the machine is fresh out of the box and hasn't been delivered to the customer yet.

8.8 Summary

OS installation is a fundamental building block of our workstation environment. It starts each machine on a good path. If machines start out life inconsistently configured, it causes confusion for our customers and more work for us.

Consistency is more important than perfection. Inconsistently configured machines look sloppy to our users. There is no such thing as perfection, but if we are consistent we can slowly evolve toward more perfect configurations over time.

Strategies for OS installation include manual, automated, and cloning processes. Manual means doing steps by hand. It is difficult to do this consistently, even if one person does all installations. Automated means following the vendor's installation steps but in a way that does not require human intervention. This brings about consistency and, most importantly, records any decisions in code: The decision making can be tracked over time.

Cloning means taking a properly installed machine and copying its hard disk to make new machines. This often improves installation speed, but it hides the history of configuration decisions.

Creating such automation is best done by starting small. Automate the pieces you can automate, leaving the more difficult parts for later. As you gain experience, those more difficult parts will become more tenable. The un-automated parts should be listed in some kind of checklist so they are done consistently.

Exercises

1. Why is consistency more important than perfection?
2. What are the three major strategies for OS installation? When is each appropriate? Which do the authors favor?
3. What's bad about manual installation and how can these issues be mitigated?
4. In what ways is the automated strategy better than manual?
5. Create basic OS install automation for the OS of your choice, and document all the steps that you needed to make. Which improvements to the installation automation would you like to implement next, and why?
6. Modify your OS install setup to enable installing different OSs on different machines, and document the steps you needed to take to do so.
7. What is OS cloning and how can its downsides be mitigated?
8. Why might an IT organization adopt different installation strategies for different operating systems, hardware, or other use cases?
9. Which strategy is used in your environment for OS and application installation and configuration? What could be improved about it?
10. What is test-driven development (TDD) as applied to OS installation and configuration?
11. In [Section 8.3](#)'s discussion of TDD, it was suggested that the test be executed before the change is made. Why would we do this when obviously the test will fail?
12. How could you apply TDD to your current environment? What would the benefits be?
13. List where you got coffee in the last two weeks. Was your selection due to quality, consistency, or availability? If you do not drink coffee, substitute

some other consumer good that you purchase regularly.

14. A traditional way to build something is to start with requirements and build to that specification. This chapter didn't do that, but instead recommended evolving a system and recording the decisions *in code*. Why is that better for OS installation? Might there be a time when the traditional way applies to OS installation?
15. In one of the examples, Tom mentored a new SA who was installing Solaris JumpStart. The script that needed to be run at the end simply copied certain files into place. How could the script—whether run automatically or manually—be eliminated?
16. Given Segal's law, would three watches be better?

Chapter 9. Workstation Service Definition

How do we define the fleet service being provided? What are the requirements for the hardware and software? How do these requirements translate into specifications, and how do those specifications translate into the workstations as they are installed and delivered to the customer?

In theory, this chapter should have come much earlier in the book. We should have started by spelling out how to define the service and then discussed how to provide it. However, the reality is that in your career generally you will fall into a fleet management situation already in progress, or it will be an informal, ad hoc process. (The fact that you understood the previous chapters proves our point.)

Only later in your career will there come a time when you need to create a formal service definition. As more departments or teams depend on you, having a written definition of the service sets expectations for all involved. It gives you a standard against which to measure your own performance. It helps people understand what to expect and what not to expect. It transforms random complaints that get ignored into constructive feature requests that can be triaged, prioritized, and accomplished.

9.1 Basic Service Definition

Service definition should begin with requirements. Once agreed upon, the requirements are turned into technical specifications. For example, the requirements might be that the platform support certain applications at a certain performance level. These get translated into technical specifications of which hardware, operating system, and accessories are deployed.

Platform

We use the term **platform** to mean a specific hardware/OS combination. Some examples include a Dell Latitude 3000 running Windows 10, an Apple MacBook running macOS 10.10, or a Intel-based desktop running Ubuntu 12.10 Linux. Some sites use the term more loosely, consolidating it down to which OS is in use. That is, they might support three platforms: Windows Server 2012, Windows 10, and macOS.

For example, in a call center the requirements might be a narrowly defined

suite of applications required by the call center employees, plus planned upgrades for the next three years. The applications dictate the minimum requirements for the hardware to be selected, and the planned upgrades dictate the spare capacity required for future software releases, as best as the supplier can predict. Management dictates the requirements, and the process of finding products that fit these needs is a collaboration between management and the technical teams involved. These requirements may layer on top of the general workstation architecture, such as described in [Chapter 5](#), “[Workstation Architecture](#),” or they may require changes.

In a general office desktop environment the requirements may be more vague. It may be that the latest OS release must be supported within a few weeks of release, and the budget priorities drive the class of machine delivered. Various accessories may be listed as required, optional but supported, or explicitly forbidden, with the remainder self-supported. For example, a design department might receive official support for a particular model of drawing pad, while others aren’t forbidden but won’t receive official support.

9.1.1 Approaches to Platform Definition

Ideally, customers should be involved in specifying the requirements from the very beginning. Designated delegates or interested managers would choose applications to include in the configuration. Every application would have a service level agreement (SLA) detailing the level of support expected from the SAs. New releases of OSs and applications would be tracked and approved, with controlled introductions similar to those described for automated patching.

However, what generally happens is that the platform definition tends to be controlled either by management, with excruciating exactness, as in the call center example, or by the SA team, which is left to make most decisions, often based on the limited choices presented from vendors. For example, perhaps the OS selection tracks Microsoft’s release schedule, and the hardware selection is guided by the introduction of new models and the end of support of older ones. The team can’t dictate such schedules, but instead must plan their work around the vendors’.

With increasing frequency, companies are supporting multiple desktop operating systems, usually macOS and Microsoft Windows. While providing support for more than one OS is complex, it is less complex than in the past. This is especially true for environments that exclusively use web-based applications. Rather than having to write every application twice—once for Mac, once for Windows—a single application simply needs to be tested for

compatibility with the web browsers of each operating system. There is also a security benefit in having a diverse set of operating systems: It is more difficult to attack an infrastructure that is not a monoculture.

9.1.2 Application Selection

A common pattern is to define a list of which software will be on all workstations. This becomes the base definition of the software standard. All other variations are defined in terms of this list. For example, the finance department may receive the base list plus applications specific to corporate finances. Entire divisions may have custom applications that are included on all workstations in that division.

This software standard is usually a particular OS and a list of specific applications. The list of applications is made up of the corporate standard for word processing and other office applications, in-house applications that all employees need, and commonly requested utilities that can be licensed economically in bulk.

In most organizations this definition process is very open, and there are no formal committee meetings. The system administration team adds to the list items that seem reasonable. This requires SAs who have close relationships with customers and therefore are in touch with their needs.

In other organizations there is a more formal process. Each customer group has a set of software applications they need. These requirements are documented and become part of the design. There is a committee made up of customers, system administrators, and managers who define and approve these requirements. Each new OS release, or even a patch to an existing one, is not deployed until it is certified to work with the approved applications.

The earlier customers are involved in the service definition, the less likely they are to resist attempts to standardize. While SAs think of standards as beneficial, many customers consider standards to be an annoyance to be tolerated or worked around, especially when the first standardized PC definition is introduced into an environment used to having no standards.

Google's Self-Service Software Store

Another way of involving the customers is to enable people to install applications on demand. At Google, an internal web site provides a shopping cart-like application for selecting additional software to be installed on the user's machine. Soon after the user selects the desired

items, configuration management software installs it on his or her machine. The system manages software licenses when needed. It keeps track of which licensed software is on which machines. Employees can view a dashboard that shows how much their machine is costing the company and gives them the ability to “give back” unused software.

This system helps employees be more productive by assuring they can get the tools they need to do their jobs frictionlessly. It also empowers them to save the company money. When given visibility to the information they need to make smart choices, employees tend to make smart decisions.

9.1.3 Leveraging a CMDB

At mature sites, a configuration management database (CMDB) will be used to track all computer assets, including what the machine type is, who the primary user is, and which software is installed on each machine. Usually, the motivation for this system arises from accounting and management needs—tracking compliance with software licensing, understanding who is affected by which changes and upgrades, and so on. However, SAs can also leverage this database for the automation of the install and reinstall processes.

In an environment such as this, when new machines are ordered, the customer specifies exactly which additional packages should be installed when the machine is built. All of this information goes into the CMDB. After delivery, customers can order additional software through an online order processing system. Once the order has been approved, the CMDB is updated and the software configuration management system deploys the software to the designated machine.

The SAs can then leverage this information during installs and reinstalls. The build process should access the CMDB to find out which additional software packages need to be installed, and install them automatically. With such a system, any machine can be reliably rebuilt after a failure. Also, additional machines can be built to the same specifications by cloning the CMDB entries onto another machine.

9.2 Refresh Cycles

There needs to be an orderly way of identifying older hardware and replacing it. Hardware eventually becomes obsolete. The newest OS releases no longer support it. New applications require more powerful hardware and run too slowly

on older models.

Refresh cycles are planned upgrades for old hardware. For example, a fleet might be on a 3-year refresh cycle, which means that machines are retired and replaced after 36 months.

Without a properly designed and enforced refresh policy, organizations tend to refresh based on who complains the loudest, or the issue becomes a political volleyball used to play favorites. The best refresh cycle policy prevents this kind of dysfunction by design.

9.2.1 Choosing an Approach

There are various approaches to hardware refreshes. The best one for most environments is the generational approach. In an environment where there is a predominantly tech-savvy user base, an approach based on hiring date can also be a reasonable choice.

There are also many other variations that we do not recommend. They are included here to describe their disadvantages, so that you can effectively argue against those approaches if it looks like your company is heading in that direction—or worse, is already there.

Generational

Likely the best strategy is to plan refresh cycles on a generational basis. Workstations are installed in large groups, each called a **generation**. To rein in support complexity, there is a limit on the number of generations being supported at any time. For example, a call center with hundreds of similarly configured workstations might define a new generation every year or two as older models are no longer available from the vendor. When one generation is rolled out, all machines from the oldest generation are replaced. Thus, the number of generations being supported at any time is constant, except during a transition period.

Hiring Date

Another strategy for refresh cycles is based on hiring date. For example, people might be permitted to replace their workstation every three years. At Google the policy was engineers could upgrade every 24 months and everyone else every 36 months. (It's good to be an engineer.) This refresh cycle works well because people feel ownership of their machines and such a policy gives them a certain amount of control over the upgrade. While they might be eligible for an upgrade,

they may postpone it if they are happy with their current model, or know that waiting a few months will mean a newer Apple MacBook model will be announced.

Departmental

Other organizations push workstation refresh decisions to the individual departments. In this case, each department creates its own strategy and policy. Some departments will do a good job but most will not. Those departments do not have the experience and knowledge of an IT department, nor have they read this book. There will be a duplication of effort as each department reinvents policies, evaluates models, and so on.

Short-Sighted Approaches

Many organizations make decisions that are short-sighted. These decisions include foregoing upgrades for multiple years, ignoring the corporate hardware standards, or buying models that have a deceptively low initial purchase price.

We call these decisions short-sighted because they optimize the department's budget but create an overall cost increase to the organization as a whole.

Skipping upgrades for multiple years saves budget money this year but costs more overall. Users are left with older, failing hardware, which results in lower employee productivity and higher support costs.

Ignoring the corporate hardware standard is more time-consuming in the purchasing process and usually means the organization does not benefit from volume-purchasing agreements. Sometimes this is done to select a lower-price model, though often it entails purchasing from the consumer or small-business product lines discussed in [Section 6.1.3](#). These models highlight a lower purchase price but obscure a higher total cost of ownership. These additional costs will be invisible to the person making the purchasing decision and will be borne by the IT department.

Lastly, such policies leave the refresh cycle vulnerable to favoritism and politics. In a highly political environment the result is that certain people will receive a new laptop frequently while others suffer with older machines.

Another anti-pattern is the “shift down” strategy: Any new hardware purchased is given to the most senior person; that individual's old hardware is used to upgrade the most senior person with the previous-generation hardware, and that individual's old machine is used to upgrade the most senior person with the next-generation hardware. Each time the organization purchases a new

machine, it creates multiple upgrade projects. Each involves considerable work for the IT team and the disruption of switching machines for the user. This is often two to four times more work for the IT team than a simple generational strategy.

9.2.2 Formalizing the Policy

Whatever the refresh policy is, it should be a written policy, signed off by management. It should be accessible, like all policies, on an internal company web site. It doesn't need to be a 100-page document. In fact, the shorter the policy, the easier it is to understand and implement. It takes only one sentence to explain that employees are eligible for a new machine when their current one is 36 months old, and that the new machine is selected from the standard models available at the time.

Also provide a link for people to look up more specific information, but keep that page simple, too. Keep it to a single screen. If the hardware for the next refresh has already been selected, provide a table to show which new model is available. Make it easy for people to determine when their machine is eligible for refresh by providing a dashboard that displays all hardware attributed to them.

9.2.3 Aligning with Asset Depreciation

The refresh cycle should be aligned with the company's asset depreciation policy. This sometimes involves advocating to have the depreciation policy changed. Suppose most hard goods are depreciated at your company on a 5-year schedule but computers are expected to be retired after 3 years. Then when a computer is retired, it still has 2 years of depreciation remaining and cannot be disposed of for 24 months, unless the company is willing to pay 2 years of depreciation all at once. If, instead, computers are depreciated on a 3-year schedule, replacing them at retirement is easier.

Case Study: Defining a Refresh Strategy

When Tom arrived at Cibernet, he learned that the company's computers were on the same depreciation schedule as the furniture: 20 years. There was no refresh policy, so most of the fleet was aging and unreliable. He worked with the CFO to establish a 3-year refresh policy that would be aligned with a 3-year depreciation schedule.

Tom's primary concern was that the fleet was upgraded every 3 years.

The CFO's primary concern was that spending was predictable and, preferably, evenly distributed throughout the year. Tom proposed that a 3-year refresh cycle meant that, on average, one third of all machines would be replaced each year. The CFO agreed.

The next issue was how to distribute that spending over the year. Tom presented three options: upgrade 33 percent of the fleet once a year, or do proportional batches two, three, or four times a year. The CFO requested quarterly expenditures, and Tom was able to establish a routine around that.

At the start of each quarter the oldest 1/12th machines were identified and upgraded. The quarterly upgrade process became a repeatable process that could be improved iteration after iteration. About once a year Tom's team established a new hardware standard, and over time support became easier because variations in hardware were capped at three generations.

When management understands the computer life cycle and IT understands the finance cycle, collaboration improves and global optimizations become possible. It becomes easier for SAs to get funding for a dedicated deployment group, a repair department, and so on. In short, it is your job to educate management about the refresh cycle.

Hidden Network Access Controls

Often replacing a machine is not as simple as just replacing the hardware. There are many dependencies lurking behind the scenes. It is best to have all such dependencies documented in a way that makes such transitions easier. The reality, however, is that the only such documentation is likely to be the configuration data of various firewalls, routers, and application servers involved.

A cautious approach is to permit overlap time, where the old and new machines are used in parallel until any issues are resolved. The old machine is then removed. When doing this it is best to set a time limit or people will end up with two machines forever.

Unfortunately, such overlap brings its own problems. Christine was in a highly regulated environment that enforced certain access controls via firewall rules specific to a workstation's IP address, which also required a static IP address configuration for the workstation in DHCP. During overlap time the new machines also needed static assignments and the

firewall rules had to be replicated for the new machines' IP addresses. Simply placing a new machine at the old machine's IP address was insufficient because the old machine was still using it. At the end of the overlap time the old firewall rules and DHCP configurations had to be deleted.

All told, the need for overlap time doubled the amount of work for the firewall and DHCP teams. Considering that firewall configurations could be updated only during specific change-management windows, and that there were hundreds of machines being upgraded in each batch, the entire upgrade project was highly complex, labor intensive, and risky. It was also frustrating to the customers, who often had to wait days for issues to be fixed.

The lesson learned was that you need to identify as many dependencies ahead of time as possible.

9.3 Tiered Support Levels

Some environments require their IT team to support any and all operating systems. Typically this happens in university and research environments, where experimentation with new technology is their business.

This can be a support nightmare. Trying to support too many platforms spreads the team too thin. No one platform is supported very well. It is a better use of resources to provide full support to a few, specific platforms and have a policy of limited support for all others. This is called a **tiered support system**. The majority of users will be drawn to the better-supported platform, while the people who demand tailored, boutique, or highly customized platforms are usually technical enough to support themselves anyway.

When Tom was at Bell Labs, his group was asked to support just about every kind of computer and OS one could imagine. Because it would be impossible to meet such a demand, it was established that some platforms would receive better support than others, based on the needs of the business. "First-class citizens" were the platforms that would receive full support. SAs would receive training in hardware and software for these systems, documentation would be provided for users of such systems, and investments would be made to assure that OS installation, configuration, and updating would be fully automated. This would permit these hosts to be maintained in a cost-effective manner while providing first-rate support to the end users.

All other platforms received less support, usually in the form of network configuration information, security guidelines, and best-effort support. If the device caused a problem, the network jack it was connected to could be disconnected without warning. SAs could attempt to resolve an issue with an unsupported item for 30 minutes but no more.

If your environment has a 30-minute rule for unsupported hardware, it is important that this rule is communicated to users before the 30 minutes begins. If, instead, you get to the end of the 30 minutes and remind the user that you are at the time limit, that person will be surprised and upset. If you take a moment to remind the user of the 30-minute rule at the start, the individual will feel that the 30 minutes spent with him or her is a gift.

The 30-Minute Rule for Unsupported Hardware

When Tom was at Bell Labs in the late 1990s, supporting Linux workstations was difficult because the kernel supported very few video cards. Members of Tom's team accumulated a very short list of video cards that had been verified to work. Others wouldn't work no matter how many combinations of drivers and settings were attempted. Linux just didn't have the proper software.

Around the same time one team member would disappear for hours at a time. We learned this usually meant he was stuck in someone's office trying to get an unsupportable video card to work. We learned this only because after hours of failing, he would give up and the customer would call our manager to complain that his team was a bunch of idiots who didn't know anything—because obviously we should hire people who can break the laws of physics and make software drivers magically appear out of thin air.

This situation was bad for many reasons. First, it resulted in unhappy customers. Second, an employee lost a day of productivity. Considering a typical SA's salary, it would have been cheaper to buy the customer a dozen supported video cards.

Our manager adopted a new policy called “the 30-minute rule.” If SAs came across a video card that was not on our list, they were to stop what they were doing, take their hands off the keyboard, and back away. They would then say, “I'm sorry but this is not on our list of supported cards. My boss will let me try to get it to work for 30 minutes. At the 30-minute mark, I will install one of our supported video cards and bill your

department. Is that acceptable?”

The customer would agree and then the SA would attempt to get the unsupported card to work for 45 minutes before giving up and installing a supported card. The additional 15 minutes was “built in” to the policy and made customers feel like they were getting something extra. It delighted them.

What we learned from this:

- Customers would always agree to the offer. They’d rather have a working system than the dream of a better video card.
- If the SA forgot to make the speech until the end of the 45 minutes, our manager would get the same kind of irate phone calls as before.
- If the SA made the speech at the start of the 45 minutes, failing to get the video card to work would result in our manager getting fan mail because customers felt they got an extra 15 minutes.

It didn’t matter what the SA did for 45 minutes. The card was never going to work, but the customer didn’t want to believe that. Yet, by simply making the speech 45 minutes earlier, the customers were much more satisfied.

Every SA team should have a 30-minute rule.

When creating a tiered support system, you should define and document the tiers and determine a way to promote a platform to a new tier. More formally, a tiered support system can look like this:

- **First tier:** These machines are fully supported, with a list of benefits including hardware and software driver support, easy OS reinstallation, and automatic software updates, plus new OS updates are tested on these platforms before they are deployed.
- **Second tier:** These machines receive best-effort support. These machines may connect to centralized services if it happens to work but with no support and no promise that what works today will work tomorrow. For example, non-Microsoft email clients will talk to MS Exchange if the IMAP connector is enabled. Such clients won’t be blocked, but if a particular client stops working after the next MS Exchange patch, the SA team won’t attempt to fix it, nor will they contact Microsoft for support.
- **Ad hoc support:** These machines receive no direct support. All support is self-support. They might be required to be in a specific VLAN that is

firewalled off from the rest of the company.

- **Forbidden:** All other machines are not permitted to connect to the corporate network.

Many sites simplify this by having two tiers: first tier and ad hoc.

Document definitions of the support levels offered. This can be as simple as a bullet list for each tier. The first tier's list includes automated installation and patching, the SLA for requests submitted to the helpdesk, the fact that new releases are tested against official corporate applications before they are deployed, and so on. The second tier's list states that support is available for connecting such machines to the network, and it recommends a (manual) install procedure. Otherwise, it refers people to a wiki and mailing list established to help users help each other. Ad hoc support dictates that network configuration information, such as an IP address and netmask, will be provided, but specific instructions for how to configure the device will not be. Ad hoc support should also include under what circumstances the device will be disconnected from the network—for example, if it is interfering with normal operation of the network.

There should be an official process to promote something from one tier to another. Customers should be able to request a platform be promoted. However, since the support burden of a first-tier device is high, promotion to the first tier should require budgetary approval from management, the helpdesk should sign off that they are prepared to support it, network and systems engineering teams should be given time to create approved configurations, and so on. These requests are not taken lightly and customer expectations should be set accordingly. Promotion to the second tier would require considerably fewer approvals.

Some promotions are instigated internally. For example, each new major release of Microsoft Windows might be promoted by mandate. The IT organization might have a goal of being able to support new Windows releases within a certain number of weeks, with deployment as the default after stability criteria are met. A new platform may be promoted due to business initiatives that include that particular technology, or due to the SA team predicting a rise in its popularity.

Promoting Can Be Better Than Banning

Sometimes it is cheaper to promote a platform than to deal with the headaches caused by customers' own botched installations. One platform had the ability to take down the network if it was improperly configured.

A naive user enabled all features rather than carefully selecting the specific features needed. This turned the device into an 802.3 Spanning Tree Protocol bridge, which disrupted the network and caused a major outage. (Modern network switches have a defense against this.) After numerous disruptions resulting from many users carelessly enabling this feature, the platform was promoted so as to take the installation process away from customers.

The automated installation process resulted in a configuration that enabled only safe features that people actually needed. This configuration was also more secure than the default configuration. Customers who had previously thought it was “intrusive and bothersome” for the SA team to request specific security settings never complained when those settings were on by default. The investment in resources to create and maintain this automation paid for itself many times over.

9.4 Workstations as a Managed Service

Some organizations bring together all the pieces of fleet management into one big package provided to users at a fixed cost. This is often called a **managed workstation service**.

The package of services includes all elements of the workstation life cycle: the initial machine and a replacement every 36 months, pre-installed OS with self-service OS reinstalls, periodic OS and application updates, off-machine file storage, helpdesk support, better security and privacy, repair depot support, and so on. Departments are automatically billed a fixed monthly cost, which makes the cost predictable and easy to budget for.

The benefit of this strategy is that it provides a richer service to departments than they could provide for themselves. The service is provided less expensively due to the economy of scale. The service can be provided more expertly since SAs’ attention is undivided.

Workstations as a managed service is often called a **leased workstation** model because of the way it is billed. However, this is more than just a billing model: It is a strategy that permits large organizations to provide a rich workstation experience to many departments.

Such systems are less expensive because they reduce duplication of effort. Large organizations often find each department reinventing the wheel, creating its own OS installation system, patching system, helpdesk, and support infrastructure. This duplicate effort is very expensive.

The managed service model permits concentrated expertise to benefit the largest number of users. Without centralization, some departments will do a better job than others, creating neighborhoods of technology haves and have-nots. With centralization, a professional team is dedicated to providing and managing the service. It is not a part-time job, a hobby, or work done when other priorities subside. This team can hone their skills through concentrated effort.

There are downsides. The service can appear expensive to customers who compare the monthly cost to the price of a PC, ignoring all the other services. A new PC can be purchased for what they'll be charged for several months of service. If one ignores the cost of support, disk storage, network connectivity, and all the services included in the package, it can look very expensive.

University Managed Workstations

A major university in North Carolina instituted such a system and found varying degrees of participation. Nontechnical departments opted in immediately, since this was an obvious upgrade compared to their current patchwork of services. Technical departments generally opted out because they felt the service was more expensive than their current system of free graduate student labor—though some did opt in later, when convinced that permitting graduate students to do research might be a better use of their time.

The lease model is generally used by organizations that want to go from individual support (chaotic, no policy) to a standardized platform. This is common at universities and research organizations where individual departments require flexibility and have the ability to resist top-down decisions. It is designed as an opt-in system not only to ease resistance, but also to permit the global organization to start small and grow as it gains experience. If participation rates rise or fall, that change is an indication of the perceived value of the service being offered. If participation rates are dropping, it is a sign you aren't listening to your customers.

We like the concept of the managed workstation service because it brings all the pieces of fleet management together in a unified, professional way that reduces cost and improves service. It is very common at colleges and universities, research organizations, and large businesses.

9.5 Summary

Workstations are a service we provide to the customers of our organization. As service providers, we must define what we are providing if we are to communicate with our customers what to expect and what not to expect. This definition also gives us a way to measure whether our actual performance matches what we intended to provide. Start by understanding your customers' requirements.

The service definition should include the refresh cycle: a schedule, policy, and process defining when old machines are replaced. Typically companies replace workstations every two or three years. Some companies replace all equipment of a particular generation at once. The refresh policy affects capital budget planning, IT resource planning, and application deployments.

Supporting many types of platforms overloads an IT team, so the policy should state that only a certain number of platforms will be supported. One approach to reaching the goal of having only a limited number of platforms to support is to introduce a tiered support system, where an IT team provides first-class support for a small number of platforms, and lesser support for others.

Some companies bring together all fleet-related services into a package of services, often called a managed workstation service. For example, an IT department might provide service that includes a workstation (laptop or desktop), network access, automated software updates, helpdesk support, and a replacement every 36 months. This approach offers a better economy of scale than each department managing its own small fleet.

Migrating to a new platform standard is difficult but can be done successfully by abiding by a few simple guidelines. Involve customers early in the planning stages. Communicate in the customer's language. First expose a small group of customers to the new configuration, fix any problems, and then repeat with successively larger groups. Provide both the old and new systems during a transition interval; do not try to flash-cut to the new system all at once. Use a ratcheting strategy to prevent regressions. Lastly, set a cut-off date at which point legacy machines will lose network access; otherwise, the project will never end.

Exercises

1. What are the elements of a workstation service definition?
2. Most sites have no service definition and simply rely on an informal or ad hoc definition. Why do they do so? How is it possible to have a service without a definition?

- 3.** What is tiered support?
- 4.** Give examples of tiered support from organizations you've been a customer or a member of.
- 5.** Explain the 30-minute rule for unsupported devices and how you would implement it in your organization.
- 6.** What is a refresh cycle?
- 7.** Describe your IT organization's refresh cycle. How does it work? How would you improve it?
- 8.** What is a configuration management database (CMDB) and how is it used?
- 9.** Which CMDB is used in your organization? If there is none, how would the existence of one improve your organization's ability to provide support?
- 10.** Describe your organization's workstation definition. Is it formal or informal? Which platform(s) are supported? Is there tiered support? What is the refresh cycle? How do customers pay for hardware and your IT department's support?
- 11.** In your IT organization, describe a time where a new standard platform was rolled out. How was resistance addressed? Describe the timeline of the project and whether you consider it a success.

Chapter 10. Workstation Fleet Logistics

Workstation fleet logistics is the business process of physically delivering new workstations to users. Workstations are either laptop or desktop computers, generally assigned to a specific individual. Fleet logistics can be a very large endeavor in a big organization with a high rate of workstation churn or turnover. While the previous chapters focused on technical matters, this chapter is about the high-level process that delivers the final result.

Companies have different names for fleet logistics. Some call it fleet operations, fleet deployment, delivery, or inventory management.

Fleet logistics is most efficient when done at scale. This way we benefit from the economies of mass-production.

Our first examples assume an organization large enough that each function is done by a different subteam. It is easier to deconstruct and describe the organization this way. Later we show how smaller organizations can achieve the same thing with a single team or with a part-time fleet coordinator.

10.1 What Employees See

The best way to envision fleet logistics is by first looking at the end result, and then looking inside to see how it is accomplished.

What new employees see is that on their first day their workstation—whether it is a thin client, VDI, laptop, or desktop—is prepared and waiting for them.

What existing employees see is a little more revealing. They see people carting around new machines to be delivered and removing old ones. A few days before a new co-worker arrives, someone from this team shows up to install a new workstation.

Existing employees also know that machines are replaced with newer models now and then. They receive upgrades on a two-or three-year cycle, based on their hire date. Or machines are managed in groups, such as in a technology refresh project where every workstation is upgraded.

10.2 What Employees Don't See

While employees see the end result, they are unaware of what goes on behind the scenes to make it all happen. Many processes work in concert to deliver the entire service.

In a small company that hires new employees rarely, once or twice a year, it would be reasonable to expect a single system administrator to take care of all aspects of the process. She would determine which model machine to purchase and any accessories and add-ons that are needed. She would shepherd the order through the company's purchasing process. When it arrived, she would install it at the new hire's desk, enable the network jack, load the operating system and applications, and do any other necessary preparations.

In contrast, if many machines were installed each week, a division of labor would be more efficient. Having highly skilled system administrators rolling machines on carts to various offices is a waste of their expertise. It would be better to have a team of less-skilled people delivering the machines, and a separate team responsible for technical work such as maintaining the OS installation automation.

To help understand the processes, let's imagine what would be required in an exaggeratedly huge company. Exaggerating the size accentuates the details. Later we can extract out the components and scale the process down.

To achieve this exaggeration let's imagine a rapidly growing company. This company hires 100 new employees each week, all of whom need new workstations delivered at their desks before they arrive. Let's also imagine that this company replaces all workstations on their third birthday and there are approximately 50 upgrades to be processed each week. In total, 150 new machines are delivered every week and 50 old machines are taken away and recycled.

Delivering 150 new machines each week averages to about 4 per hour. At that rate it is important that the process never stall. Anytime the process stalls, it creates a backlog that must be dealt with later.

To keep the machines flowing, we mass-produce the process. We set up a prep room where machines are prepared in an assembly-line fashion. The machines are delivered to their final destination and the delivery people plug in keyboards, monitors, network cables, and so on. The network group preconfigures the network jacks ahead of delivery. Behind the scenes, the OS installation automation is maintained by one team, while another develops the software applications that drive the entire process.

The division of labor requires many teams:

- **Purchasing team:** Procures machines to be installed
- **Prep team:** Prepares the machines for use
- **Delivery team:** Delivers the machines to the users

- **Platform teams:** Define the hardware/OS platform and are responsible for OS-related automation
- **Network team:** Configures network jacks and plans network capacity
- **Tools team:** Develops applications that coordinate the entire fleet program
- **Project management:** Manages individual projects and day-to-day operations
- **Program office:** Oversees the entire fleet management program

10.2.1 Purchasing Team

The purchasing team orders machines in bulk. The number of machines to be ordered is based on business plans such as special upgrade projects, or from forecasts based on hiring projections, analysis of how many older machines are coming due for refresh, and so on.

This team and the platform team decide jointly which models and options to purchase. The platform team advises on the technical issues while the purchasing team sets the budget and purchasing logistics.

At high volumes it makes sense to limit the number of models that are available. Reducing the number of models reduces inventory costs and makes support easier and less expensive. For example, there might be a standard desktop and a standard laptop. Some companies have a special model for engineers who require particularly beefy machines, or a laptop model that is particularly light weight for people whose jobs involve a lot of travel. Special requests might be simply forbidden, or they might be handled by a special process where new hires take one of the standard models initially and then trade it for their specially requested model later.

The purchasing team should place the orders and track them until arrival at the prep room. This team is also responsible for vendor management and contract negotiation. This goes for all vendors, including those that supply hardware and software, and any hardware disposal companies used.

10.2.2 Prep Team

The machines that are ordered get delivered to the prep room, which is where the prep team prepares the machines for use. Global companies will have many prep rooms, as hardware will be ordered “in region” because of regional variations in keyboards and power plugs, for example.

At this scale, it doesn’t make sense to unbox the machine and configure it at

the future customer's desk. Instead, the process is completed for all machines in the prep room. Once prepped, they are delivered to the users' desks, where they are cabled and tested. Working in batches makes the process much more efficient.

Case Study: Prep Room Process

At one site the day started by determining how many desktops and laptops had to be prepared that day. A web-based dashboard provided this information. It also generated printouts that listed where each was to be delivered.

The appropriate number of machines were taken from inventory and unboxed. One person would haul them from the storage area. A person in the prep room would unbox them and then move them on to the next person, who was responsible for the next step. By working in parallel, the entire process flowed very quickly.

Laptops moved through the process with the box they arrived in because it contained various adapters and other parts that the future user needed.

The desktop models were unboxed and handed to the person doing the OS installation. The keyboard, mouse, and any cables were put in a bin on the delivery cart. The box and remaining refuse were discarded.

The prep room had an area for laptops big enough for eight to be processed at a time. There were eight laptop power cords and eight network cables. Each machine could be quickly connected and powered on, and the OS would be installed. When finished, the laptop was put back in its box and onto the cart for delivery. A preprinted sticker was attached to the box that listed the recipient's name and desk location. Because the purchasing system was highly automated, the inventory database was prepopulated with the serial number and other information. Better vendors deliver serial numbers electronically to make this kind of automation happen.

The prep room's desktop area had enough room for 12 desktops. It had 12 bundles of cables. Each bundle had a power cord, a network cable, a keyboard cable, and a video monitor cable. The keyboard and video cables went to a KVM (keyboard, video, and mouse) system allowing connection to any one desktop machine at a time. There was no need for a mouse at this point. Once the OS was installed, the preprinted sticker was

attached to the desktop, which was loaded onto the delivery cart.

Once a cart was full, the delivery team went to work delivering the equipment. Laptops were delivered in their boxes. Desktops were connected to their peripherals.

While creating this system, this site learned a number of hard lessons. The most important was to identify bottlenecks and resolve them quickly. To make sure that the OS installation wasn't a bottleneck, the network in the prep room had to be specially engineered to have enough bandwidth for 20 machines to be installing simultaneously.

Often the bottleneck was a single failed machine being debugged while the others sat on the cart waiting to be delivered. The prep team found that if a machine had a hardware problem, it was better to set it aside and substitute another machine. The suspect machine could be better diagnosed and repaired by other, more technical personnel.

Using a prep room has many benefits. Working in batches is much more efficient. Unboxing 12 machines in a row, and then processing them as a group, takes a lot less time than unboxing a single machine and doing some other work before coming back to unbox the next one of the 12 machines. The specially engineered network connection prevented any network congestion from affecting other users in the building. For sites that use **network access control (NAC)** based on MAC addresses to restrict unauthorized hosts from accessing the network, the prep room network will need special NAC configuration. Isolating these settings to one physical room makes it easier to do so securely.

The team working in this room often consists of one manager coordinating many unskilled laborers. This division of labor is more economical than having highly skilled system administrators do this work.

10.2.3 Delivery Team

The delivery team delivers the machines to their destinations. Desktops are delivered to the individual desks where they are cabled, the monitor is unboxed and attached, and so on.

Laptops are easy to steal, so they might be delivered to the manager or group administrator. If there is a local physical IT helpdesk, they might be delivered there for pickup. New hires would then pick up their laptops as part of their first-day orientation.

For sites with a local helpdesk, one optimization is to locate the prep room near the helpdesk so there is less distance to travel. Another is to eliminate desktops and provide only laptops. This way there is one less delivery process to orchestrate, fewer platforms to support, and so on. It also means that the helpdesk never has to visit someone's desk; instead, the person can bring his or her computer to the helpdesk.

Both the prep and delivery teams need to reduce the monotony of their tasks; otherwise, boredom will lead to low morale and other problems. To keep employees from becoming bored, commonly the prep and delivery teams are one group, with people alternating roles, or doing prep in the morning and delivery in the afternoon.

One benefit of mass-production is that over time we notice opportunities for optimization that we might not have spotted otherwise. For example, we may observe that delivery is more difficult when the hallways are filled with people going to and from lunch. Therefore we can avoid doing deliveries during those times, focusing on prepping instead. In this case, we can get more work done not by working harder, but by simply scheduling when things are done.

Another optimization is to put jingle bells on your delivery cart so that everyone knows someone is getting a new machine. Why you would want to do that is explained in [Section 49.1.1](#).

10.2.4 Platform Team

The platform team is responsible for defining the final product: the workstation that people use to do their jobs. This team is concerned with all the technical aspects of the machines themselves. This includes model selection, OS installation automation, and OS upgrades and patches. The platform team is also responsible for verifying that new hardware, especially video cards, are compatible with the OS.

Members of the platform team work with the network team to define and create the prep rooms. They work with the purchasing team to select new models and deprecate old ones.

The platform team provides technical support to the prep team. If the OS installation automation breaks or doesn't work on a particular machine, the platform team gets involved to resolve the issue.

Typically each major operating system has a separate platform team. For example, there may be separate teams responsible for Microsoft Windows platforms, the macOS platform, and the Linux platform. There is often overlap

since one hardware platform can be used with two or more operating systems. They may also share infrastructure, such as the PXE-based installation servers. The platform teams are jointly responsible for engineering the prep rooms. In most companies these teams report to the same manager or are simply one big team with different people specializing in one platform or another.

A platform team is made up of highly skilled system administrators. Their salary is commensurate with their responsibility and rare abilities. The same small team supports the platform whether there are 50 or 50,000 machines. The work is very technical and highly specialized. It requires that people stay up-to-date with the rapidly changing hardware industry as well as with the platform of the OS vendor they specialize in. Debugging an installation server problem requires expertise in networking, protocols, and operating systems, and arcane knowledge about how this all fits together.

10.2.5 Network Team

We discussed making machines self-configuring using DHCP in [Section 5.4](#). For this strategy to work, the network itself must be configured to accept these new machines.

Workstations can talk to the network wirelessly (via WiFi) or via wired connection (Ethernet). Once a WiFi network is set up, there usually isn't any additional configuration needed for each new machine. However, the network engineering team does need to address capacity issues. Each new machine brings the network one step closer to being overloaded. Dealing with this issue should be a normal part of the monitoring and capacity planning process.

For sites that use physical connections, the simplest way to ensure that a network jack is configured prior to the desktop's arrival is to preconfigure all network jacks. Each office is pre-wired with two or more network jacks that are preconfigured on switch ports. While this costs more in network switch capacity, it greatly reduces the labor for the network team and makes the users more self-sufficient. The best process for configuring network jacks is the process that doesn't require the network team to be directly involved in each new machine's installation.

Such network configurations are not always possible. Often different employees are on different VLANs. For example, there may be a general VLAN plus a special VLAN for engineers or system administrators. In this case you should configure jacks for the majority VLAN and handle the others as a special case.

For sites that use NAC, each new host will need to be configured into NAC before it is delivered to the customer. This step is typically automated and incorporated into the installation process. Good vendors can supply this information electronically in advance, along with the serial numbers, so that this information can be automatically loaded into inventory and NAC.

No matter which variation you choose, there needs to be coordination between the network team and the fleet logistics team so that the new-machine delivery process runs smoothly. The network team should have a delegate who attends all of the fleet team's engineering and process meetings. There should be a special communication channel for network issues related to new-machine delivery so that the person receiving the call has special knowledge of the delivery process that will help her resolve the issue more quickly. Since stalling the delivery process creates a backlog, problems should receive special attention. If all new hires start on Mondays, deliveries might all be done on Thursdays and Fridays. This means the networking team has to provide this special support only two days a week.

10.2.6 Tools Team

The tools team develops applications needed to manage the entire process. The fleet logistics organization needs applications that track and coordinate their work: everything from the software that coordinates tasks between teams, to the dashboards that give status information to management. Those preprinted stickers don't print themselves, after all. Some of the applications typically needed include

- **New-hire delivery process automation:** This system determines who should receive machines during a given week and which kind of machine. It might scan the human resources database to determine who the new employees are and, based on job classification and title, which model of hardware they are to receive. For example, at Google typically salespeople get a laptop and engineers get both a laptop and a desktop.
- **Refresh process automation:** A dashboard shows employees their eligibility for a refresh and permits them to click to order an upgrade. Purchasing approvals can be automated by emailing the approving manager a link to a purchase approval site. When an old hardware model needs to be eliminated, automation generates emails to the appropriate machine owners, who are directed to the dashboard to order replacements.
- **Forecasting and ordering automation:** This system creates the first draft

of any order for the purchasing team to issue to a vendor. It also forecasts the quantities and types automatically.

- **Project management dashboards:** These dashboards show the flow of machines in the installation process, generating metrics to track machines per week, percentage delivered on time, and other service-level objectives.
- **Remote logistics automation:** Remote workers and small offices have special needs. It may be more economical to prepare machines at a large office and ship them to smaller offices and the homes of remote workers. Software that supports these processes may track machines as they are purchased and prepared, as well as print shipping labels and take care of other critical tasks.

While having all these applications is nice, smaller sites can get by with spreadsheets, especially multiuser shared spreadsheets such as Google Sheets. You can also simplify processes by having machine refreshes tied to work anniversaries instead of complicated schedules.

10.2.7 Project Management

A large operation like the example organization also requires a lot of project management. A **project** is defined in terms of outcome. Delivering a certain number of machines in a fixed time frame is a project. The project will have milestones that can be tracked, such as the number of workstations that should be delivered each week. A **project manager** oversees the process using various techniques to see that the objectives are met.

Fleet logistics includes a mixture of long-term ongoing projects, such as a refresh of one third of the workstation fleet, and some shorter-term projects, such as introducing a new platform. Before a new release of an operating system can be used, the installation process must be updated, the new OS must be validated and tested, and so on. The objective may be to fully support the new OS release by a certain date. Coordinating all these tasks for the platform team at a large company is the work of a project manager.

10.2.8 Program Office

A **program** is a group of projects. For example, everything to do with fleet logistics might be considered one large program. The fleet logistics program would cover all hardware at a company—not just workstations, but also servers, printers, phones, storage, network devices, and so on.

The **program office (PMO)** manages all programs, allocating resources and

setting standards for processes and reporting. The PMO consolidates weekly reports on the various projects and reports on the overall program to management. The PMO may also coordinate steering committee meetings, provide upper management with reports on the various programs, solve resource issues, handle prioritization, and so on.

The program office creates standards for how the individual projects will be managed. For example, it creates a common terminology to avoid confusion, establishes similar processes across all teams, and replicates the teams across all divisions or buildings.

In fleet logistics, the program office defines the product as seen by the customers. It works with the various divisions and customer groups to define what the service is (laptops, desktops, or both) and how it is delivered.

The program office often represents all teams when addressing executive management at budget meetings. It also determines hiring needs, using various forecasting tools to determine how big each team needs to be in the future.

It maintains quality metrics to measure how well the organization as a whole is doing. It is responsible for the end-to-end quality. For example, it might have a metric that represents how many machines are delivered each week, how many are delivered on time, and how many are returned as defective. The program office is also responsible for monitoring the cost-efficiency of the program—for example, by tracking the average cost per machine (total budget of all teams divided by the number of machines delivered).

One-Time Projects Eventually Repeat

Projects such as introducing new hardware into an environment may be a rare event, but taking copious notes improves the process when it happens again. Recognizing that a seemingly one-time project is worthy of standardizing for future reuse is an important leadership skill.

Tom's team at Bell Labs developed techniques for renumbering and splitting IP networks for a once-in-a-lifetime corporate trivestiture. They were practically embarrassed to publish a paper about the techniques. When would such techniques ever be used again? Yet by the time the paper was published, they had used the techniques on more networks than were documented in the paper itself ([Limoncelli, Reingold, Narayan & Loura 1997](#)).

After the domestic terrorist attack at the 1992 Summer Olympics in Atlanta, new security coordination and administration techniques were

developed for the 1996 Olympics in Los Angeles. Its success led the Clinton administration to turn it into the standard security model used for all large events even today ([Clarke 2004](#)).

The truth is that while the same exact situation won't repeat, the processes and skills developed for that situation will be useful in the future. Success is the best advertisement. The success of one project will draw similar projects to you.

To help maintain a high standard of quality, the program office should encourage a process of continuous improvement. There is a human tendency for teams to become insular and stop talking. Soon they make optimizations that benefit themselves but hurt the overall process. This state is called siloing, because each team works in its own vertical silo of information. The program office should encourage activities that break down silos. For example, each team may have status meetings; delegates from the other teams should attend those meetings. There should be periodic cross-team reviews of plans and projects. The program office should make sure that there are channels for communication between teams. The solution may be as simple as assuring that each team uses a ticket system to solicit problem reports. The program office might also encourage cross-training. For example, if people from the platform team spend a week each year working on the prep team, they can learn which pain points exist and improve the overall flow of their installation automation.

10.3 Configuration Management Database

The **configuration management database** (CMDB) is a system that stores information about all machines in a fleet. The CMDB becomes the mechanism that binds all the previously mentioned teams together. If everyone's tools and applications work from the same database, everyone can work as a cohesive unit even though there are many different independent teams.

The CMDB stores information about a machine—its name, serial number, who uses it, which operating system is deployed in it, and so on. It also stores desired state; that is, settings might be stored in the database as an indicator of what is to be done, rather than just reflecting reality.

Initially, the CMDB may just be used as an information repository. In the prep room, as each machine is being installed, records are created in the CMDB, populating the database with the machine's serial number, model, OS type, and initial owner. However, the goal is to use the CMDB to drive better automation,

and to deliver a better customer experience.

A more sophisticated version of the machine installation process would start even earlier. During the purchasing process, placeholder entries are entered into the database to indicate that the machines have been ordered, along with company-assigned asset tag numbers. During a refresh program, a machine is allocated as a replacement for some other machine, and the desired configuration of the old machine in the CMDB is cloned onto the new machine. For new hires, the standard configuration for the department is cloned as soon as the new owner is identified. After the machines are manufactured, the vendor transmits serial numbers, MAC addresses, and other information to further populate the database. When the MAC address is inserted into the database, the configuration files for automated installation are generated and pushed to the boot servers. When the machines arrive, scanning the barcodes on the outside of the boxes marks the machines as arrived. During installation, machines receive the OS, applications, and other configuration settings defined in the CMDB and the associated delivery instructions are printed. When the customer receives his or her machine, the delivery is marked in the CMDB.

While the user uses the machine, if special applications are needed, the user visits an internal web site that provides a shopping cart–like experience for selecting them. For example, if Adobe Photoshop is selected, a license is allocated to this machine from a pool of bulk-purchased licenses. This information is recorded in the CMDB. An hourly configuration management program runs on the machine, sees that Photoshop is desired on this machine, and installs it.

If the customer contacts the helpdesk with a support issue, the technicians working there use the CMDB to look up the person's machine and see which hardware and software the person is using. Often the work involved in responding to the customer's request is accomplished by updates to the CMDB. For example, if the person needs to give a co-worker the ability to log into this machine, the technician updates the access controls in the CMDB.

The employee is able to access a web-based dashboard that lists which hardware and software is allocated to him or her. The CMDB is used to generate this display. Hardware that is more than three years old is highlighted as eligible for replacement and a link is provided that leads the customer through upgrade options.

Lastly, the CMDB is used by various departments to make long-term plans, as in the following examples:

- **Budgeting:** A major component of next year's budget is how many machines will need to be retired and replaced. A query of the CMDB determines which machines will hit their retirement age.
- **Upgrades:** The next release of a corporate application requires significantly more RAM. The exact machines that need to be upgraded can be determined and upgraded ahead of time using the CMDB. Otherwise, the roll-out would be a painful disaster.
- **Information security:** A security hole in a particular application has been discovered. The CMDB is used to determine whether this will be a major upgrade project or simply a matter of fixing a few machines.
- **Life cycle:** It is becoming very difficult to support a particular old model of hardware. The CMDB is used to determine where the remaining units are. The number is small enough that it is less expensive to replace those few machines than to expend the effort required to support them.

The CMDB is not just a simple inventory database. Inventory databases are often manually updated after the fact. In many companies they are updated yearly as part of a floor-by-floor audit. This means that inventory databases are, by definition, always outdated; they are a trailing indicator. A CMDB stays updated because the information is actively used. If, for example, access controls are determined by information in the CMDB, then inaccuracies are found and fixed quickly because accuracy determines people's ability to get work done. This is not to say that audits and inventory databases are useless. In fact, a configuration management system that runs periodically on machines often performs a hardware and software inventory, which is then uploaded to a separate inventory database. The two can subsequently be compared for auditing and compliance purposes.

Preventing Upgrade Disasters with a CMDB

A small private college in New Jersey outsourced most of its IT operations to a company known by SAs for providing poor service at high prices. The contract was based on billing for costs plus time, rather than a fixed amount for specific services. Unfortunately, this outsourcing model is common but offers the wrong financial incentive—efficiency reduces profits.

To maximize its revenues, rather than use a CMDB, the outsourcing provider manually inventoried machines. The cost of this manual process was paid for by the college, so there was no incentive to automate and

streamline the process. One year the college decided to upgrade its entire fleet to the new version of Microsoft Windows and MS Office. The outsourcing company provided a cost estimate that included only software licenses and installation labor costs.

One of the deans asked the chief information officer (CIO) if the existing machines could handle the new software. The CIO assured her that everything would be just fine. The dean then asked about the very old machines in computer labs in the basements of the dorms.

The CIO confessed that he didn't know there were labs in the basements of the dorms. Unfortunately, he still assumed that he and the outsourcing company knew more about the IT needs of the college than she did. After all, she was only a dean from one of the humanities schools, and IT was their business.

So the upgrades began and soon it was a disaster. Many machines did not have enough hard disk space for the new software, and the dorm computer labs were full of machines that weren't supported by the new OS.

The outsourcing company then offered to sell the university hardware upgrades and new machines. Because the summer was nearly over, there was no time for the university to put this contract out to bid. In addition, the outsourcing company charged this work at a higher hourly rate because it had to be completed in a short time frame.

A CMDB with up-to-date hardware and software configuration information would have saved the university a lot of money. The university would have known which equipment it had, and would have been able to generate reports on which hardware would need replacing and upgrading as soon as its leaders started discussing the project. Then they could have put it out to bid in plenty of time, and they would have found a better deal. In addition, an automated installation process that used the CMDB to generate system configuration information would have enabled rapid deployment of the new OS with low labor costs.

It is always wise to consider incentives and how they drive behavior. In this case, the outsourcing company earned more money by doing a poor job because it was a time and materials contract. If the contract had used unit costs for the support and upgrade of the machines, then the outsourcing company would maximize profits by minimizing labor costs, and a CMDB and automation would have been the result.

10.4 Small-Scale Fleet Logistics

So far we've discussed the many roles in a fleet logistics organization. But what if a company is too small to have a dedicated organization?

Whether you work at a company that deploys one machine a year or hundreds of machines per month, these roles still exist. However, in smaller organizations there will be fewer specialized teams. It is more likely that each person involved will be responsible for multiple roles.

10.4.1 Part-Time Fleet Management

An environment with a handful of machines will not need such a formal, systematic fleet logistics program. If new machines are acquired rarely, fleet logistics may be a tiny part of an SA's job. It may even be performed by a consultant who visits a few times a month.

If you are such a person, you wear many hats, fulfilling all the roles described in this chapter with ad hoc procedures invented in real time. Keep in mind which hat you are wearing at any given moment to help maintain your focus.

Work becomes easier when we reinvent the wheel less often. Keep good notes so that the ad hoc procedures can be reused. Keep these notes on a wiki or even a subfolder full of text files. As the organization grows, these procedures will evolve and grow into the large scale processes of the future. Here are some items to maintain on your wiki:

- **Vendor list:** For each vendor, record a bullet list of contact names and numbers. Maintain a history of each vendor, work done, commitments made, and so on.
- **Installation procedures:** Maintain a bullet list of steps taken to install and configure machines.
- **Delivery checklist:** List what you take with you when you deliver a new machine (tools, cables, spare parts, and so on), and which tests you perform.
- **Welcome letter:** Create a one-page document that you print out and leave with any new machine. It should include contact information for technical support.

Delivering a machine to someone's office may involve running back to your desk a few times because you forgot a cable, a screwdriver, or a mouse. By keeping a checklist of things to bring for each delivery, we learn from the past

and improve our process.

10.4.2 Full-Time Fleet Coordinators

As the company grows, the work of a fleet logistics team grows. It may become necessary to have a dedicated IT coordinator. This person will have a narrower focus and can formalize many of the processes previously done ad hoc. Such personnel may do all tasks themselves, or coordinate between different technical groups.

Another model that works well is having a two-person team: one technical person and one nontechnical person. They might share tasks that are in the middle, such as kicking off highly automated machine preparation processes or machine delivery.

This model succeeds when the two people clearly define who does what. This shared understanding helps them cooperate and not step on each other's toes. They make their decisions together, but each defers to the other in matters specific to the partner's expertise.

Another successful model involves a medium-size SA team who uses a dedicated fleet coordinator rather than expecting all SAs to spend part of their time doing machine prep and delivery. Customers receive better service because there is a central person dedicated to making sure their purchases are expertly coordinated.

The coordinator in this model is not an SA, but rather a power user. The coordinator focuses on running and improving the business processes, relying on the SA team for technical work such as maintaining the OS installation automation.

Dedicated fleet logistics may be appropriate only during growth years. Some companies grow to the point that this role transforms into a dedicated fleet logistics team. If growth stops, the person can scale down the operation and find another role.

Case Study: Fleet Coordinator and Chief Negotiator

One company experiencing rapid growth hired its first full-time fleet coordinator. This individual was somewhat technical but not a system administrator. He enjoyed working with the team, and the SAs appreciated not having to install machines any more. In a short amount of time the coordinator revamped the process, making many improvements. He was a particularly skillful negotiator, and negotiated deals with the vendors that

saved more money than his own salary. He also became the unofficial negotiating consultant for the IT department, accompanying co-workers to auto dealers and electronics stores.

10.5 Summary

This chapter showed us the strategies for how organizations handle the process of distributing workstations. What employees see is new workstations being delivered to people as they are hired, or on some periodic refresh schedule. Behind the scenes a lot of work is required to make this happen smoothly. This is a complex operation because it involves many teams: human resources, IT, purchasing, and more.

A large organization can mass-produce the process and benefit from economies of scale and division of labor. Some people prep the machines, others deliver them, others maintain the automation that makes the process work, and others manage the business and financial side of things. Organizations need project management to keep everything coordinated.

The configuration management database (CMDB) is used to coordinate these efforts, tracking a machine from arrival to decommissioning. It is also useful when making long-term plans, such as planning upgrades and other large projects.

Smaller organizations need a smaller effort. The entire process might be coordinated by single person.

Exercises

1. What are the elements of a fleet logistics operation?
2. Which parts of a fleet logistics operation are visible to the end user?
3. As a user, what would an ideal fleet operation program look like?
4. As an SA, what would an ideal fleet operation program look like?
5. Why is it less expensive to have a division of labor?
6. What are the benefits of a division of labor, when applied to workstation fleet logistics?
7. What is the difference between a project and a program? What does the program office do?
8. Compare and contrast how the configuration management database (CMDB) is used in this chapter and [Chapter 9](#), “[Workstation Service](#)”

[Definition.](#)”

9. Fleet logistics seems to be something that very large companies do. If you work at a smaller company, which elements are useful or what can be learned and applied to your organization?
10. Suppose a company has hired 10 new employees every Monday since it began. Each new employee receives a laptop that is replaced every 24 months. Assume no one ever leaves the company. How many laptops need to be installed on the first Monday of the company’s 18th, 24th, and 36th months?
11. Suppose a company has 100 employees today and plans on doubling its workforce every 12 months. Each new employee receives a laptop, and the laptop is replaced every 24 months. Assume no one ever leaves the company, and that today all existing laptops are less than 12 months old. How many laptops will need to be purchased each year for the next 5 years? Assuming laptops cost \$1,500 each, what should the annual budget for laptops be?
12. Repeat Exercise 11, but assume laptops are replaced every 36 months.
13. What are the budget differences between the 24-and 36-month policies in Exercises 11 and 12?

Chapter 11. Workstation Standardization

How does one take an organization from a mix of workstation configurations to a unified standard? This is surprisingly difficult. People resist change. Asking them to leave the comfort of their existing environment, no matter how terrible it is or how good the new environment will be, is asking for a lot of unearned trust.

In previous chapters we've discussed the benefits of having a standard workstation configuration. In theory we'd like a company to have a standard at the beginning so that it starts right and grows from there. Of course, that never happens. Instead, a company starts with no standard configuration and grows. Each new workstation is its own unique island, or a number of mini-standards, unofficial standards, ad hoc brilliant ideas, and common themes evolve out of the chaos. Eventually, there comes a day when the value of a standardized workstation configuration becomes apparent, and a project is created to define and transition to a standard configuration.

In your career you will experience this kind of standardization project frequently. Either you will join a company that is introducing a standard for its first time, or it has a standard but there is a new OS or technology that prompts the organization to move to a new standard.

We have seen such standardization efforts succeed and fail. From the successful ones we've learned these key points:

- **Involve customers early.** Involve customers early in the planning stages to gather requirements and to help identify the project's justification.
- **Release early and iterate.** Expose a small group to the new platform first, fix problems, and then repeat with larger and larger groups.
- **Have a transition interval.** Maintain the old and new systems at the same time. Do not expect to convert everyone at the same time.
- **Ratchet.** Move in only one direction, toward the goal. Adopt a strategy that prevents regressions.
- **Don't let it last forever.** Set a cut-off date at which point legacy machines lose access.

Adapt these key points based on the culture of the organization. A small company can make such transitions quickly; larger companies may be able to rely on corporate power hierarchies to push the change.

Use “the carrot” over “the stick.” The carrot is making the new environment

so awesome that customers are drawn to it. The stick is motivating people by punishment, threats, or management fiat. The carrot is more effective.

11.1 Involving Customers Early

The earlier customers are involved in the process, the more time we have to respond to their feedback. If customers are involved from the start, their feedback is incorporated into the design and testing of the new standard. If they are not involved, we receive feedback as the new standard is being rolled out and there is little or no time to implement their suggestions or fix the problems they find.

Prepare for Resistance

Since we know that people resist change, it is irrational to make plans that ignore this fact. Yet, we often see SA teams who begin to roll out a major change without warning customers and are shocked to meet resistance. Rationally speaking, we should take the time to do the things that prevent resistance. The end result is not just a smoother roll-out, but a higher-quality service. While it may seem like a burden to involve customers early in the process, it is less work to fix problems early. The burden we want to avoid is trying to fix problems after the launch, when we are under extreme time pressure and everyone is watching.

Months before the new configuration is rolled out, approvals and buy-in must be sought. The more people feel involved in the project, the more likely they are to go along with it.

Start by involving key customers in the design of the project. They may not have input on the technical details, but they'll help you recognize why the project is important to them, in their own language. This will help you explain the project to other customers.

Justifications for a project like this might include that everything will be easier to support, but that's not going to impress customers—that helps you, not them. To find the value to the customer, you must engage with them. Spend more time listening than speaking. They may like the fact that installation becomes self-service, meaning they no longer have to wait on someone from the IT department. They may like that they can now access their data more conveniently. They may like the color of the logo. If that's what makes people like the project, emphasize that!

11.2 Releasing Early and Iterating

Once you have a prototype of the new configuration, get feedback early. Let users try out the new configuration in a test lab or on a loaner machine for a week. They'll feel listened to and will be your partners instead of a roadblock.

Early Access to the New Standard

In one situation we set up a new standard PC in a place everyone could easily find it: near a public printer. We sat with people as they used the machine and kept notes about any problems they had. We resolved or found workarounds to any problems or concerns raised. Often we fixed problems in real time and were able to verify the fix worked. This helped make our customers confident that we really were focused on making the system work well for them.

11.3 Having a Transition Interval (Overlap)

The first few conversions will be difficult, as there will be unexpected bumps in the road. Given this reality, you should start with the people who are willing and eager to try the new system, warts and all.

At this point you shouldn't worry too much about the people who resist conversion. Your hands will be full with the initial group of early adopters. These early experiences will help work the bugs out of the system, making the conversions smoother in the future.

Once the new configuration is tested and approved, adopt it as the new standard and start using it for all new PCs.

Transitioning Late Adopters

As the kinks are worked out of the conversion process, more people will volunteer. In one case previously resistant people were exposed to the new standard because they purchased a new machine. Soon they were requesting their old, legacy machines be converted. In other cases customers saw that others were happy with the new configuration and didn't want to be left behind. No one wants to be the only kid on the block without the new toy that the cool kids have.

Eventually the easy conversions were finished and we had the time and energy to work on the long tail of late adopters. Some of these people had verified technical issues such as software that didn't work on the new

system. Each of these problems could require many hours or days to make the software work or find replacement products. In some cases, we'd extract the disk of an old physical machine and convert it into a virtual machine, a process called p2v. This enabled the customer to run the old software—in fact, the old computer—in a window on the new machine.

This last approach should be used only as a stop-gap measure until the application can be supported on the new configuration or replaced by something that is. This should be made clear to the end users, and an expiration date agreed upon. The old OS will no longer get security patches applied and therefore is a risk.

11.4 Ratcheting

Ratcheting means adopting a policy that all new new machines use the new standard, no exceptions. A ratchet is a mechanical device that allows movement in only one direction. Like a ratchet, you want the conversion process to move in only one direction: toward the goal. With this model, all new machines use the new standard; no new machines may use the old standard. The number of PCs using the old standard should only decrease and the percentage of PCs using the new standard should only increase.

Implementing this rule means dedicating yourself to preventing any regressions or back-sliding. Back-sliding means installing PCs with the old configuration out of habit, by mistake, or by granting an exception. This can be a large, difficult effort. The IT staff has to be vigilant. Management has to have a backbone. Everyone has to police each other and the users. It also helps if the old way of installing machines happens to stop working and no one can remember how to fix it.

A request for an exception, however, should not be greeted with a loud “no” but by helping the person get what he or she needs using the new standard. Interview the person about why the exception was requested. By listening and having empathy, we are able to get people onto the new standard.

If ratcheting is done right, eventually the conversion will take care of itself as all old PCs are thrown away and replaced with new ones. However, such conversion through attrition could take years, which would be much too slow. Therefore once the new standard has proven itself and gained popularity, it is time to require old machines to be converted.

11.5 Setting a Cut-Off Date

Eventually the last few hold-outs must be converted. Supporting the legacy environment has a cost. This cost is amortized over all the remaining legacy machines. As the number of machines shrinks, the cost per machine grows. In the end, that entire cost is the burden of a single machine. That last machine is very expensive—so the faster we can eliminate the last legacy machine, the better.

To motivate the last few stragglers, announce that unconverted machines will be unsupported after a certain date, and they will lose access to the file servers and other systems a month later. You need management buy-in for those dates plus a commitment that no exceptions will be made. Communicate the dates broadly at first. Later, send warnings only to the specific people affected plus their managers.

In organizations that use 802.1x access controls on their network, you can simply exclude machines with the old configuration from the network.

Do not entertain the “I’ll just support it myself” argument. The old configuration doesn’t automatically receive security patches and therefore is a security problem waiting to happen. Eventually the last machine will be converted and the project is done.

11.6 Adapting for Your Corporate Culture

Conversion projects like this pop up all the time. Sometimes they are undertaken to standardize on a platform. Other times they are due to a smaller company being bought by a larger company.

If tasked with a similar conversion project, you’ll need to adjust the techniques you use depending on many factors, especially company size and corporate culture. Some companies may have a culture where “more stick, less carrot” permits the process to move faster.

Sometimes it is easier to simply link the conversion to a major operating system release. A major new Microsoft Windows release is a good excuse to start a standard. Use the early-access versions to develop the standard and test it with a small group of users. The moment the final release is available, insert it into the system. Now you can begin ratcheting and soon the entire organization will be converted.

These techniques are important whether you are converting to a new workstation standard, introducing a new server hardware platform, migrating users to a new email system, or adopting a new application. Involve the customer early, test with small groups before larger roll-outs begin, ratchet, and

eventually set hard deadlines.

11.7 Leveraging the Path of Least Resistance

Frequently in IT, the best way to get the outcome that you are aiming for is to make it the path of least resistance. If the easiest thing to do is to get the standard workstation, and to use automated systems for rapidly acquiring additional applications, then most people will go that route without much prodding. In the end, everyone just wants to get on with the work they are meant to be doing.

Case Study: A Tale of Two Standardizations

In the 1990s Tom was part of the IT support team in a research department at Bell Labs. Introducing a standardized PC was received well by some departments, but one in particular resisted.

The department director justified his resistance: He felt that since he hired world-class computer scientists, they should have whatever desktops they wanted. While we couldn't disagree with that reasoning, we felt we could achieve the same goal while providing a better experience.

The actual new-hire experience looked like this: On the researcher's first day he or she was handed the paper catalog from Sun Microsystems and Dell, the leading workstation makers at the time, and the phone number of the salespeople who were assigned to our account. (This was before the web was as ubiquitous as it is today.) The researcher would spend a week poring over the catalog, selecting their desired model and accessories. Then the new hire would call the salesperson, who would walk them through the process starting over from scratch—this time picking parts that were actually compatible. A few weeks later the items would arrive, and within a week or so the workstation would be assembled and installed by the IT group.

Typically new employees did not have a functioning computer for their first month, sometimes two. During this time, the researchers couldn't get much done. In fact, they had to suffer the humiliation of begging coworkers to borrow their workstations to check email. This was a terrible experience and hardly appropriate for world-class researchers.

The director wouldn't budge from his stance. Finally, one of Tom's coworkers negotiated a brilliant solution. Seeing that the ordering process was transitioning to the web, and that email and web access was no longer a once-a-day thing, he offered to install a loaner PC at each new

employee's desk for up to one month. This machine was to be used to order the world-class computer the researcher so richly deserved. After a month the loaner could be returned to the IT department. If it was kept, the department would be appropriately billed.

The director agreed, as this met all his requirements. New researchers were very happy to find a fully functioning PC on their desk on their first day. As they realized that a bird in the hand is worth two in the bush, none of these loaner workstations was ever returned.

The new process not only saw new hires becoming productive employees sooner, but also saved considerable money. When left to their own purchasing decisions, customers opted for features and add-ons that went unused. There were concerns that the standardized PC would have a cheap video card or save money some other way that would lead to an inferior workstation. The opposite was true: We were able to provide a better video card and other components for the same price due to the volume discounts we were able to negotiate.

Fast-forward nearly 20 years, and Tom found himself working at Stack Overflow, where the CEO felt that it was important that “you get the tools you need to get your job done.” This translated into a policy where developers requested any workstation configuration they wanted. There was no standard. New hires who were hardware buffs enjoyed spec'ing out every detail of their dream machine. Others, however, found this process confusing and frustrating. Unlike at Bell Labs, however, new hires were not without a computer for their first month because the IT team would reach out to new hires in advance.

This policy created a support nightmare for the IT team. Every user had a slightly different machine, and there were a wide variety of operating systems to support. It also meant that the onboarding process could get stalled by hardware problems. For example, a new hire, working remotely, received the wrong kind of memory for the motherboard that had been ordered. Without a computer for his first week, onboarding required various workarounds. The policy also created an information deficit: New employees, by definition, are ill informed as to which tools they need for a job because they haven't started yet.

To improve the situation but still abide by the CEO's directive, the IT team created a list of suggested hardware. New employees would be encouraged (but not forced) to select a standard configuration, while at

their hardware refresh date (every two years) they could choose between the current standard or select a fully customized dream machine. The past policy created an expectation that made it politically untenable to make the standard model mandatory.

The workstation standard was developed in collaboration with existing employees, who verified that it met the needs of the roles within the company. The standard configuration was a Mac laptop (choice between two models) with the maximum amount of RAM and SSD storage available, plus an external monitor, headphones, and other accessories.

The new policy solved many problems. The onboarding process was more standardized and trouble-free. New employees could choose to receive a machine that had everything they needed to get their job done. More experienced employees had the option to build their own machine. Everyone benefited from the fact that the IT team could provide better support.

Economists call this the default effect. People tend to select the default option especially when they are new to a situation or the choices are confusing. If you want to encourage a particular choice, make it the choice that requires little or no action.

11.8 Summary

Introducing a standard workstation configuration into an organization that has never had one before is quite difficult. There is likely to be a surprising amount of resistance, even if the new standard is technologically superior.

There are five key factors to successfully converting a fleet of workstations to a standard, or even upgrading to a new standard.

Since we know there will be resistance, it is logical to prepare for it. It is irrational to assume there will be no resistance; your organization isn't special. Therefore, involve customers early in the planning stages. Involve customers in formulating the initial justification, the design, and the roll-out.

Release early and iterate. Expose customers to the platform early, fix the problems they find, and then repeat with larger and larger groups until the roll-out is complete.

Have a transition interval. Maintain the old and new systems at the same time. Having everyone switch to the new system at once destroys your ability to learn from early iterations and overloads any resources involved in the conversion

process itself.

Use ratcheting: Permit only forward progress. The percentage complete should only grow. Once the standard is stabilized and approved by all, disable the ability to create systems using the old standard.

Set a cut-off date. Deadlines are motivating. Having a deadline enforced at the network level, such as disconnecting service from unconverted hosts, is the only way for such deadlines to have teeth.

The advice in this chapter must be adapted to your own company's corporate culture. There may be unique politics, policies, and scale differences.

Often it is easier to link standardizations to major OS release deployments. For example, the only way to receive Windows 10 might be via an installation process that deploys the new standard.

Finally, the power of defaults should be leveraged to make the easiest thing someone could do provide the result you want to see. For example, if opting into the standard hardware platform requires zero paperwork or effort, magically appears on an employee's desk on his or her first day with the company, and is silently billed as the result of not doing anything, all other options suddenly feel like burdensome work and will be avoided.

Exercises

1. Describe the five elements that lead to successful conversion projects.
2. How does early customer involvement help justify the project?
3. How does early customer involvement help create a higher quality standard?
4. What is meant by release early and iterate? Wouldn't it be better to release a standard that is perfect on the first try?
5. What is a transition interval? How long should it be?
6. What is ratcheting and how does it work?
7. Why is having a cut-off date important?
8. Why would people resist a standard that is technically superior to what they currently use?
9. Which ways are the default options used in your fleet environment?
10. Select a standardization or conversion project you've been involved with. Explain how the five elements were included or omitted and how that affected the project.

Chapter 12. Onboarding

This chapter is about onboarding new employees, how the IT department fits into this process, and how to make the process work best.

Onboarding is the process by which new employees are brought into the company. It starts when they accept their job offer and ends when they are settled in and productive. Onboarding includes providing new employees with their desk location, computer, phone, accounts, and anything else they should have to start their job. Onboarding is the responsibility of the human resources department but much of the process cannot be successful without the involvement of the IT department. A good process requires interdepartmental cooperation.

Joining a company is a major life event. It is a big risk and can involve up-ending one's life and family to move to a new city. It is scary and stressful. We owe it to our new employees to have a good onboarding process. Everything you do to make their transition smooth reassures them that they've made the right decision. A messy or chaotic onboarding experience adds to their stress, indicating that maybe they've made the wrong decision. Arrival of new employees should be a soft landing, not a crash.

As the number of people being onboarded increases, we have opportunities to improve the overall efficiency of the onboarding process. For example, most companies start new employees only on a particular day of the week so that they can efficiently process people in batches. It is better to give the new employee orientation lecture to a large group of people each Monday than to individuals each day of the week. The new employees appreciate the efficiency. They just want to get through all the startup tasks so they can get to work.

12.1 Making a Good First Impression

A good onboarding experience is key to making a good first impression. It is difficult to repair a relationship that starts off on the wrong foot. If it starts out well, future mistakes are more easily forgiven.

Furthermore, the onboarding experience is an opportunity for your company to set expectations for all new employees. How the onboarding process is conducted sends a message about how they are expected to behave. If what they experience is friendly, happy people who are helpful and courteous, it sets the expectation that they should be helpful and courteous. If they are exposed to a

process that involves multiple departments exhibiting disdain for each other, the message is clear that an adversarial relationship between departments is permitted.

The onboarding process at a small startup often involves an ill-defined process full of ad hoc and improvised processes. The message that this sends is that the company is new and just figuring things out, which reflects the truth about every startup. The onboarding process at Google is so sophisticated and well run that it sends a clear message to new hires: You've just joined a company that can do complex things at scale very well; you'll be expected to do the same. When Tom joined Bell Labs in 1994, each onboarding step was done by a person who didn't seem to have a clearly defined process but was muddling through by copying what he or she could remember was done for the previous person—which seems to be how many things were done back then.

One way to measure the success of your onboarding process is by how soon certain milestones will be reached. For example, how soon will new employees be able to send email from their own machine, using their own account? Every IT department should be able to achieve this milestone by the end of the first day. In the best case, the email they send is to a loved one saying that they've arrived and their onboarding experience was awesome.

Another milestone is how soon new employees can make useful contributions to their team. For developers this means the ability to write code that gets into production. For salespeople it means making their first sale. Since most positions require training before they can be productive, the IT department's influence may be limited to setting up prerequisites for such training.

Etsy's Onboarding

Etsy, Inc. has a high bar for onboarding. As part of every employee's introduction to the company, they write code and push it to the production web site. Even nontechnical staff are led through the process by which their name and photo are added to the company web site. They edit code, push it to the source code repository, watch automated triggers detect the change, run it through a battery of automated tests, then deploy it to the live web site. For this to work, all the new hire's accounts, access permissions, and tools must be set up and working. As a result, the company is able to meet its goal of instilling a key part of corporate culture: fully automated testing and deployment as a way of doing business.

More details can be found in John Goulah's blog post, "Making It Virtually Easy to Deploy on Day One" ([Meggs 2012](#)).

12.2 IT Responsibilities

The IT-related responsibilities include setting up the person's accounts, workstation, network connections, desk and mobile phones, security badges, and any other technology-related issues. In a small company all of these tasks may be done by the IT team. In a larger company they are divided among appropriate departments: the IT team, the network group, the physical security, and so on.

So much of onboarding involves technology that if the process goes badly, the IT department often receives the blame. For example, if HR arranged for the wrong kind of workstation to be delivered, people will assume the error was with the IT department.

You have only one chance to make a good first impression on your customers. A good first impression creates goodwill between you and your users. Goodwill is like a bank account. If you start out with a positive balance, you can make a few mistakes and stay in the black. If you start out in debt, it may be impossible to crawl out of that bad situation.

Because of the importance of a good first impression, some IT departments drive the onboarding process. Their reputation is too important to leave to someone else.

If your company's onboarding process is messy, disorganized, and error-prone, and leaves new employees wondering if the company they've just joined is one pair of oversized shoes short of a clown car, then it is your professional responsibility to spearhead an initiative to work with HR and fix the onboarding process.

12.3 Five Keys to Successful Onboarding

There are five keys to assuring a well-organized onboarding experience:

- Drive the process by a timeline of onboarding actions.
- Determine the person's needs ahead of his or her arrival.
- Perform each onboarding action by its deadline.
- Encourage cross-team communication about issues as they arise.
- Periodically reflect on the process and improve it.

12.3.1 Drive the Process with an Onboarding Timeline

Onboarding involves many different people and departments. There are at least four people: the new hire, the hiring manager, someone from HR, and you. There needs to be a way to make sure everyone agrees about what needs to be done and can track the work as it progresses.

The simplest solution is a timeline of onboarding tasks. This is usually a checklist of when various tasks are started, or when they must be completed. Once this timeline is developed, it can be used and revised again and again. Each task on the timeline should have a time or event that triggers it, and the person responsible for completing the task should be designated.

Before you can create the timeline you need to list the requirements, or end results, of onboarding. For example:

- Onboarding paperwork completed
- Office/desk allocated
- Accounts created for ActiveDirectory and email
- Developer-related accounts created (engineers only)
- Desktop workstation set up (engineers only)
- Laptop workstation set up (all employees)
- Desk phone set up
- Business cards ordered

Once everyone has agreed that these are the requirements, we list out the dependencies for each:

- Onboarding paperwork completed:
 - Paperwork is sent to new hires after they sign the offer letter.
- Office/desk allocated:
 - Space planner receives request. Allocates space.
- Accounts created for ActiveDirectory and email:
 - New hires are asked for their preferred usernames.
- Developer-related accounts created (engineers only):
 - Source code repo account created.
 - Permissions set up in dev database.
- Desktop workstation set up (engineers only):
 - New hires are asked if they require Linux or Windows installed on their

workstations.

- Laptop workstation set up (all employees):
 - New hires are asked if they prefer an Apple MacBook or a Dell Latitude laptop.
- Desk phone set up:
 - Request put in to telecomm department.
- Business cards ordered:
 - Preferred name to be listed on business card is confirmed, as well as any social media IDs.
 - Telecomm department has confirmed phone number.
 - Email account has been created.

As we discuss this list, we realize that we will want the computer accounts to be created but disabled until a certain time or event, based on security policy. For example, the accounts may be enabled after the onboarding paperwork has been completed, or perhaps just the day before the employee starts work. We adjust the task list to include this step.

Now that the dependencies are known, we can turn this into a timeline. Consider the lead time for each item and its dependencies. Each task should list the person or department responsible. Additional items are added as the entire experience is designed. For example, we've added a visit on the person's first day by someone from the IT department to answer any questions and assure the new employee is set up as needed. The IT department also visits a week later to answer any follow-up questions.

Each section starts with when the tasks start. We've listed the team responsible in parentheses at the start of each item. Also note the use of short, pithy phrases rather than complete sentences. This is a checklist, not an essay. The result looks like this:

- When: Offer letter is signed
 - (HR) Create row in onboarding shared spreadsheet for the new hire. Fill in name, start date, and engineer/non-engineer status.
 - (HR) Onboarding paperwork sent to new hire.
 - (HR) (engineers only) Open ticket with purchasing to request desktop workstation.
 - (HR) Request preferred username. Use HR dashboard to reserve it. Open ticket with IT; request account creation.

- (HR) Request laptop preference from new hire. Enter response in onboarding spreadsheet. Open ticket with purchasing to request laptop.
- (HR) Request desk assignment with space planning team.
- When: 2 weeks prior (or sooner)
 - (Purchasing) Engineering desktop ordered (same hardware runs either OS).
 - (Purchasing) Laptop ordered.
 - (Facilities) Office/desk location entered in onboarding spreadsheet.
- When: 1 week prior (or sooner)
 - (HR) Business cards ordered (first: verify phone number rings when called).
 - (IT) (engineers only) Desktop prepped and delivered.
 - (IT) Laptop prepped and delivered.
 - (IT) Accounts created (but disabled) using information from the spreadsheet.
- When: Day before arrival
 - (hiring manager) Verify everything is ready. Open high-priority ticket if any of these are not complete: Laptop delivered. Accounts created. Engineering desktop delivered (if engineer).
- When: Arrival day
 - (HR) Greet new hire.
 - (HR) Verify onboarding paperwork complete. Use HR dashboard to enable accounts.
 - (HR) Conduct new employee orientation.
 - (HR) Deliver new hire to hiring manager's office.
 - (Hiring manager) Verify new hire can log into laptop and access email.
 - (IT) 3 PM: Visit new hire to answer any IT questions. Deliver one-page FAQ.
- When: 1 week after arrival
 - (IT) Visit new hire to answer any IT questions.

Now you have an organized list of action items. This will be used to drive all other processes. As each new hire joins the company, use this as your checklist or action plan to bring that person onboard successfully.

12.3.2 Determine Needs Ahead of Arrival

Most of the timeline tasks need to be done ahead of the new hire's arrival, so we need to know that someone is arriving as early as possible. For example, to create new employees' accounts, we need to know their preferred username (and if it isn't available, what their second and third choices are). We need to know what kind of workstation and software they require. Determining this may require knowing their role (e.g., mobile sales requires a laptop, engineering needs a desktop), or having enough time to ask them their preferences and order the model they need.

Being notified that someone is arriving kicks off the timeline for that person. Commonly IT departments are not informed of new hires until the last minute or, in the worst case, after the person has arrived. This leads to chaos and confusion, and a bad first impression. If this is the case in your organization, your first step must be to fix this problem.

Improving Onboarding at Bell Labs

For a while the onboarding process at Bell Labs left much to be desired. Often the IT team was not warned of a new employee's arrival until the person's first day. The IT team would scramble to arrange for the person to have an email account and a new machine. It was an embarrassment. The IT team took the initiative to fix the process.

Creating a more orderly process was impossible without knowing when new employees would arrive. The HR department refused to provide this information, claiming it to be confidential information—an assertion that to this day Tom does not believe. The lab directors were unwilling to provide this information, claiming that they often hire people on little notice—an assertion that Tom also doubted, considering that Bell Labs couldn't do anything without weeks of approvals.

A better plan was needed.

One of Tom's co-workers pointed out that the administrative assistants know more about what's going on in the company than anyone else. He hatched a plan to recruit them as spies in the name of a better onboarding experience.

Soon one administrative assistant had been recruited and agreed to give IT a heads-up about any new hires in her department. In exchange she received a new PC, which happened to be much fancier than any other PC in the assistant pool. It wasn't a bribe; it was simply a benefit of being

part of the unofficial onboarding improvement committee (UOIC). Soon the other assistants were jealous and approached IT about joining the committee.

Eventually all department assistants were members of the committee and the onboarding process could be reengineered.

12.3.3 Perform the Onboarding

The timeline is full of tasks that are triggered either by a specific date or by the completion of other prerequisites. Who will do each step must be determined, and then the work can begin.

A small site might have one person doing all the tasks, or a coordinator who delegates the tasks. For example, the office manager might have a checklist that he works through for each new hire. Each task is delegated, with the office manager being the central point that all information flows through. There are benefits to having one person coordinate a new hire's end-to-end process. It is more efficient because the one person retains knowledge along the way. Each additional person involved in a process doubles the number of potential communication paths, and increases the possibility of miscommunication. However, a single person doesn't scale. It is rare that one person has the skills or access to perform every step of the onboarding process.

At a larger company, communication is done through creating tickets in request-tracking systems. Tasks are assigned to appropriate groups, and then the groups receiving the tasks assign individual resources. Alternatively, a shared spreadsheet such as Google Sheets may be used as a central repository for information about each new hire. One row per new hire lists the new employee's name, start date, job role, preferences, and so on. Alternatively, a Kanban board such as Trello can be used to track the progress. Having all information about a new hire in one place reduces confusion as well as the amount of back-and-forth communication. People can look at, for example, the shared spreadsheet, instead of interrupting someone to ask for information. This level of transparency saves everyone time.

At a much larger company, custom software manages the entire process. It both provides a central information repository and tracks workflow and progress.

Often there is a single timeline. Sometimes there is a master timeline with individual timelines for each stakeholder. For example, the master timeline might list that IT is informed of the arrival two weeks prior to the new employee's start date. IT then has its own timeline of things to do during those

two weeks.

12.3.4 Communicate Across Teams

Onboarding involves many teams, and as the onboarding continues there needs to be a way for subteams to communicate issues that arise. There are also some fixed points where interteam communication is needed. For example

- To request that a task be done
- To notify other teams when a task is complete
- To ask questions or for clarifications
- To report problems or roadblocks so they may be addressed

The onboarding process should define how such things are to be communicated.

If one person coordinates the entire process, that person may be the clearing-house for all communication. She assigns tasks, and knows how each subteam prefers to be contacted and how to format the request. She is notified when tasks are complete. Seeing the prerequisites complete, she kicks off moving forward with the next task. Process questions, problem reports, and roadblocks are communicated to the coordinator, who either resolves the issue or assembles the right people to do so.

If there is no one central coordination point, then each team needs to know how to communicate with any other team with whom they interact. The most streamlined system is for each team to communicate the same way. For example, each team might have a queue in a request-tracking system. A simple wiki page of contacts and request queues is sufficient to ensure that each team knows how to contact the other teams. Do not assume that everyone magically knows the right person or people to talk to when an issue arises. Not everyone is as “in the know” as you are.

Often the process can be simplified by eliminating a lot of communication. For example, if a process is sufficiently automated, the coordinator becomes more self-sufficient. She can view the automation’s dashboard to determine the status of tasks without having to ask people. She can use the automation’s portal or API to request that a task be done. The only need for human interaction is if there is a bug or other problem. An example of this level of automation is described in [Section 4.6](#), “Message Bus Architectures,” in Volume 2 of this book series. It is natural that problems and roadblocks will arise. Not planning how to handle them is equivalent to assuming that they will never happen, which is irrational. For example, the software used to automatically prepare a new computer might stop working due to a bug, or because the vendor changed a

small but vital element of the hardware. Or, an unexpected situation might arise, such as a delayed shipment of hardware. Any such issues need to be communicated to the other teams.

Any step that relies on software should have a clear way to file bugs and feature requests. People are really good at finding workarounds for minor problems. However, if these minor problems go unreported, they do not get fixed, and they turn into bigger problems. People do not feel the need to report minor problems because they assume everyone knows about them already, or they don't want to be a bother, or they don't see how it affects the bigger picture, or they feel heroic by muddling through the issue, suffering in silence. We saw this previously in [Section 8.4](#)'s anecdote "[Make Brokenness Visible](#)."

Encourage a culture of raising the visibility of problems small and large. Encourage people to communicate ideas for improvements. The people doing the work are the best experts in what's wrong with a process. Give them the tools to share what they discover so that developers and people working at higher levels can make the improvements.

12.3.5 Reflect On and Improve the Process

There needs to be a way for all involved to come together, reflect on the process as it is, and discuss ways to make it better. This reflection can generate some of the biggest, systemic improvements as well as valuable micro-improvements. However, this happens only if space is created to let it happen.

The team may have a weekly status meeting that works through current roadblocks. On the one hand, this is a time when people are often too busy to pause and look at the big picture. On the other hand, carving out some time from every meeting to reflect on the broader issues is better than nothing.

It is better to establish periodic meetings to focus on the big picture. For example, a quarterly meeting might be scheduled where the process is examined and areas of improvement are identified. One way to do this is to walk through the end-to-end process, perhaps on the whiteboard, with each team explaining what they do at each step, what their pain-points are, and what they would like to see improved.

Rather than using a whiteboard, a live exercise involves walking through the actual process, doing each step for real. This may be the only chance the entire team has to observe each step, which is otherwise hidden from view. You'll be surprised at how many "aha" moments happen in this kind of cross-team exercise.

Often these meetings are scheduled at the end of a major milestone, such as after a new group of employees have started. In this case, the goal is to record what happened and determine areas for improvement while everything is fresh in people's minds. If this knowledge is not captured immediately, it may be forgotten by the next planning cycle.

The issues raised will come from many sources. Sometimes external forces bring the need for change. For example, sudden growth may result in the need to switch to a more batch-oriented strategy. Internal forces may arise, such as feature improvements or changes in the software being used. Often the issues that arise are simply the recognition of existing problems that were easy to ignore at first, but once the system was running they became death by a thousand cuts.

Some of the changes will be major, others minor. Major changes might include an increase in volume that suggests it would be better to process onboarding in batches or groups of people instead of one at a time. As a result, you could have all new hires start on Monday. Some changes will be minor but much appreciated by the people doing the work. For example, something as simple as sorting a printout or web page alphabetically for one team, and sorting it by floor and room number for another, might be a small change that means a lot to the people doing the task. Similarly, a user interface that seemed reasonable when it was new might now feel like 15 error-prone clicks per item. Reducing it to 1 click for the common case doesn't just save time, but also makes the process more bearable.

Lastly, it is important that the team be willing to try new things and roll them back if they are not a success. In some organizations it is so painful to change a process that if the change turns out not to be the improvement that was envisioned, it is equally painful to revert to the old process. If the culture of the company punishes this failure, people will be less likely to suggest changes in the future and the company will miss out on the good changes. A better attitude is to treat changes as experiments, measure the corresponding improvements, and have a decision point where the change is either made permanent or rolled back. If it is rolled back, it is not looked at as a mistake or failure, but examined to see what can be learned. In that case, even a "failure" is a positive thing because it informs future decisions. An organization that is always experimenting and learning is one that is always able to innovate.

The Three Ways of Operational Improvement

The DevOps movement talks about the Three Ways, which are categories

of operational improvements: the workflow of the process, the communication between the processes, and the ability to experiment and improve the process in general.

The First Way is a focus on the process itself, making sure it is consistent and repeatable. If you picture the process timeline, it is a series of steps that flow from left to right. The process should look more like an arrow pointing forward than a pile of spaghetti. No steps should be ill defined or done differently by different people. Fixing these issues results in a process that is a smooth and clear left-to-right flow.

The Second Way is a focus on the flow of information. Feedback needs to flow upstream, from right to left. This pathway enables empathy between teams, and is a requirement for problems to be solved and improvements to be made. Without it, problems accumulate and are never fixed.

The Third Way is a focus on experimentation and learning. When we bring all the teams together and reflect on the process it generates ideas for improvements. A good corporate culture encourages trying new things. Without trying new things, there is no innovation. The ability to innovate also requires permission to fail. We can only truly know if a proposed change is good or not by trying it. That means we must be willing to try new things, even if some will fail. We will learn from these failures, however, and they will inform future attempts to improve the process. Therefore a high-performing organization has a culture that encourages experimentation and learning.

12.4 Cadence Changes

The number of new employees starting each week changes over time. Sometimes a company announces a new strategy that requires a sudden flood of new hires. Other times hiring is frozen. Sometimes the cadence, or pace of work, for the onboarding team simply grows over time organically. Suddenly you notice the onboarding structure of the quiet past is inadequate to meet the demands of the stormy present.

No matter the reason for the change, someone needs to recognize it and start a process of reflection to determine which changes are needed given the new reality. Often this begins with smaller discussions between the person or subteam most affected and the program coordinator. Eventually it may lead to the coordinator calling for a larger meeting of all stakeholders.

Growth requires streamlining and turning the process into more of an assembly-line process. Institute a division of labor where previously one person did many tasks. When possible deal with tasks in batches instead of one at a time. For example, have all new hires start on Monday so that all hiring efforts are concentrated once per week, rather than spread out. Giving the orientation lecture to one person takes about as much time as doing it for a room of ten. Therefore it is better to start all ten new hires on Monday than two new employees each day of the week.

Growth also benefits from automation. It can be helpful to create software tools that improve one particular task by making it less error prone or faster to do. Even better is software that eliminates human involvement, such as creating accounts automatically when new people appear in the human resources database, rather than waiting for a request to create the accounts. Software takes time to develop, so plan ahead.

Decreased hiring or stagnation requires reorganizing, too. Batching may also help, but the start dates may come less often: monthly, biweekly, or ad hoc as the opportunity presents itself. It may be possible to consolidate many tasks to fewer people. For example, a person doesn't have to be a technical genius to hand out laptops if they are prepared ahead of time. In that case a checklist created jointly by the IT department and the person managing the onboarding process may be sufficient.

12.5 Case Studies

Over our careers we experience a mixture of good and bad onboarding processes. Seeing the onboarding process from the new employee's perspective can give us some ideals to aspire to, and some experiences show us what we should strive to avoid. Here are some accounts of good and bad onboarding processes.

12.5.1 Worst Onboarding Experience Ever

Gina was an experienced software engineer who joined a major insurance company, one you have surely heard of. After the first week she still didn't have a computer, but her manager insisted this was normal and she should attend all of the team's meetings and spend her time reading the system architecture documents that would bring her up-to-date with the team's projects. She received her computer during her second week but it had no software. It was locked down to prevent her from installing software herself. Therefore she didn't have the tools she needed. She was informed that all software had to be installed by the IT

team. Her requests for the software were delayed because the developer tools weren't on the standard list and had to be special ordered. This, of course, is odd because everyone else on her team used these tools.

To make matters worse, her desk was in a different building than her team. She was in the new building to which the rest of her team would be moving soon. Facilities didn't want to do an extra move.

In summary, Gina suffered the indignity of not having the tools to do her job and interacting with her team by roaming from desk to desk like a nomad. Unable to do any work, Gina spent most of her day bored, reading books she brought from home. Her boss was surprisingly okay with this.

Eventually Gina had caught up on all the novels she had in her backlog, plus had reread all of her favorite novels. After three months she still did not have a working development system and she quit.

12.5.2 Lumeta's Onboarding Process

Lumeta was a small startup with about 40 employees. The onboarding process began a few weeks before the new hire's first day.

The office manager had a timeline-based checklist of tasks. The IT person was informed of the person's arrival as soon as possible, often even if the candidate hadn't accepted the offer yet. The hiring rate was slow enough that it didn't make sense to pre-purchase a pool of machines for later distribution, but the company used a vendor that could deliver its standard laptop in less than a week.

The office manager had a series of questions she asked all future employees. Most of these questions came from the IT manager and gathered information such as their preferred username, type of machine, and so on.

On their first day, new employees were met by the hiring manager, who would welcome them and walk them to HR for paperwork, then take them to their desk where their PC and phone were set up ahead of time. The IT manager would visit to walk them through logging in and changing their password, and give them a simple one-page FAQ that listed the URL of online documentation and instructions on how to open support requests in the helpdesk system.

The IT manager would answer any questions then, and also return a week later to check up on the new employee. On new hires' first day they received so much information it was like drinking from a firehose. Therefore the follow-up visit gave people a chance to ask questions when they were feeling less overloaded. By then they had had a chance to settle in enough to develop some questions.

While the office manager had a timeline-based checklist, the IT department

maintained its own checklist. It was more detailed and included step-by-step instructions for each task. While most of the process was fairly automated, the checklist included items that had to be manual, such as verifying that the mouse worked, and personally visiting the person a week later. The checklist assured consistency and made sure steps weren't skipped.

Installation Checklists

Whether your OS installation is completely manual or fully automated, you can improve consistency and completeness with a checklist.

The usefulness of such a checklist is obvious if installation is completely manual. There are too many steps to remember. However, even if OS installation is completely automated, a good checklist is still useful. Certain things can't be automated because they are physical acts, such as starting the installation, making sure that the mouse works, cleaning the screen before it is delivered, and giving the user a choice of mousepads. Other related tasks may be on your checklist: updating inventory lists, ordering more network cables if you are below a certain limit, and checking in on a new customer a week later.

Even a solo system administrator who feels that all OS loads are consistent because he does them himself will find benefits from using a written checklist. If nothing else, your checklists can be the basis of training a new system administrator or freeing up your time by training a trustworthy clerk to follow your checklists.

Like it or not, we forget things. Our brains are imperfect. That's often difficult to accept, considering that SAs are usually hired for our awesome brains, not our good looks. But think of it this way: Pretend you haven't done the task for six months and have started to forget some of the steps. The list needs to be just detailed enough to help you in that situation. Don't write the checklist for you. Write it for "six months from now" you, or "I didn't get a lot of sleep" you.

12.5.3 Google's Onboarding Process

Google has a sophisticated onboarding process. In its larger offices dozens of people start on the same day. No matter your level and position, you go through the same process.

All Google employees start on Monday. All teams involved in the onboarding

process schedule their work around this weekly cadence. Teams that are solely focused on onboarding, such as the workstation delivery team, have one deadline each week rather than five, which makes it easier to do work in batches. Teams that have multiple responsibilities need to focus on onboarding only once a week. For example, a 30-minute segment of the new employee orientation presentation is given by someone from IT security. This team needs to find a volunteer to give this talk just once a week, rather than every day.

Start-Day Friendships

Starting all new employees on Monday doesn't just improve batching; it also helps create friendships. Tom started on the same day as dozens of others and made friendships that lasted many years even though the other employees were in different teams, and eventually different time zones. The batch was about five times larger than if people started every day of the week, making it easier to make new friends.

All Nooglers, the term for a new Google employee, are told to report on their first day to a particular lobby. Someone from physical security checks them in as they arrive. Someone from People Operations (Google's human resources department) arrives at 9 AM and brings everyone to a room with classroom-style desks. Each desk has a computer set up specifically for filling out and electronically signing all their paperwork. The room is used for other purposes during the week, and someone from the helpdesk is responsible for setting up the computers before the end of the day on Friday, and packing them up sometime on Monday.

After signing all paperwork, a lot goes on behind the scenes: Accounts are enabled, and so on. These things take time, so various activities are planned. Nooglers sit through a number of presentations by different teams. They are walked to security to have their ID photos taken. They get a tour of the campus. They have lunch. They visit the helpdesk and pick up their laptops. All of these activities are timed so that any prerequisites have time to complete. For example, ID photos can be taken anytime, but the badges take a certain amount of time to print and be activated. Laptops cannot be picked up until accounts are enabled and shouldn't be picked up too early in the day; otherwise, they must be lugged around on the campus tour. Everything is carefully orchestrated.

Each Noogler is assigned a mentor from his or her new team. At 3 PM the mentors arrive to pick up their assigned Noogler and bring them to meet the rest of their new team.

When they log into Google Calendar, they see it has been prepopulated with events for the next two weeks based on which training sessions their role is assigned to.

When Tom entered Google he was impressed at how well run the entire onboarding process worked. Every edge case and error-prone situation was handled flawlessly. For example, unlike most Nooglers, his first week was at the headquarters in Mountain View even though he would be working in the New York office, which has a different insurance provider and tax forms. While he expected this to cause confusion, the coordinator was prepared. While everyone else in the room received a packet of information for Mountain View employees, Tom was handled the equivalent packet for NYC employees.

Nooglers were also told that when they submitted tickets to the helpdesk they should put the word “NOOGLER” in the subject line. This would result in the request receiving special attention. It reduced the embarrassment that Nooglers might feel when asking ignorant questions. It also let the helpdesk technicians work more effectively. It alerted the technician that the request might be coming from someone who wasn’t familiar with Google terminology, and that the technical issue should be debugged from the standpoint of getting something to work for the first time, rather than fixing a previously working system.

12.6 Summary

Onboarding is the process that brings newly hired employees into the organization. It is the responsibility of the human resources department but many of the tasks touch on the IT department. The onboarding process is important because it forms the new employee’s first impression of the IT department, and the company as a whole.

The ultimate goal of onboarding is employees who are productive in their new role, as soon as possible. There are five practices that are required to have a smooth onboarding process:

- **Drive the process by a timeline of onboarding actions.** We turn the requirements into a set of tasks, and then order the tasks as a set of milestones or a timeline. This communicates to all teams what their responsibilities are and avoids confusion. This timeline is then used for each new hire as a checklist. A good process is transparent and all teams can see the status of the checklist in real time.
- **Determine the person’s needs ahead of his or her arrival.** Onboarding starts long before the person arrives. Information must be collected from

the new employee, hardware must be ordered, and accounts and other systems need to be provisioned.

- **Perform each onboarding action by its deadline.** One person might do all the tasks, or the tasks might be spread out among many people on many teams. There might be a central coordinator, or the process might be self-organizing. Larger organizations might have a project manager coordinating the entire onboarding program.
- **Encourage cross-team communication about issues as they arise.** Problems and unexpected situations will arise. Each team needs to be able to communicate with the coordinator, peer teams, or everyone. A common problem is teams muddling through an issue because they feel they can't speak up about it, or don't know who to talk to. This creates extra work and delays.
- **Periodically reflect on the process and improve it.** The best suggestions for improvement come from the people doing the work. There should be frequent opportunities to raise issues and, occasionally, larger meetings focused exclusively on process improvement.

Exercises

1. What is onboarding and what are its goals?
2. How do we judge if an onboarding process is done well? What are internal and external indicators?
3. What are IT's responsibilities in the onboarding process?
4. What are the five keys to a well-organized onboarding experience?
5. Why is it important that IT's involvement in onboarding a new employee start before that person's first day? Give examples.
6. Describe what you experienced as the onboarding process at your current or recent place of employment. Identify both IT and non-IT tasks.
7. How is onboarding coordinated in small, medium, and large organizations?
8. How is onboarding coordinated in your organization? Identify both IT and non-IT tasks.
9. How many people are involved in your company's onboarding process? Name them.
10. In your organization, how do the teams involved in onboarding communicate? How could this be improved?

11. In your organization, if someone involved in onboarding had a suggestion for how to improve the process, how would he or she make this change?

Part III: Servers

Chapter 13. Server Hardware Strategies

Another group of machines in your organization is the server fleet. This population of machines is used to provide application services, web services, databases, batch computation, back-office processing, and so on.

A single machine might provide one service or many. For example, a single server might be a dedicated file server, or it might be a file server, a DNS server, and a wiki server while performing many other functions. Alternatively, some services might be too large to fit on a single machine. Instead, the work of the service may be distributed over many machines. For example, Google's Gmail service is distributed over thousands of machines, each doing a small fraction of the work.

By definition a server has dependents, usually many. A single server may have hundreds of clients relying on it. A web server may have thousands of users depending on it. Contrast this to a laptop or desktop, which generally has just a single user.

In a model that has one big server, or just a few, any investment made in a server is amortized over all the dependents or users. We justify any additional expense to improve performance or increase reliability by looking at the dependents' needs. In this model servers are also expected to last many years, so paying for spare capacity or expandability is an investment in extending its life span.

By comparison, when we have hundreds or thousands of servers, saving a little money on a per-machine basis saves a lot of money overall. If we are buying 100 machines, and can save a few dollars on each one, those savings add up quickly.

This chapter's advice applies to a typical enterprise organization that mostly purchases off-the-shelf hardware and software, with a limited number of home-grown applications. Volume 2 of this book series is more specifically geared toward web-based applications and large clusters of machines.

There are many strategies for providing server resources. Most organizations use a mix of these strategies.

The three most common strategies are:

- **All eggs in one basket:** One machine used for many purposes
- **Beautiful snowflakes:** Many machines, each uniquely configured

- **Buy in bulk, allocate fractions:** Large machines partitioned into many smaller virtual machines using virtualization or containers

In addition, there are variations and alternatives:

- **Grid computing:** Many machines managed one as unit
- **Blade servers:** A hardware architecture that places many machines in one chassis
- **Cloud-based compute services:** Renting use of someone else's servers
- **Software as a service (SaaS):** Web-hosted applications
- **Server appliances:** Purpose-built devices, each providing a different service

The rest of this chapter describes these strategies and provides advice on how to manage them.

What Is a Server?

The term “server” is overloaded and ambiguous. It means different things in different contexts. The phrase “web server” might refer to a host being used to provide a web site (a machine) or the software that implements the HTTP protocol (Apache HTTPD).

Unless specified, this book uses the term “server” to mean a machine. It refers to a “service” as the entire hardware/software combination that provides the service users receive. For example, an email server (the hardware) runs MS Exchange (the software) to provide the email service for the department.

Note that Volume 2 of this book series deals with this ambiguity differently. It uses the term “server” to mean the server of a client–server software arrangement—for example, an Apache HTTPD server. When talking about the hardware it uses the word “machine.”

13.1 All Eggs in One Basket

The most basic strategy is to purchase a single server and use it for many services. For example, a single machine might serve as the department's DNS server, DHCP server, email server, and web server. This is the “all eggs in one basket” approach. We don't recommend this approach. Nevertheless, we see it often enough that we felt we should explain how to make the best of it.

If you are going to put all your eggs in one basket, make sure you have a

really, really, really strong basket. Any hardware problems the machine has will affect many services. Buy top-of-the-line hardware and a model that has plenty of expansion slots. You'll want this machine to last a long time.

In this setting it becomes critical to ensure data integrity. Hard disks fail. Expecting them to never fail is irrational. Therefore RAID should be used so that a single disk can fail and the system will continue to run. Our general rule of thumb is to use a 2-disk mirror (RAID 1) for the boot disk. Additional storage should be RAID 5, 6, or 10 as needed. RAID 0 offers zero data integrity protection (the “zero” in the name reflects this). Obviously, different applications demand different configurations but this fits most situations. Putting plain disks without RAID 1 or higher on a server like this is asking for trouble. Any disk malfunction will result in a bad day as you restore from backups and deal with any data loss. When RAID was new, it was expensive and rare. Now it is inexpensive and should be the default for all systems that need reliability.

Data integrity also requires regular data backups to tape or another system. These backups should be tested periodically. As explained in [Section 43.3.3](#), RAID is not a backup strategy. If this system dies in a fire, RAID will not protect the data.

If a server is expected to last a long time, it will likely need to be expanded. Over time most systems will need to handle more users, more data, or more applications. Additional slots for memory, interfaces, and hard drive bays make it easy to upgrade the system without replacing it. Upgradable hardware is more expensive than fixed-configuration equivalents, but that extra expense is an insurance policy against more costly upgrades later. It is much less expensive to extend the life of a machine via an incremental upgrade such as adding RAM than to do a wholesale replacement of one machine with another.

Forklift Upgrades

The term **forklift upgrade** is industry slang for a wholesale replacement. In such a situation you are removing one machine with a metaphorical forklift and dropping a replacement in its place.

Another problem with the “one basket” strategy is that it makes OS upgrades very difficult and risky. Upgrading the operating system is an all-or-nothing endeavor. The OS cannot be upgraded or patched until all the services are verified to work on the new version. One application may hold back all the others.

Upgrading individual applications also becomes more perilous. For example, what if upgrading one application installs the newest version of a shared library, but another application works only with the older version? Now you have one application that isn't working and the only way to fix it is to downgrade it, which will make the other application fail. This Catch-22 is known as **dependency hell**.

Often we do not discover these incompatibilities until after the newer software is installed or the operating system is upgraded. Some technologies can be applied to help in this situation. For example, some disk storage systems have the ability to take a snapshot of the disk. If an upgrade reveals incompatibility or other problems, we can revert to a previous snapshot. RAID 1 mirrors and file systems such as ZFS and Btrfs have snapshotting capabilities.

Upgrading a single machine with many applications is complex and is the focus of [Chapter 33](#), “[Server Upgrades](#).”

The more applications a big server is running, the more difficult it becomes to schedule downtime for hardware upgrades. Very large systems permit hardware upgrades to be performed while they remain running, but this is rare for components such as RAM and CPUs. We can delay the need for downtime by buying extra capacity at the start, but this usually leads to more dependencies on the machine, which in turn exacerbates the scheduling problem we originally tried to solve.

13.2 Beautiful Snowflakes

A better strategy is to use a separate machine for each service. In this model we purchase servers as they are needed, ordering the exact model and configuration that is right for the application.

Each machine is sized for the desired application: RAM, disk, number and speeds of NICs, and enough extra capacity, or expansion slots, for projected growth during the expected life of the machine. Vendors can compete to provide their best machine that meets these specifications.

The benefit of this strategy is that the machine is the best possible choice that meets the requirements. The downside is that the result is a fleet of unique machines. Each is a beautiful, special little snowflake.

While snowflakes are beautiful, nobody enjoys a blizzard. Each new system adds administrative overhead proportionally. For example, it would be a considerable burden if each new server required learning an entirely new RAID storage subsystem. Each one would require learning how to configure it, replace

disks, upgrade the firmware, and so on. If, instead, the IT organization standardized on a particular RAID product, each new machine would simply benefit from what was learned earlier.

13.2.1 Asset Tracking

When managing many unique machines it becomes increasingly important to maintain an inventory of machines. The inventory should document technical information such as the operating system and hardware parameters such as amount of RAM, type of CPU, and so on. It should also track the machine owner (either a person or a department), the primary contact (useful when the machine goes haywire), and the services being used. When possible, this information should be collected through automated means; this makes it less likely the information will become outdated. If automated collection is not possible, an automated annual review, requiring affirmations from the various contacts, can detect obsolete information that needs to be updated.

A variety of inventory applications are available, many of which are free or low cost. Having even a simple inventory system is better than none. A spreadsheet is better than keeping this information in your head. A database is better than a spreadsheet.

13.2.2 Reducing Variations

Always be on the lookout for opportunities to reduce the number of variations in platforms or technologies being supported. Discourage gratuitous variations by taking advantage of the fact that people lean toward defaults. Make right easy: Make sure that the *lazy path* is the path you want people to take. Select a default hardware vendor, model, and operating system and make it super easy to order. Provide automated OS installation, configuration, and updates. Provide a wiki with information about recommended models and options, sales contact information, and assistance. We applaud you for being able to keep all that information in your head, but putting it on a wiki helps the entire organization stick to standards. That helps you in the long term.

The techniques discussed in [Part II](#), “[Workstation Fleet Management](#),” for managing variation also apply to servers. In particular, adopt a policy of supporting a limited number of generations of hardware. When a new generation of hardware is introduced, the oldest generation is eliminated. This practice also works for limiting the number of OS revisions in use.

Google's SysOps (internal IT) organization had a policy of supporting at most two Linux releases at any given time, and a limited number of hardware generations. For a new OS release or hardware model to be introduced into the ecosystem, the oldest one had to be eliminated.

Eliminating the old generation didn't happen automatically. People were not continuing to use it because of laziness; there was always some deep technical reason, dependency, lack of resources, or political issue preventing the change. Finding and eliminating the stragglers took focused effort.

To accelerate the process, a project manager and a few SAs would volunteer to form a team that would work to eliminate the stragglers. These teams were called "death squads," an insensitive name that could have been invented only by people too young to remember the politics of Central America in the late twentieth century.

The team would start by publishing a schedule: the date the replacement was available, the date the old systems would no longer receive updates, and the date the old systems would be disconnected from the network. There were no exceptions. Management had little sympathy for whining. Employees sometimes complained the first time they experienced this forced upgrade process but soon they learned that restricting technologies to a small number of generations was a major part of the Google operational way and that services had to be built with the expectation of being easy to upgrade or rebuild.

The team would identify machines that still used the deprecated technology. This would be published as a shared spreadsheet, viewable and editable by all. Columns would be filled in to indicate the status: none, owner contacted, owner responded, work in progress, work complete. As items were complete, they would be marked green. The system was very transparent. Everyone in the company could see the list of machines, who had stepped up to take ownership of each, and their status. It was also very visible if a team was not taking action.

The death squad would work with all the teams involved and offer support and assistance. Some teams already had plans in place. Others needed a helping hand. Some needed cajoling and pressure from management. The death squad would collaborate and assist until the older technology was eliminated.

This system was used often and was a key tactic in preventing Google's

very large fleet from devolving into a chaotic mess of costly snowflakes.

13.2.3 Global Optimization

While it sounds efficient to customize each machine to the exact needs of the service it provides, the result tends to be an unmanageable mess.

This is a classic example of a local optimization that results in a global de-optimization. For example, if all server hardware is from one vendor, adding a single machine from a different vendor requires learning a new firmware patching system, investing in a new set of spare parts, learning a new customer support procedure, and so on. The added work for the IT team may not outweigh the benefits gained from using the new vendor.

The one-off hardware might be rationalized by the customer stating that spares aren't required, that firmware upgrades can be skipped, or that the machine will be self-supported by the customer. These answers are nice in theory, but usually lead to even more work for the IT department. When there is a hardware problem, the IT department will be blamed for any difficulties. Sadly the IT department must usually say no, which makes this group look inflexible.

The Unique Storage Device

A university professor in the mathematics department purchased a custom storage system against the advice of the IT department. He rationalized the exception by promising to support it himself.

During a software upgrade the system stopped working. The professor was unable to fix the problem, and other professors who had come to rely on the system were dead in the water.

After a month of downtime, the other professors complained to the department chair, who complained to the IT department. While the department chair understood the agreement in place, he'd hear none of it. This was an emergency and IT's support was needed. The IT group knew that once they got involved, any data loss would be blamed on them. Their lack of experience with the device increased that risk.

The end of this story is somewhat anticlimactic. While the IT group was establishing an action plan in cooperation with the vendor, the professor went rogue and booted the storage server with a rescue disk, wiped and reinstalled the system, and got it working again. Only then did he concede that there was no irreplaceable data on the system.

Unfortunately, this experience taught the professor that he could ignore IT's hardware guidance since he'd get support when he needed it, and that he didn't really need it since he fixed the problem himself.

The other faculty who had come to depend on this professor's system realized the value of the support provided by the IT group. While the previous system was "free," losing access for a month was an unacceptable loss of research time. The potential for catastrophic data loss was not worth the savings.

What the IT department learned was that there was no such thing as self-support. In the future, they developed ways to be involved earlier on in purchasing processes so that they could suggest alternatives before decisions had been made in the minds of their customers.

13.3 Buy in Bulk, Allocate Fractions

The next strategy is to buy computing resources in bulk and allocate fractions of it as needed.

One way to do this is through virtualization. That is, an organization purchases large physical servers and divides them up for use by customers by creating individual virtual machines (VMs). A virtualization cluster can grow by adding more physical hardware as more capacity is needed.

Buying an individual machine has a large overhead. It must be specified, ordered, approved, received, rack mounted, and prepared for use. It can take weeks. Virtual machines, by comparison, can be created in minutes. A virtualization cluster can be controlled via a portal or API calls, so automating processes is easy.

VMs can also be resized. You can add RAM, vCPUs, and disk space to a VM via an API call instead of a visit to the datacenter. If customers request more memory and you add it using a management app on your iPhone while sitting on the beach, they will think you are doing some kind of magic.

Virtualization improves computing efficiency. Physical machines today are so powerful that applications often do not need the full resources of a single machine. The excess capacity is called **stranded capacity** because it is unusable in its current form. Sharing a large physical machine's power among many smaller virtual machines helps reduce stranded capacity, without getting into the "all eggs in one basket" trap.

Stranded capacity could also be mitigated by running multiple services on the

same machine. However, virtualization provides better isolation than simple multitasking. The benefits of isolation include

- **Independence:** Each VM can run a different operating system. On a single physical host there could be a mix of VMs running a variety of Microsoft Windows releases, Linux releases, and so on.
- **Resource isolation:** The disk and RAM allocated to a VM are committed to that VM and not shared. Processes running on one VM can't access the resources of another VM. In fact, programs running on a VM have little or no awareness that they are running on VMs, sharing a larger physical machine.
- **Granular security:** A person with root access on one VM does not automatically have privileged access on another VM. Suppose you had five services, each run by a different team. If each service was on its own VM, each team could have administrator or root access for its VM without affecting the security of the other VMs. If all five services were running on one machine, anyone needing root or administrator access would have privileged access for all five services.
- **Reduced dependency hell:** Each machine has its own operating system and system libraries, so they can be upgraded independently.

13.3.1 VM Management

Like other strategies, keeping a good inventory of VMs is important. In fact, it is more important since you can't walk into a datacenter and point at the machine. The cluster management software will keep an inventory of which VMs exist, but you need to maintain an inventory of who owns each VM and its purpose.

Some clusters are tightly controlled, only permitting the IT team to create VMs with the care and planning reminiscent of the laborious process previously used for physical servers. Other clusters are general-purpose compute farms providing the ability for customers to request new machines on demand. In this case, it is important to provide a self-service way for customers to create new machines. Since the process can be fully automated via the API, not providing a self-service portal or command-line tool for creating VMs simply creates more work for you and delays VM creation until you are available to process the request. Being able to receive a new machine when the SAs are unavailable or busy reduces the friction in getting the compute resources customers need. In addition to creating VMs, users should be able to reboot and delete their own VMs.

There should be limits in place so that customers can't overload the system by creating too many VMs. Typically limits are based on existing resources, daily limits, or per-department allocations.

Users can become confused if you permit them to select any amount of disk space and RAM. They often do not know what is required or reasonable. One strategy is to simply offer reasonable defaults for each OS type. Another strategy is to offer a few options: small, medium, large, and custom.

As in the other strategies, it is important to limit the amount of variation. Apply the techniques described previously in this chapter and in [Part II, "Workstation Fleet Management."](#) In particular, adopt a policy of supporting a limited number of OS versions at any given time. For a new version to be introduced, the oldest generation is eliminated.

13.3.2 Live Migration

Most virtual machine cluster management systems permit live migration of VMs, which means a VM can be moved from one physical host to another while it is running. Aside from a brief performance reduction during the transition, the users of the VM do not even know they're being moved.

Live migration makes management easier. It can be used to rebalance a cluster, moving VMs off overloaded physical machines to others that are less loaded. It also lets you work around hardware problems. If a physical machine is having a hardware problem, its VMs can be evacuated to another physical machine. The owners of the VM can be blissfully unaware of the problem. They simply benefit from the excellent uptime.

The architecture of a typical virtualization cluster includes many physical machines that share a SAN for storage of the VM's disks. By having the storage external to any particular machine, the VMs can be easily migrated between physical machines.

Shared storage is depicted in [Figure 13.1](#). The VMs run on the VM servers and the disk volumes they use are stored on the SAN. The VM servers have little disk space of their own.

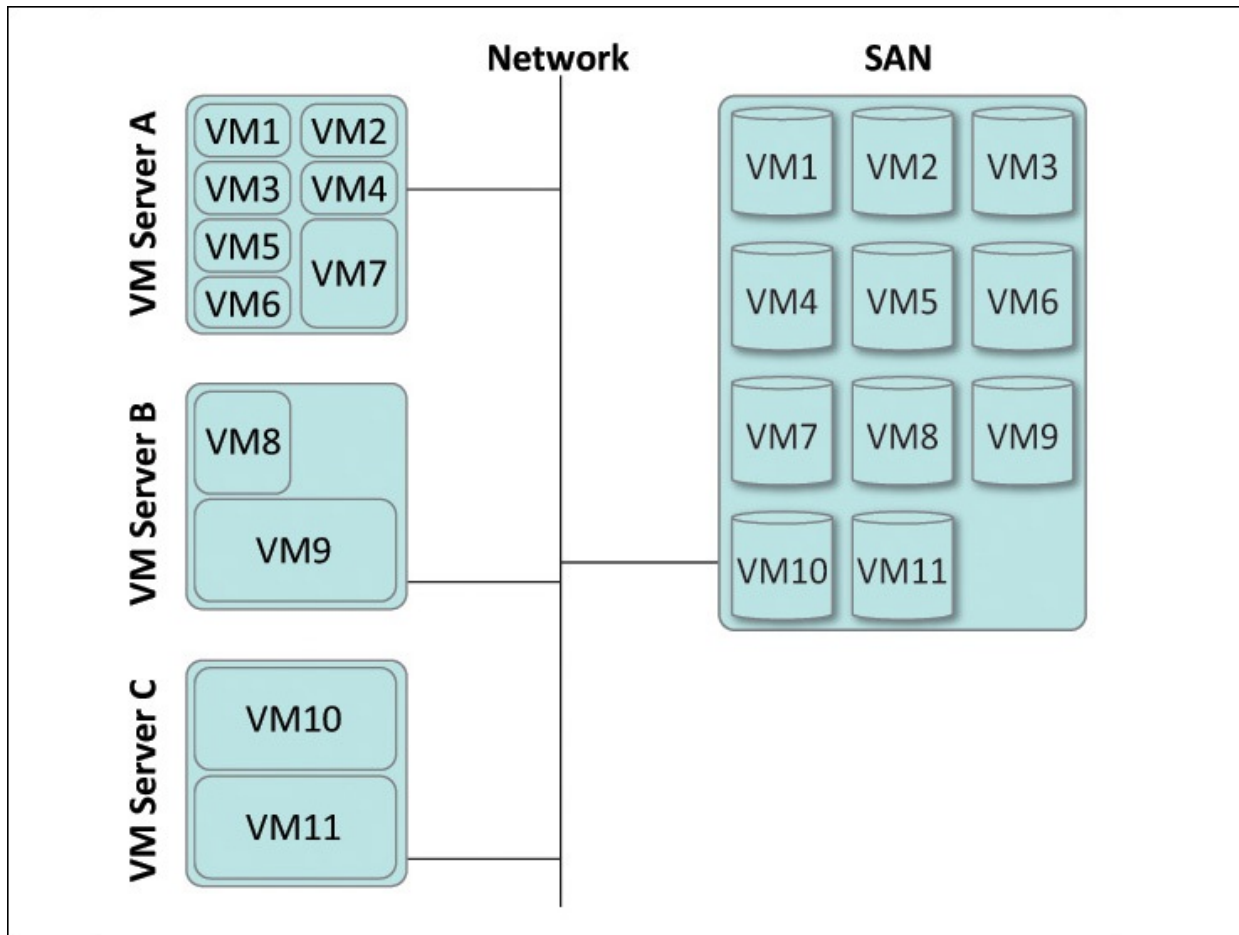


Figure 13.1: VM servers with shared storage

In [Figure 13.2](#) we see VM7 is being migrated from VM Server A to VM Server B. Because VM7's disk image is stored on the SAN, it is accessible from both VM servers as the migration process proceeds. The migration process simply needs to copy the RAM and CPU state between A and B, which is a quick operation. If the entire disk image had to be copied as well, the migration could take hours.

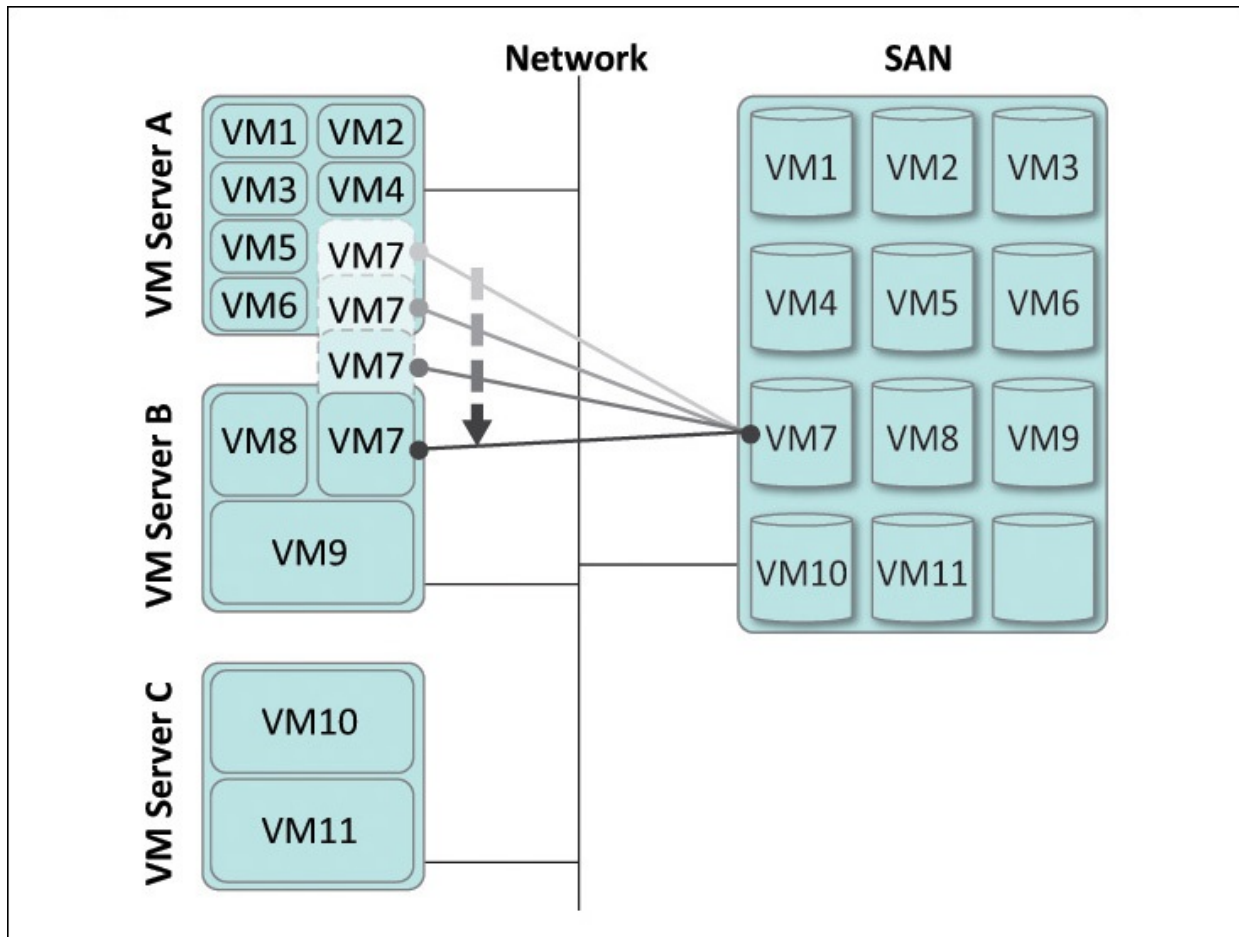


Figure 13.2: Migrating VM7 between servers

13.3.3 VM Packing

While VMs can reduce the amount of stranded compute capacity, they do not eliminate it. VMs cannot span physical machines. As a consequence, we often get into situations where the remaining RAM on a physical machine is not enough for a new VM.

The best way to avoid this is to create VMs that are standard sizes that pack nicely. For example, we might define the small configuration such that exactly eight VMs fit on a physical machine with no remaining stranded space. We might define the medium configuration to fit four VMs per physical machine, and the large configuration to fit two VMs per physical machine. In other words, the sizes are $\frac{1}{8}$, $\frac{1}{4}$, and $\frac{1}{2}$. Since the denominators are powers of 2, combinations of small, medium, and large configurations will pack nicely.

13.3.4 Spare Capacity for Maintenance

If a physical machine needs to be taken down for repairs, there has to be a place where the VMs can be migrated if you are to avoid downtime. Therefore you must always reserve capacity equivalent to one or more physical servers. Reserving capacity equivalent to one physical server is called $N + 1$ redundancy. You may need $N + 2$ or higher redundancy if there is a likelihood that multiple physical machines will need maintenance at the same time. For example, a network with hundreds of physical machines is likely to have many physical machines out for repair at any given time.

One strategy is to keep one physical machine entirely idle so that, when needed, the VMs from the machine to be repaired can all be migrated to this machine. This scheme is depicted in [Figure 13.3\(a\)](#). When physical machine A, B, C, or D needs maintenance, its VMs are transferred to machine E. Any one machine can be down at a time. It doesn't matter which combination of VMs are on the machine, as long as the spare is as large as any of the other machines.

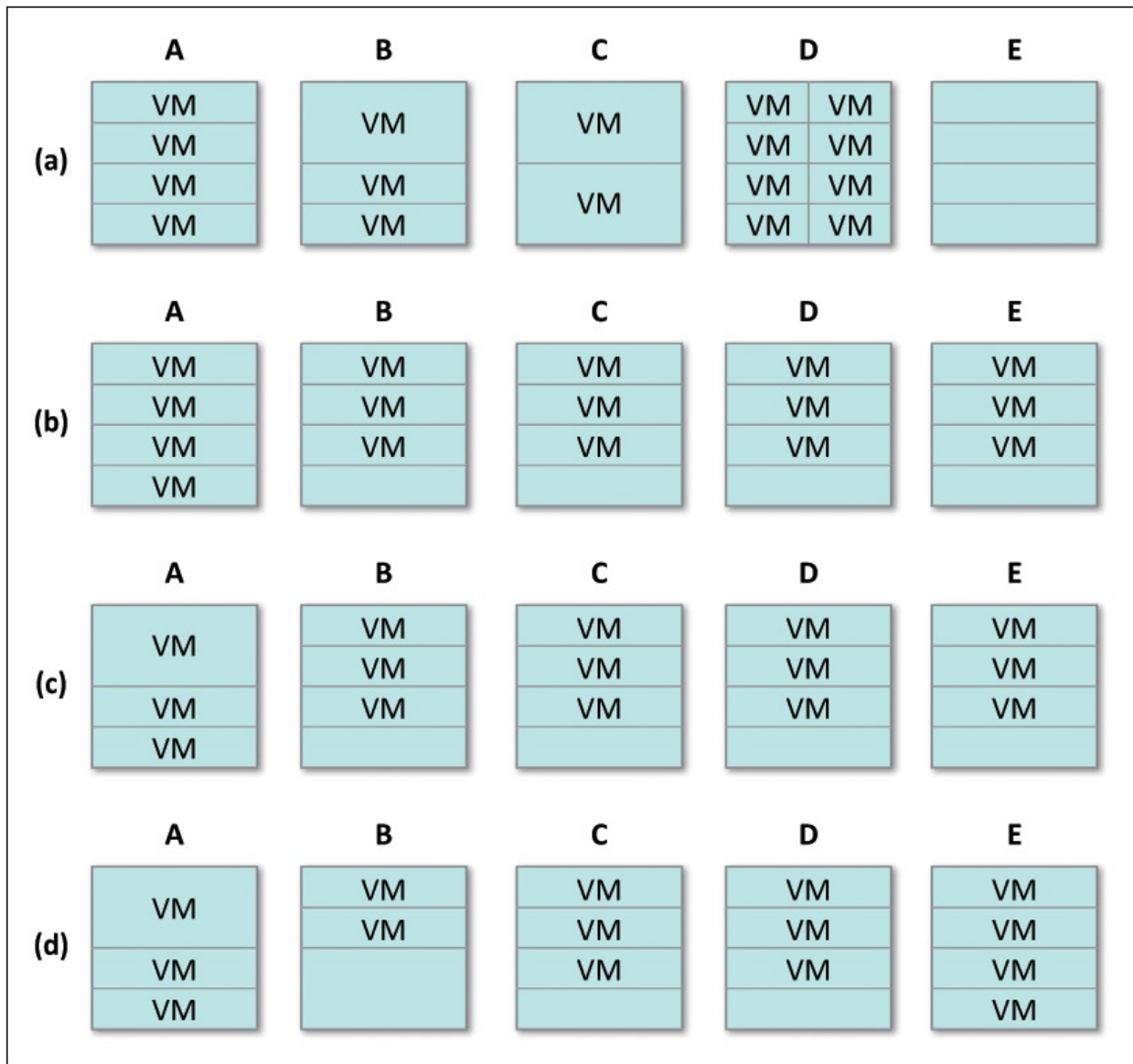


Figure 13.3: VM packing

Of course, this arrangement means that the spare machine is entirely unused. This seems like a waste since that capacity could be used to alleviate I/O pressure such as contention for network bandwidth, disk I/O, or other bottlenecks within the cluster.

Another strategy is to distribute the spare capacity around the cluster so that the individual VMs can share the extra I/O bandwidth. When a machine needs to be evacuated for repairs, the VMs are moved into the spare capacity that has been spread around the cluster.

This scheme is depicted in [Figure 13.3\(b\)](#). If machine A requires maintenance, its VMs can be moved to B, C, D, and E. Similar steps can be followed for the

other machines.

Unfortunately, this causes a new problem: There might not be a single machine with enough spare capacity to accept a VM that needs to be evacuated from a failing physical host. This situation is depicted in [Figure 13.3\(c\)](#). If machine A needs maintenance, there is no single machine with enough capacity to receive the largest of its VMs. VMs cannot straddle two physical machines. This is not a problem for machines B through E, whose VMs can be evacuated to the remaining space.

If machine A needed to be evacuated, first we would need to consolidate free space onto a single physical machine so that the larger VM has a place to move to. This is known as a Towers of Hanoi situation, since moves must be done recursively to enable other moves. These additional VM moves make the evacuation process both more complex and longer. The additional VM migrations affect VMs that would otherwise not have been involved. They now must suffer the temporary performance reduction that happens during migration, plus the risk that the migration will fail and require a reboot.

[Figure 13.3\(d\)](#) depicts the same quantity and sizes of VMs packed to permit any single machine to be evacuated for maintenance.

Some VM cluster management systems include a tool that will calculate the minimum number of VM moves to evacuate a physical machine. Other systems simply refuse to create a new VM if it will create a situation where the cluster is no longer $N + 1$ redundant. Either way, virtual clusters should be monitored for loss of $N + 1$ redundancy, so that you are aware when the situation has happened and it can be corrected proactively.

13.3.5 Unified VM/Non-VM Management

Most sites end up with two entirely different ways to request, allocate, and track VMs and non-VMs. It can be beneficial to have one system that manages both. Some cluster management systems will manage a pool of bare-metal machines using the same API as VMs. Creating a machine simply allocates an unused machine from the pool. Deleting a machine marks the machine for reuse.

Another way to achieve this is to make everything a VM, even if that means offering an extra large size, which is a VM that fills the entire physical machine: $\frac{1}{1}$. While such machines will have a slight performance reduction due to the VM overhead, unifying all machine management within one process benefits customers, who now have to learn only one system, and makes management easier.

13.3.6 Containers

Containers are another virtualization technique. They provide isolation at the process level instead of the machine level. While a VM is a machine that shares physical hardware with other VMs, each container is a group of processes that run in isolation on the same machine. All of the containers run under the same operating system, but each container is self-contained as far as the files it uses. Therefore there is no dependency hell.

Containers are much lighter weight and permit more services to be packed on fewer machines. Docker, Mesos, and Kubernetes are popular systems for managing large numbers of containers. They all use the same container format, which means once a container is created, it can be used on a desktop, server, or huge farm of servers.

A typical containerization cluster has many physical machines running a stripped-down version of a server OS. Since each container is self-contained, the OS does not need much installed other than the kernel, the management framework, and the files required to boot the system. CoreOS and Red Hat Atomic are two such stripped-down operating systems.

Pros and cons of virtualization and containerization, as well as technical details of how they work, can be found in Volume 2, [Chapter 3](#), “Selecting a Service Platform,” of this book series.

Self-service container administration is made possible by Kubernetes and other container management systems. Customers provide the container specification and the management system finds an available machine and spins up the container.

One bad situation people get into with containers is a lack of reproducibility. After using the system for a while, there comes a day when a container needs to be rebuilt from scratch to upgrade a library or other file. Suddenly the team realizes no one is around who remembers how the container was made. The way to prevent this is to make the container’s creation similar to compiling software: A written description of what is to be in the container is passed through automation that reads the description and outputs the container. The description is tracked in source code control like any other source code. Containers should be built using whatever continuous integration (CI) system is used for building other software in your organization.

13.4 Grid Computing

Grid computing takes many similar machines and manages them as a single unit.

For example, four racks of 40 servers each would form a grid of 160 machines. Each one is configured exactly alike—same hardware and software.

To use the grid, a customer specifies how many machines are needed and which software package to run. The grid management system allocates the right number of machines, installs the software on them, and runs the software. When the computation is done, the results are uploaded to a repository and the software is de-installed.

A big part of grid management software is the scheduling algorithm. Your job is held until the number of machines requested are available. When you request many machines, that day may never come. You would be very disappointed if your job, which was expected to run for an hour, took a week to start.

To start a big request, one has to stall small requests until enough machines are available, which wastes resources for everyone. Suppose your job required 100 machines. The scheduler would stall any new jobs until 100 machines are idle. Suppose at first there are 50 machines idle. An hour later, another job has completed and now there are 75 machines idle. A day later a few more jobs have completed and there are 99 machines idle. Finally another job completes and there are 100 free machines, enough for your job to run. During the time leading up to when your job could run, many machines were sitting idle. In fact, more than 75 compute-days were lost in this example.

The scheduler must be very smart and mix small and big requests to both minimize wait time and maximize utilization. A simple algorithm is to allocate half the machines for big jobs. They will stay busy if there is always a big job ready to run. Allocate the other half of the machines for smaller jobs; many small and medium jobs will pass through those machines.

More sophisticated algorithms are invented and tested all the time. At Google the scheduling algorithms have become so complex that a simulator was invented to make it possible to experiment with new algorithms without having to put them into production.

Typically grids are very controlled systems. All allocations are done through the grid management and scheduling system. Each machine runs the same OS configuration. When entropy is detected, a machine is simply wiped and reloaded.

Because many machines are being purchased, shaving a few dollars off the cost of each machine pays big dividends. For example, a video card is not needed since these machines will not be used for interactive computing. For a 1,000-machine cluster, this can save thousands of dollars. RAID cards, fancy

plastic front plates with awesome LED displays, redundant power supplies, and other add-ons are eliminated to save a few dollars here and there. This multiplies out to thousands or millions of dollars overall.

Data integrity systems such as RAID cards are generally not installed in grid hardware because the batch-oriented nature of jobs means that if a machine dies, the batch can be rerun.

While one wants a grid that is reliable, compute efficiency is more important. Efficiency is measured in dollars per transaction. Rather than examining the initial purchase price, the total cost of ownership (TCO) is considered. For example, ARM chips are less expensive than x86 Intel chips, but they are slower processors. Therefore you need more of them to do the same job. Is it better to have 1,000 high-powered Intel-based machines or 2,000 ARM-based machines? The math can be fairly complex. The ARM chips use less power but there are more of them. The amount of power used is related directly to how much cooling is needed. The additional machines require more space in the datacenter. What if the additional machines would result in a need to build more datacenters? TCO involves taking all of these factors into account during the evaluation.

Grid computing has other constraints. Often there is more bandwidth between machines on the same rack, and less bandwidth between racks. Therefore some jobs will execute faster if rack locality (putting all related processes in one rack) can be achieved.

Grid computing is more efficient than virtualization because it eliminates the virtualization overhead, which is typically a 5 to 10 percent reduction in performance.

Grids are easier to manage because what is done for one machine is done for all machines. They are fungible units—each one can substitute for the others. If one machine dies, the scheduler can replace it with another machine in the grid.

Case Study: Disposable Servers

Like many web sites, Yahoo! builds mammoth clusters of low-cost 1U PC servers. Racks are packed with as many servers as possible, with dozens or hundreds configured to provide each service required. Yahoo! once reported that when a unit died, it was more economical to power it off and leave it in the rack rather than repair the unit. Removing dead units might accidentally cause an outage if other cables were loosened in the process. Eventually the machine would be reaped and many dead machines would

be repaired en masse.

13.5 Blade Servers

There is a lot of overhead in installing individual machines. Each machine needs to be racked, connected to power, networked, and so on. Blade servers reduce this overhead by providing many servers in one chassis.

A blade server has many individual slots that take motherboards, called **blades**, that contain either a computer or storage. Each blade can be installed quickly and easily because you simply slide a card into a slot. There are no cables to connect; the blade's connector conveys power, networking, and I/O connectivity. Additional capacity is added by installing more blades, or replacing older blades with newer, higher-capacity models.

Blades can be used to implement many of the strategies listed previously in this chapter. Computers within the blade chassis can be allocated individually, or used to create a virtualization cluster, or used to create a grid.

Another benefit of blade systems is that they are software configurable. Assigning a disk to a particular computer or a computer to a particular network is done through a management console or API. Does a particular blade computer need an additional hard drive? A few clicks and it is connected to storage from another blade. No trip to the datacenter is required.

Blade systems are most cost-effective when the chassis lasts many years, enabling you to upgrade the blades for the duration of its lifetime. This amortizes the cost of the chassis over many generations of blades. The worst-case scenario is that shortly after investing in a chassis, a newer, incompatible chassis model becomes available. When you buy a blade-based system, you are betting that the chassis will last a long time and that new blades will be available for its entire lifespan. If you lose the bet, a forklift upgrade can be very expensive.

Grid Uniformity

A division of a large multinational company was planning on replacing its aging multi-CPU server with large grid computing environment, implemented as a farm of blade servers. The application would be recoded so that instead of using multiple processes on a single machine, it would use processes spread over the blade farm. Each blade would be one node of a vast compute farm to which jobs could be submitted, with the results consolidated on a controlling server.

This had wonderful scalability, since a new blade could be added to the farm and be usable within minutes. No direct user logins were needed, and no SA work would be needed beyond replacing faulty hardware and managing which blades were assigned to which applications. To this end, the SAs engineered a tightly locked-down minimal-access solution that could be deployed in minutes. Hundreds of blades were purchased and installed, ready to be purposed as the customer required.

The problem came when application developers found themselves unable to manage their application. They couldn't debug issues without direct access. They demanded shell access. They required additional packages. They stored unique state on each machine, so automated builds were no longer viable. All of a sudden, the SAs found themselves managing 500 individual servers rather than one unified blade farm. Other divisions had also signed up for the service and made the same demands.

A number of things could have prevented this problem. Management should have required more discipline. Once the developers started requesting access, management should have set limits that would have prevented the system from devolving into hundreds of custom machines. Deployment of software into production should have been automated using a continuous integration/continuous deployment (CI/CD) system. If the only way to put things in production is by automated deployment, then repeatability can be more easily achieved. There should have been separate development and production environments, with fewer access restrictions in the development environment. More attention to detail at the requirements-gathering stage might have foreseen the need for developer access, which would have led to the requirement for and funding of a development environment.

13.6 Cloud-Based Compute Services

Another strategy is to not own any machines at all, but rather to rent capacity on someone else's system. Such cloud-based computing lets you benefit from the economies of scale that large warehouse-size datacenters can provide, without the expense or expertise to run them. Examples of cloud-based compute services include Amazon AWS, Microsoft Azure, and Google Compute Engine.

13.6.1 What Is the Cloud?

Cloud computing is a marketing term fraught with confusing interpretations.

Marketing people will gladly tell you that the cloud solves all problems, as if there are no problems left in the world now that the cloud is here.

There are three common definitions for **the cloud**, each coming from different communities:

- **Consumers:** When a typical consumer uses the term the cloud, they mean putting their data on a web-based platform. The primary benefit is that this data becomes accessible from anywhere. For example, consumers might have all their music stored in the cloud; as a result their music can be played on any device that has Internet access.
- **Business people:** Typically business people think of the cloud as some kind of rented computing infrastructure that is elastic. That is, they can allocate one or thousands of machines; use them for a day, week, or year; and give them back when they are done. They like the fact that this infrastructure is a pay-as-you-go and on-demand system. The on-demand nature is the most exciting because they won't have to deal with IT departments that could take months to deliver a single new machine, or simply reject their request. Now with a credit card, they have a partner that always says yes.
- **IT professionals:** When all the hype is removed (and there is a lot of hype), cloud computing comes down to someone else maintaining hardware and networks so that customers can focus on higher-level abstractions such as the operating system and applications. It requires software that is built differently and new operational methods. IT professionals shift from being the experts in how to install and set up computers to being the experts who understand the full stack and become valued for their architectural expertise, especially regarding how the underlying infrastructure affects performance and reliability of the application, and how to improve both.

This book uses a combination of the last two definitions. The on-demand nature of the cloud benefits us by providing an elastic computing environment: the ability to rapidly and dynamically grow and shrink. As IT professionals, it changes our role to focus on performance and reliability, and be the architecture experts.

13.6.2 Cloud Computing's Cost Benefits

Over the years computer hardware has grown less expensive due to Moore's law and other factors. Conversely, the operation of computers has become

increasingly expensive. The increased operational cost diminishes and often eliminates the cost reductions.

The one place where operational cost has been going down instead of up is in large grids or clusters. There the entire infrastructure stack of hardware, software, and operations can be vertically integrated to achieve exceptional cost savings at scale.

Another way to think about this issue is that of all the previously discussed strategies, the “buy in bulk, allocate fractions” strategy described earlier is generally the most economical and proves the most flexible. Cloud-based compute services take that strategy to a larger scale than most companies can achieve on their own, which enables these smaller companies take advantage of these economics.

As these cost trends continue, it will become difficult for companies to justify maintaining their own computers. Cloud computing will go from the exception that few companies are able to take advantage of, to the default. Companies that must maintain their own hardware will be an exception and will be at an economic disadvantage.

Adoption of cloud computing is also driven by another cost: opportunity cost. Opportunity cost is the revenue lost due to missed opportunities. If a company sees an opportunity but the competition beats them to it, that could be millions of potential dollars lost.

Some companies miss opportunities because they cannot hire people fast enough to staff a project. Other companies do not even attempt to go after certain opportunities because they know they will not be able to ramp up the IT resources fast enough to address the opportunity. Companies have missed multi-million-dollar opportunities because an IT department spent an extra month to negotiate a slightly lower price from the vendor.

If cloud computing enables a company to spin up new machines in minutes, without any operations staff, the ability to address opportunities is greatly enhanced. At many companies it takes months, and sometimes an entire year, from the initial request to realizing a working server in production. The actual installation of the server may be a few hours, but it is surrounded by months of budget approvals, bureaucratic approvals, security reviews, and IT managers who think it is their job to always say “no.”

We are optimistic about cloud computing because it enables new applications that just can't be done any other way. Kevin McEntee, vice-president at Netflix, put it succinctly: “You can't put a price on nimble.” In his 2012 talk at re:Invent,

McEntee gave examples of how the elastic ability of cloud computing enabled his company to process video in new ways. He extolled the value of elastic computing's ability to let Netflix jump on opportunities that enabled it to be a part of the iPad launch and the Apple TV launch, and to quickly launch new, large libraries of videos to the public. Netflix has been able to make business deals that its competition couldn't. "If we were still building out datacenters in the old model we would not have been able to jump on those opportunities" ([McEntee 2012](#)).

There are legal and technical challenges to putting your data on other people's hardware. That said, do not assume that HIPAA compliance and similar requirements automatically disqualify you from using cloud-based services. Cloud vendors have a variety of compliant options and ways of managing risk. A teleconference between your legal compliance department and the provider's sales team may lead to surprising results.

It is also possible to have an in-house cloud environment. With this approach, a department runs an in-house cloud and bills departments for use. This is common in industries that must meet strict HIPAA compliance requirements but want the economic benefits of massive scale.

Volume 2 of this book series, especially [Chapter 3](#), "Selecting a Service Platform," contains advice about selecting and managing a cloud platform.

13.6.3 Software as a Service

Software as a service (SaaS) means using web-based applications. Many small companies have no servers at all. They are able to meet all their application needs with web-based offerings. For example, they host their web site using a hosting service, they use a web-based payroll provider, they share files using Dropbox, they use Salesforce for customer relationship management and sales process management, and they use Google Apps for Work for word processing, spreadsheets, and presentations. If they use Chromebooks, iPads, or other fixed-configuration web-browsing devices, they don't need traditional computers and can eliminate a lot of the traditional enterprise IT infrastructure that most companies require.

This strategy was impossible until the early 2010s. Since then, ubiquitous fast Internet connections, HTML5's ability to create interactive applications, and better security features have made this possible.

When a company adopts such a strategy, the role of the IT department becomes that of an IT coordinator and integrator. Rather than running clients and

services, someone is needed to coordinate vendor relationships, introduce new products into the company, provide training, and be the first stop for support before the provider is contacted directly. Technical work becomes focused on high-level roles such as software development for integrating the tools, plus low-level roles such as device support and repair management.

13.7 Server Appliances

An **appliance** is a device designed specifically for a particular task. Toasters make toast. Blenders blend. One could do these things using general-purpose devices, but there are benefits to using a device designed to do one task very well.

The computer world also has appliances: file server appliances, web server appliances, email appliances, DNS/DHCP appliances, and so on. The first appliance was the dedicated network router. Some scoffed, “Who would spend all that money on a device that just sits there and pushes packets when we can easily add extra interfaces to our VAX and do the same thing?” It turned out that quite a lot of people would. It became obvious that a box dedicated to doing a single task, and doing it well, was in many cases more valuable than a general-purpose computer that could do many tasks. And, heck, it also meant that you could reboot the VAX without taking down the network for everyone else.

A server appliance brings years of experience together in one box. Architecting a server is difficult. The physical hardware for a server has all the requirements listed earlier in this chapter, as well as the system engineering and performance tuning that only a highly experienced expert can do. The software required to provide a service often involves assembling various packages, gluing them together, and providing a single, unified administration system for it all. It’s a lot of work! Appliances do all this for you right out of the box.

Although a senior SA can engineer a system dedicated to file service or email out of a general-purpose server, purchasing an appliance can free the SA to focus on other tasks. Every appliance purchased results in one less system to engineer from scratch, plus access to vendor support in case of an outage. Appliances also let organizations without that particular expertise gain access to well-designed systems.

The other benefit of appliances is that they often have features that can’t be found elsewhere. Competition drives the vendors to add new features, increase performance, and improve reliability. For example, NetApp Filers have tunable file system snapshots that allow end users to “cd back in time,” thus eliminating

many requests for file restores.

13.8 Hybrid Strategies

Most organizations actually employ a number of different strategies. Typically a small organization has a few snowflakes and possibly a few eggs in one basket. Medium-size organizations usually have a virtualization cluster plus a few snowflakes for situations where virtualization would not work. Large organizations have a little of everything.

Use what is most appropriate. Small companies often can't justify a grid. For large companies it is best to build a private, in-house cloud. If a company needs to get started quickly, cloud computing permits it to spin up new machines without the delay of specifying, purchasing, and installing machines.

Often one platform is selected as the default and exceptions require approval. Many IT departments provide VMs as the default, and bare-metal requests require proof that the application is incompatible with the virtualization system. Some companies have a "cloud first" or "cloud only" strategy.

Depending on the company and its culture, different defaults may exist. The default should be what's most efficient, not what's least expensive.

The only strategy we recommend against is an organization trying to deploy all of these strategies simultaneously. It is not possible to be good at all strategies. Pick a few, or one, and get very good at it. This will be better than providing mediocre service because you are spread too thin.

13.9 Summary

Servers are the hardware used to provide services, such as file service, mail service, applications, and so on. There are three general strategies to manage servers.

All eggs in one basket has one machine that is used for many purposes, such as a departmental server that provides DNS, email, web, and file services. This puts many critical services in one place, which is risky.

With beautiful snowflakes, there are many machines, each configured differently. Each machine is configured exactly as needed for its purpose, which sounds optimal but is a management burden. It becomes important to manage variations, reducing the number of types of things that are managed. We can do this many ways, including adopting a policy of eliminating one generation of products before adopting a new one.

Buy in bulk, allocate fractions uses virtualization or containers to achieve economies of scale in hardware management but provides a variety of machine sizes to users. Virtualization provides the ability to migrate VMs between physical machines, packing them for efficiency or to move VMs away from failing hardware. Packing VMs must be done in a way that maintains $N + 1$ redundancy.

There are several related strategies. Grid computing provides hundreds or thousands of machines managed as one large computer for large computational tasks. Blade servers make hardware operations more efficient by providing individual units of computer power or storage in a special form factor. Cloud-based computing rents time on other people's server farms, enabling one to acquire additional compute resources dynamically. Software as a service (SaaS) eliminates the need for infrastructure by relying on web-based applications. Server appliances eliminate the need for local engineering knowledge by providing premade, preconfigured hardware solutions.

Organizations use a mixture of these strategies. Most have one primary strategy and use the others for specific instances. For example, it is very common to see a company use virtualization as the default, physical machines as the exception, and SaaS for designated applications.

Exercises

1. What are the three primary hardware strategies? Describe each.
2. What are the pros and cons of the three primary hardware strategies? Name two situations where each would be the best fit.
3. What is the hardware strategy used in your organization? Why was it selected?
4. How does your organization benefit from the hardware strategy it uses? Why is it better or worse than the other two strategies?
5. Why is data integrity so important in the all eggs in one basket strategy?
6. Why won't RAID protect data if there is a fire?
7. What is dependency hell?
8. With beautiful snowflakes, each server is exactly what is needed for the application. This sounds optimal. How can it be a bad thing?
9. In your environment, how many different server vendors are used? List them. Do you consider this to be a lot of vendors? What would be the benefits and problems of increasing the number of vendors? Decreasing

this number?

10. What are some of the types of variations that can exist in a server fleet? Which management overhead does each of them carry?
11. What are some of the ways an organization can reduce the number and types of hardware variations in its server fleet?
12. In what way is buy in bulk, allocate fractions more efficient than the other strategies?
13. What is a server appliance?
14. Which server appliances are in your environment? What kind of engineering would you have to do if you had instead purchased a general-purpose machine to do the same function?
15. Which services in your environment would be good candidates for replacement with a server appliance (whether or not such an appliance is available)? Why are they good candidates?
16. Live migration sounds like magic. Research how it works and summarize the process.
17. If your organization uses virtualization, how quickly can a new VM be created? Request one and time the duration from the original request to when you are able to use it.
18. [Section 13.3](#) says that it can take minutes to create a VM. If the answer to Exercise 17 was longer than a few minutes, investigate where all the time was spent.
19. The cloud means different things to different groups of people. How are these definitions related? Which definition do you use?
20. What can cloud-based computing do that a small or medium-size company cannot do for itself?

Chapter 14. Server Hardware Features

Server hardware is designed with a bias toward performance and remote manageability. It doesn't earn the "server" designation just because it can be mounted in a rack. Server hardware differs from workstation hardware both in form factor and in the options available at time of purchase. It is also different in how we configure it and manage it.

The previous chapter discussed several strategies for providing server capacity to our customers. Most of those strategies involved buying and maintaining hardware rather than using cloud-based services. This chapter is about high-level issues related to server hardware. The next chapter will cover more low-level technical issues.

Server hardware generally falls into one of two major categories: enterprise and cluster.

Enterprise server hardware prioritizes the needs of commercial applications that enterprises tend to use: business applications, email servers, file servers, and so on. These applications tend to assume data integrity and resiliency issues are handled at the hardware layer. This is a polite way of saying they assume these responsibilities are someone else's problem. As a result, these models emphasize reliability, high availability, and performance. Their marketing emphasizes reliability worth paying extra for.

Cluster server hardware prioritizes the needs of cluster or distributed computing applications such as Hadoop, Cassandra, web server farms, and so on. These applications work around hardware failures through software-based resiliency techniques such as service replication with automatic failover. They don't require expensive hardware reliability options such as RAID cards. These models are stripped down to the bare essentials: fast CPUs, network connections, and RAM. Their marketing emphasizes their excellent total cost of ownership: initial purchase price plus lifetime power, cooling, and other operational costs.

This chapter focuses primarily on enterprise servers. Cluster management is the focus of Volume 2 of this book series, especially [Chapter 3](#), "Selecting a Service Platform."

14.1 Workstations Versus Servers

Workstations and servers are fundamentally different. The differences stem from the ways that each is used, which gives us a different set of requirements for

each. For example, servers have higher uptime requirements than workstations, because many people rely on them. Servers have higher data integrity requirements than workstations, because core enterprise data is stored and processed there. Servers have higher CPU and memory requirements than workstations, because they are used for complex data processing, or to provide a service for many people.

As a result, server hardware is different from workstation hardware. Server operating systems are different from workstation operating systems. Also, we manage servers differently, from patching to configuration. We will examine those differences in more detail.

14.1.1 Server Hardware Design Differences

Server hardware is designed with different priorities than workstation hardware. Workstations are designed for an individual user to sit in front of, and interact with directly. They need a display and user input devices, such as a keyboard, mouse, touchscreen, and microphone. Servers are designed to be housed in datacenters and accessed remotely by many users, usually indirectly through the services that they supply. Some of the characteristics that differ for servers versus workstations are profiled here:

- **More CPU performance:** Servers tend to have faster CPUs and more of them. CPUs may be available in a variety of speeds. Typically the fastest revision of a CPU is disproportionately more expensive—this is a surcharge for being on the cutting edge. Such an extra cost can be more easily justified on a server that is supporting hundreds of customers, or is hosting many virtual machines, or is expected to last many years.
- **High-performance I/O:** Servers tend to have more I/O capacity. I/O requirements are often proportional to the number of clients being served. This necessitates faster I/O subsystems. Faster I/O may be achieved by using the latest, fastest technologies that aren't available elsewhere yet. Connections to storage may be multiple dedicated paths, one for each disk, rather than a shared channel. The path between devices is highly optimized so that there are no bottlenecks even at full utilization. This requires special care when designing the system and choosing tradeoffs that favor performance over cost.
- **Expandability:** Servers usually have more physical space inside for hard drives and more slots for cards and CPUs. Adding more slots and the connecting circuitry is expensive. Therefore consumer hardware

manufacturers tend to eliminate them to lower the initial purchase price, whereas they are easily justified on server hardware.

- **Upgrade options:** Servers are designed for growth. They generally have the ability to add CPUs or replace individual CPUs with faster ones. Typically server CPUs reside on separate cards within the chassis, or are placed in removable sockets on the system board for ease of replacement.
- **Rack mountable:** Generic servers and server appliances should be rack-mountable. Servers should be rack-mounted, not stacked or put on shelves. Whereas desktop hardware may have a pretty, molded plastic case in the shape of a gumball, a server should be rectangular and designed for efficient space utilization in a rack. Any covers that need to be removed to do repairs should be removable while the host is still rack-mounted. More importantly, the server should be engineered for cooling and ventilation in a rack-mounted setting. A system that has only side cooling vents will not maintain its temperature as well in a rack as one that vents front to back. Having the word *server* included in a product name is not sufficient; care must be taken to make sure that it fits in the space allocated.
- **Front and rear access:** A rack-mounted host is easier to repair or perform maintenance on if tasks can be done while it remains in the rack. Such tasks must be performed without access to the sides of the machine. All cables should be on the back, and all drive bays should be on the front. Some DVDROM bays open on the side, indicating that the host wasn't designed with racks in mind. Power switches should be accessible but not easy to accidentally bump. Some systems, often network equipment, require only front access. This means that the device can be placed "butt in" in a cramped closet and still be serviceable. Some hosts require that the external plastic case be removed to successfully mount the device in a standard rack. Be sure to verify that this does not interfere with cooling or functionality.
- **High-availability options:** Server hardware should include various high-availability options, such as dual power supplies, RAID, multiple network connections, and hotswap components. RAM should be error correcting, not just error checking.
- **Remote management:** Server hardware should be capable of being remotely managed. For Unix servers and network equipment, this means serial port consoles. For modern PC hardware, it means out-of-band (OOB) management, such as Integrated Lights-Out (iLO) or Intelligent

Platform Management Interface (IPMI) for remote console, power, and baseboard management. In the past IP-KVM and remote power control systems added remote control capabilities to a system after the fact, but it is less complex and costly to have such facilities integrated into the server themselves. Integrated temperature sensors and other hardware monitoring features are also standard on server hardware today.

Don't Run Services on Desktop Hardware

It is tempting to purchase desktop hardware for use as servers. Doing so is short-sighted.

The initial purchase price is often lower, but the longer-term costs will be higher. What is the cost of lost productivity due to slow performance? What is the cost of a multi-hour outage due to a failed hard drive on a system without RAID?

Desktop hardware is not as reliable. It tends to be sold with consumer-grade hard drives and RAM, and lacks reliability enhancements such as ECC RAM or RAID storage. It usually has limited I/O capacity, leading to unsatisfactory performance. It does not mount in a rack, which seems fine at first, but after a few machines have accumulated becomes more difficult to manage. While one can purchase shelves and other devices to make desktop towers fit in a rack, they do not pack as efficiently as rackable server hardware.

Desktop hardware does not include remote management facilities, which means problem diagnostics and anything requiring console access will require physically visiting the machine. Walking to the machine delays when such technical work starts and takes you away from your desk—neither of which is an efficient way of working.

14.1.2 Server OS and Management Differences

How we configure servers is also different. Servers use a different OS and their configurations are managed differently.

Server hardware runs a server OS. Microsoft Windows Server Edition includes additional software for providing services such as ActiveDirectory. It also is tuned for server operation instead of interactive performance: Various defaults are changed to provide better performance for long-running applications. Workstation editions are tuned with a bias for best interactive

performance.

In the Linux world, and more recently with Windows Server Core, the server edition is stripped down to the bare essentials, eliminating even the windowing and graphic subsystems. Functionality is added as needed by installing the appropriate package.

A system is more reliable and secure when it is smaller. Not having a GUI greatly reduces the memory footprint and not activating any video drivers avoids a major source of bugs and crashes. Debugging is easier because what hasn't been installed can't get in the way. Maintenance is easier because there are fewer software packages to be upgraded. Security is improved because a package that hasn't been installed is one fewer source of vulnerabilities.

Often companies have a policy of installing security patches within a certain number of days after the vendor has released the patch. For many servers, this means a constant flow of upgrades and reboots to patch software that isn't even used. By being disciplined and not installing unnecessary software, patch and reboot cycles will be reduced.

Server OSs are often patched on a different schedule. While workstations are updated frequently, often with user approval, servers are patched on a schedule that matches the requirements of the application. That might mean weekly during off-hours, or monthly during carefully announced maintenance windows.

The configuration of servers is also different. Configuration management systems are used to configure and update the system, as was discussed in [Chapter 4](#), “[Infrastructure as Code](#).” Network configuration is generally done by hardcoding the configuration, as described in [Section 5.4](#), though DHCP INFORM is often used to automatically update network settings.

Top Five Server Configuration Mistakes

A number of common configuration mistakes may occur when people forget to clearly differentiate between workstations and servers.

1. Using the workstation edition of the OS, because it happens to be what was installed
2. Using desktop hardware for a server, because it was “free”
3. Using someone's desktop for a shared function, because they've promised never to power off their machine
4. Running on a machine under someone's desk, because if the service is successful, we'll move it to the computer room later

14.2 Server Reliability

All devices fail. We need to accept that fact, and prepare for failure. Because servers need to be more reliable, and have higher uptime than workstations, one of the ways we prepare for equipment failure is to buy server hardware with additional features for reliability and data integrity. When evaluating server hardware, it is important to understand these features, and to know which questions to pose to the vendors. In addition, servers should be housed in a restricted-access, climate-controlled environment, with protected power—in other words, a computer room or datacenter.

14.2.1 Levels of Redundancy

Servers often have internal redundancy such that one part can fail and the system keeps running. For example, there may be two power supplies in the system. Either can fail and the system keeps running. The term **$N + 1$ redundancy** is used when we wish to indicate that there is enough spare capacity for one failure. **$N + 2$ redundancy** would mean there is enough spare capacity for two failed power supplies.

Having two power supplies does not automatically create $N + 1$ redundancy. What if the system is loaded with enough power-hungry cards and components that two power supplies are required just to provide enough watts to run the system? It is possible for this system to be $N + 0$ (no spare capacity) even though there are two power supplies. In such a case, a third power supply would be required to be $N + 1$.

Just because the power supplies are $N + 1$ redundant, it doesn't mean the system can't fail. What if the CPU dies or a hard disk fails? For this reason, CPUs, storage systems, and other components may have $N + 1$ redundancy options, too.

When a salesperson says that a product has no single point of failure, or that the system is $N + 1$ redundant, a good question to ask is "Which parts aren't $N + 1$ redundant?" Every system has some failure point. That's just physics. It is your job to know what it is. That's system administration.

14.2.2 Data Integrity

Another aspect of server reliability is data integrity. How do we know if the data stored on the machine will be available and valid in the future? Hard disks and

SSDs eventually wear out and die. In turn, we must have a plan to deal with this eventuality. Not having a plan is as irrational as assuming our storage systems will last forever.

RAID

Enterprise applications are typically written assuming that storage is perfect—that it is always available and never fails. While this is an irrational assumption, we can make a reasonable simulation of this requirement by using RAID (Redundant Array of Independent Disks) levels 1 and higher. When a disk fails, the system keeps running and we have time to replace the disk.

RAID 1 stores all data twice, once on each disk of a two-disk mirror. If one disk fails, the system can simply rely on the remaining disk. The failed disk is replaced and the data from the surviving disk is mirrored to the new disk, bringing the system back to full resiliency. As long as this happens before the remaining disk also fails, we have a system of constantly available storage. If the second disk fails before restoration is complete, we lose all data and must restore data from backups. However, the probability of a two-disk failure is small if we act quickly.

RAID levels 2 and higher are similar but create redundancy in ways that are more efficient, have better performance, or can survive two disk failures.

RAID 0 is unlike other RAID levels in that it does not aid data integrity. It is actually less resilient to failure, but provides better performance. For the purpose of this chapter, unless specified otherwise, when we refer to RAID we mean levels 1 and higher.

[Chapters 43](#), “[Data Storage](#),” discusses the RAID levels in detail, including how to select which RAID level to use, and the performance implications of each. That said, for a general enterprise server a two-disk RAID 1 mirror is the minimum for a small server, and RAID 5 or 6 is the minimum for a server with a handful of disks.

The brilliance of RAID is that it decouples component failure from service failure. Before RAID, if a disk died, the service died: Dead disk equals outage. You would spend the day replacing the failed disk, restoring data from backups, and apologizing to your customers. It could take hours or days to bring the service back. With RAID 1 and higher, the system can survive a single disk failure. Now a dead disk equals potentially lower performance, but the system keeps running. People would rather have a slow system than no system. With RAID, customers are often unaware that anything has failed.

Non-RAID Approaches

The alternative to RAID is to assume data integrity is handled elsewhere. Plain disks are used with no RAID protection and failures are handled at another layer of the design. For example, suppose a group of redundant web servers all contain the same data, which they receive from a master. The web servers themselves do not need RAID because if a disk fails, the web server is simply shut down for repairs and the traffic is divided among the remaining web servers. After the server hardware is repaired, the data is recopied from the master.

Another example of handling data integrity elsewhere is a distributed storage system such as Google's GFS, Hadoop's HDFS, Cassandra, and others. These systems store copies of data on many hosts, essentially creating RAID-like storage that is distributed among many machines, not just many disks.

A strategy you should avoid is to rely on backups. With this strategy, you copy all data to tape or disk periodically and, if a disk fails, you take the service down while the data is restored from the backup copy. This strategy includes a built-in outage, even if you could restore the data very fast. Backups are important and needed for other reasons—for example, to keep a copy off-site in case the building burns down. RAID is not a backup strategy. RAID and backups are both needed; they are complimentary. Backups will be discussed in greater detail in [Chapter 44](#), “[Backup and Restore](#).”

14.2.3 HotSwap Components

Server hardware has many redundant components, and these components should be hotswappable. **Hotswap** refers to the ability to add, remove, and replace a component while the system is running.

Normally, parts should be removed and replaced only when the system is powered off. Being able to hotswap components is like being able to change a tire while the car is driving down a highway. It's great not to have to stop a service to fix components that fail frequently, such as fans, disks, and power supplies.

Hotswappable components increase the cost of a system. This additional cost is justified when it eliminates downtimes for expansion or repairs.

The more quickly a failed component can be replaced, the better. RAID systems usually run more slowly until a failed component has been replaced and the RAID set has been rebuilt. More importantly, while the system is not fully redundant, you are at risk of a second disk failing and losing all data.

When a vendor makes a claim of hot-swappability, always ask two questions:

Which parts aren't hotswappable? In what way, and for how long, is service interrupted when the parts are being hotswapped? Some network devices have hotswappable interface cards, but the CPU is not hotswappable. Others claim hotswap capability but do a full system reset after any device is added. This reset can take seconds or minutes. Some disk subsystems must pause the I/O system for as long as 20 seconds when a drive is replaced. Others run with seriously degraded performance for many hours while the data is rebuilt onto the replacement disk. Be sure that you understand the ramifications of component failure. Don't assume that hotswap parts make outages disappear: They simply reduce the outage.

Vendors should—but often don't—label components as to whether they are hotswappable. If the vendor doesn't provide labels, you should create your own.

Hot-Plug Versus HotSwap

Be mindful of components that are labeled **hot-plug**. This means that it is electrically safe for the part to be replaced while the system is running, but the part may not be recognized until the next reboot. Even worse, perhaps the part can be plugged in while the system is running, but the system will immediately reboot to recognize the part. This is very different from being hotswappable.

Tom once created a major, but short-lived, outage when he plugged a new 24-port FastEthernet card into a network chassis. He had been told that the cards were hot-pluggable and had assumed that the vendor meant the same thing as hotswap. Once the board was plugged in, the entire system reset. This was the core switch for his server room and most of the networks in his division. Ouch!

You can imagine the heated exchange when Tom called the vendor to complain. The vendor countered that if the installer had to power off the unit, plug the card in, and then turn power back on, the outage would be significantly longer. Hot-plug was an improvement.

To prevent problems in the future, Tom put a big sign on the device that said, "Warning: Plugging in new cards reboots system. Vendor thinks this is a good thing."

14.2.4 Servers Should Be in Computer Rooms

Servers should be installed in an environment with proper power, fire protection,

networking, temperature and humidity control, and physical security. That means a computer room or datacenter—not a spare cubicle or closet.

The two most important points are power and cooling. For reliable operations, servers need power that is reliable and clean. Servers are designed to run at a particular operating temperature, usually around 10 to 35°C (50 to 95°F). Cooling is required to remove the heat they generate.

A rack of equipment standing in the middle of an office is a disaster waiting to happen. It is sharing a circuit not designed for the draw of a rack of power-hungry servers. The cooling in an office generally turns off on nights and weekends. It is not physically secure, and can be accidentally or maliciously interfered with.

It is a good idea to reserve the physical space for a server when it is being purchased. This is a safeguard against having the space double-booked. One can reserve the space using a high-tech datacenter inventory system, or a simple spreadsheet. Taping a paper sign in the appropriate rack is low-tech but sufficient.

After assembling the hardware, it is best to mount it in the rack immediately before installing the OS and other software. Consider the following phenomenon: A new server is assembled in someone's office and the OS and applications loaded onto it. As the applications are brought up, some trial users are made aware of the service. Soon the server is in heavy use before it is intended to be, and it is still in someone's office without the proper protections of a machine room. Now the people using the server will be disturbed by the downtime required when it is moved into the machine room. The way to prevent this situation is to mount the server in its final location as soon as it is assembled. It also reduces the risk that you may lose the rack-mounting hardware. An even worse outcome is to move the machine into the server room, discover you don't have a network cable of sufficient length, and have to move it back. Scheduling a second downtime is embarrassingly unprofessional.

Field offices aren't always large enough to have machine rooms, and some entire companies aren't large enough to have datacenters. However, field offices should at least have a designated room or closet with the bare minimum capabilities: physical security, UPS (many small ones if not one large one), and proper cooling. A telecom closet with good cooling and a door that can be locked is better than having your company's payroll installed on a server sitting under someone's desk. Inexpensive cooling solutions, some of which eliminate the need for drainage by reevaporating any water they collect and exhausting it

out the exhaust air vent, are now commonplace.

14.3 Remotely Managing Servers

Servers need to be maintained remotely. Remote management means that it should be possible to do all system administration tasks involving the machine from a remote location, except physical labor such as adding and removing physical hardware. It should be possible to remotely access the machine's console and, optionally, have remote control of the power switch.

Remote management systems have the benefit of permitting you to operate a system's console when the machine is in a bad state or otherwise disconnected from the network. For example, certain things can be done only while a machine is booting, such as pressing a key sequence to activate a basic BIOS configuration menu. Remote console systems also let you simulate the funny key sequences that have special significance when typed at the console—for example, CTRL - ALT - DEL on PC hardware and L1 - A on Sun hardware.

Remote access is important because servers are often housed in machine rooms located around the world. Requiring hands-on access for daily tasks is inefficient. You shouldn't have to fly to Europe every time you want to see console diagnostics, reboot the system, or reload the OS. Even if the server is down the hall in a server closet, having remote access eliminates time-wasting visits to the machine, and enables you to manage the server while out of the office or from home.

Security implications must be considered when you have a remote console. Often, host security strategies depend on the consoles being behind a locked door. Remote access breaks this strategy. Therefore, console systems should have properly considered authentication and privacy systems. For example, you might permit access to the console system only via an encrypted channel, such as SSH, and insist on authentication by a one-time password system, such as a handheld authenticator.

14.3.1 Integrated Out-of-Band Management

Modern servers have remote management capabilities built-in. Such systems are generically called out-of-band (OOB) management, and have an Ethernet port as an interface. Other terms you may hear (some of which are proprietary technologies) are lights-out management (LOM), Integrated Lights-Out (iLO), Intelligent Platform Management Interface (IPMI), remote insight board (RIB), or Remote Insight Light-Out Edition (RILOE).

Once an IP address is assigned to the OOB interface, one can connect over the network to a remote management subsystem that provides access to the console. OOB solutions usually just require a browser to access the console. However, some require a custom client for access to the full capabilities, so it is worth checking this point before buying a system.

These remote management systems are isolated enough from the main system that they are accessible even when the main system is down or powered off. This feature is often the differentiator between a vendor's server-class machines and other offerings.

Bear in mind that many of these built-in systems have been found to suffer from traditional security holes, such as buffer overruns. It is not wise to assume that setting a password on the OOB interface and using SSL is sufficient protection for such access to your servers. Assume that access to the OOB interface is equivalent to physical access to the machine, and establish additional security measures accordingly. For example, put OOB interfaces on a dedicated protected network. Allow access to this network only via a web proxy with certificate-based authentication, and appropriate authorization.

14.3.2 Non-integrated Out-of-Band Management

Third-party products can add some OOB functionality to systems that lack built-in remote management. These products typically focus on either remote power cycles or remote console access.

Remote Power Cycle

One can gain the ability to remotely power a machine off and on using a switched power distribution unit (PDU). A PDU is the fancy name for a power strip. A switched PDU can individually turn on or off each power receptacle. For example, if the machine has crashed in a way that can be resolved only by turning the power off and back on, one remotely connects to the switched PDU and issues the command to cycle the power of its jack.

Remote Console with IP-KVM

One can gain the ability to remotely access the console of a machine using an IPKVM switch. A **KVM switch** is a device that lets many machines share a single keyboard, video screen, and mouse (KVM). For example, you might be able to fit three servers and three consoles into a single rack. With a KVM switch, you need only a single keyboard, monitor, and mouse for the rack. Now more servers can fit in that rack. You can save even more room by having one

KVM switch per row of racks or one for the entire datacenter.

An IP-KVM is a KVM switch that can be accessed remotely. This eliminates the need for any monitors, keyboards, or mice in the computer room. You simply run a client on your workstation that connects to the IP-KVM. As you type and move your mouse, the client emulates these actions on the machine via the KVM mechanism.

Remote Console with Serial Consoles

Network equipment, appliances, and older servers have serial consoles. They have no video, keyboard, or mouse connector. In the old days one would attach a physical VT-100 terminal to the serial console port of each device.

What Is a VT-100 Terminal?

Younger readers may think of a *VT-100 terminal* only as a software package that interprets ASCII codes to display text, or a feature of a `telnet` or `ssh` package. Those software packages are emulating actual devices that used to cost hundreds of dollars each. One would be purchased for each server to be its console. Before PCs had VT-100 emulators, a mainframe would have dozens or hundreds of serial ports, each one connected to a physical terminal.

Having a large VT-100 terminal for each server would consume a lot of space in the machine room. Often there would be a long table with dozens of terminals. In the 1990s it became popular to replace physical terminals with a terminal console concentrator. This device is one box with many serial ports. The serial cables that previously connected to a physical VT-100 terminal instead plug into one of the ports on this box. If you need to connect to the serial console of a device, you SSH or TELNET to a particular port on the concentrator, and it connects you to the console of the server.

For sites with many servers, network equipment, and appliances that are connected this way, it is common to have console server software, such as Conserver (<http://www.conserver.com>), to manage the serial consoles. This software gives SAs a way to connect to a console by name, rather than having to remember which server is attached to which concentrator on which port. It also stays connected to the serial consoles, preventing other access and logging all console output, giving SAs the ability to review the history of what happened to a machine. SAs connect and authenticate to the console server

software, allowing fine-grained control of who is authorized to connect to which consoles. Also, serial console concentrators usually have the option to require authentication—for example, using RADIUS or TACACS—before allowing someone to connect to a console. However, this access control tends to permit access to all consoles or to none, rather than giving each SA access only to the consoles he or she needs.

Remote console access is discussed further in [Section 26.5](#).

Monitor All Serial Ports

Once Tom noticed that an unlabeled and supposedly unused port on a device looked like a serial port. The device was from a new company and Tom was one of its first beta customers. He connected the mystery serial port to his console and occasionally saw status messages being output.

Months went by before the device started having a problem. He noticed that when the problem happened, a strange message appeared on the console. This was the company's secret debugging system! When he reported the problem to the vendor, he included a cut-and-paste of the message he was receiving on the serial port. The company responded, "Hey! You aren't supposed to connect to that port!" Later, the company's employees admitted that the message had indeed helped them to debug the problem.

After this, Tom adopted a policy of always monitoring all serial ports.

14.4 Separate Administrative Networks

It is common for servers to have a separate NIC that is connected to a dedicated administrative network. Separating administrative traffic from normal service traffic has a number of benefits.

For servers, the primary benefit is often to isolate disruptive traffic. Backups, for example, consume a large amount of bandwidth and can interfere with normal service traffic. Sometimes servers will also have a dedicated backup interface that is connected to a high-speed network dedicated to backups.

In addition, the administrative network is often more stable and static, while the service network is more dynamic. For example, the service NIC of a server might reconfigure itself frequently as part of a load-balancing and failover mechanism. Monitoring and control of such mechanisms therefore is often done via the administrative NIC of a server so that it doesn't step on its own tail.

The administrative network often has more restrictive firewall policies than the service network, since it has more critical systems. The administrative network is often where one connects other administrative devices and control systems. For example, UPSs, PDUs, IP-KVMs, temperature and environmental monitors, and other devices now connect to the network via Ethernet for remote access, control, and monitoring. The administrative network is also where a server's iLO/IPMI interfaces connect.

This separate network should be engineered to use separate equipment so that it will not be affected by outages in the main network. Often this network is engineered very simply, with old-fashioned managed switches, no VLANs, and a simple topology. This network then provides a way for SAs to get to machines and network devices during a network outage.

14.5 Maintenance Contracts and Spare Parts

When purchasing a server, consider how repairs will be handled. All machines eventually break. Expecting them not to break is irrational. To be rational we must plan for the eventual hardware failure.

A maintenance contract is an agreement with a vendor that it will fix or replace hardware and provide other support for a certain duration of time. Maintenance contracts are like an insurance policy; you pay and hope you don't need it, and so does the vendor.

14.5.1 Vendor SLA

Vendors tend to have a variety of maintenance contract options, with different service level agreements (SLAs). For example, maintenance contracts may offer to ship a replacement for a bad part with a 4-hour response time, a 12-hour response time, or next-day options. Sometimes the options include 24/7 response, but other times just specify some number of business days. Other options include having the customer purchase a kit of spare parts and receive replacements after a spare part gets used. Usually you have to install the replacement part, though vendors offer on-site repairs for a higher price. Sometimes the service offerings vary by country or region, depending on where the vendor has local presence.

Response time is often tricky to calculate. While you might think a 4-hour response time contract means that a hard drive that fails at 7 AM will be replaced by 11 AM, the truth is quite different. For most vendors the clock starts ticking not when the part fails, but when the vendor has confirmed that the part

is dead. Suppose a machine fails at 7 AM, and you don't notice it is down until you arrive at the office at 9 AM. You know it is down because you are greeted by an angry mob. It may then take 30 minutes to get through to tech support and convince them that, yes, the machine is on the contract. Running diagnostics may require an hour. It may be 11 AM before the vendor has agreed to send the replacement part. At that point you may learn that the spare part depot in your city has the part and a courier is on the way. Or, you may discover that the spare part is not available and must be shipped overnight from the factory. Obviously, this process can be improved: The outage should have been detected by the monitoring system, not by the angry mob. The point here, however, is that the definition of the 4-hour response time is defined by the contract, not by the clock on the wall.

The faster the response time, the more expensive the maintenance contract. Select the most appropriate option for the type of server. If a server is noncritical, the default maintenance contract may be sufficient. Replacement parts are ordered as needed under this arrangement, perhaps with rush shipping. It may require an open purchase order in place to expedite the ordering.

Ironically, a highly critical server can be on the lowest-level maintenance contract. If it is truly impossible to live without this server, it should be part of a mirrored pair that automatically fails over to a hot spare if there is trouble, or at least a cold spare should be sitting beside it ready to be activated. Therefore such a system does not require an expensive, 4-hour-response-time maintenance plan. A parts replacement plan may be sufficient.

For some applications, a critical server may be too big and expensive to have a replica. In this case, a 4-hour-response-time contract is warranted. Some vendors can guarantee that spares will be kept in a depot in the same city. The SLA for the service provided by the machine must include that for major repairs, downtime may be multiple hours.

Some applications can use inexpensive servers that are replicated many times—for example, a web farm of ten machines, each performing the same function. The system has enough spare capacity that any one machine can be down and the system will be fine. In this case, a next-day or two-day response time is reasonable because the likelihood that a second machine will die in a two-day period is small. Where this model can be applied, it is the most cost-effective.

14.5.2 Spare Parts

In a large environment, the cost of many maintenance contracts may be more

expensive than staffing your own repair department. This in-house department can receive training and certification on the hardware and maintain its own inventory of spares.

Some vendors have a self-support plan where the customer purchases a spares kit and the maintenance contract is simply an expedited replacement plan for any spares that are consumed. This arrangement is typically used in high-end servers and appliances where the customer only has a few, critical units from that vendor. The spares kit is less expensive than buying a second machine, often called a **cold spare** because it is left powered off (cold) until needed. A hardware spares kit is less expensive because it does not require a software license, and because it has fewer parts. It does not include things that typically can't break, such as the chassis itself. It needs to include only one of any given part: If the original system has four CPUs, the kit needs to contain only one. Similarly, a single spares kit should be sharable among all machines it applies to. One needs to get additional spares kits only as new models are introduced into the fleet.

On-Site Spares and Staff

In the 1990s Bell Labs had so many machines from Sun Microsystems in its Murray Hill building that it negotiated to have the regional Sun maintenance person's office be in its building. Bell Labs provided the office space for free, and in exchange received many intangible benefits. Response time to the company's calls was faster since there was a good chance the Sun worker was already in the building. Since his spare time was spent at Bell Labs, he was often around to help out with issues outside of the contract, such as installing new machines. His office also became a regional spare parts depot. Common parts were in the building, reducing the time to repair.

14.5.3 Tracking Service Contracts

Sometimes during an outage we discover that the machine is not on the service contract. The solution usually involves talking to a salesperson, who will often have the machine repaired on good faith that it will be added to the contract immediately or retroactively. There are a few different strategies for improving on this approach.

It is good practice to write purchase orders for service contracts for 10 percent more than the quoted price of the contract, so that the vendor can grow the monthly charges as new machines are added to the contract.

Put machines on the contract at the time of purchase. Make sure there is a choke-point that purchases pass through where someone makes sure that any purchase without an accompanying service contract is rejected. The person assigned this task might be someone in the purchasing department, or the salesperson whom the vendor has assigned to your account. Provide guidelines for helping the person select the appropriate service level, or have a default.

Often the first 12 months of service are built into the cost of the device, and one must remember to add the service contract after that time. Prearrange that the device will be added automatically at that time.

Some vendors require you to purchase your maintenance contract in advance. For example, they may offer service for three years and require all three years to be purchased when the machine is acquired. In this case, it is important to track when the contract expires so that an extension can be negotiated, the machine can be replaced, or the machine will be designated for use by noncritical services.

Lastly, track the service contracts as part of the machine inventory. This way you can run reports to find machines missing from the contract.

The other side of this potentially problematic process is forgetting to remove machines from the service contract when they are decommissioned. It is also good practice to review the service contract annually to ensure that new servers were added and retired servers were deleted. Strata once saved a client several times the cost of her consulting services by reviewing a vendor service contract that was several years out-of-date.

Again, tracking service contracts as part of inventory is important to make this happen.

14.5.4 Cross-Shipping

Something else that's good to see in maintenance contracts is the ability to have replacement parts cross-shipped.

When a server has hardware problems and replacement parts are needed, some vendors require the old, broken part to be returned to them. This makes sense if the replacement is being done at no charge as part of a warranty or service contract. The returned part has value since it can be repaired and returned to service with the next customer that requires that part. Also, without such a return, a customer could simply be requesting part after part, possibly selling them for profit.

Some vendors will not ship the replacement part until they receive the broken

part. This practice significantly increases the time to repair. Better vendors will ship the replacement immediately and expect you to return the broken part within a certain amount of time. This practice is called **cross-shipping**; the parts cross paths as they are delivered.

Cross-shipping is generally included in a maintenance contract. If not, a credit card or open PO might be required. Be wary of vendors claiming to sell servers that don't offer cross-shipping under any circumstances. Such vendors aren't taking the term *server* very seriously.

14.6 Selecting Vendors with Server Experience

Some vendors have years of experience designing servers, and it shows. They build hardware with the server-related features listed earlier, as well as include little extras that one can learn only from years of market experience.

From time to time companies with a history of making consumer products jump into the server market and their foray is a disaster. They take their existing models and slap a “server” name onto it. The product does not integrate with enterprise systems such as network directory services. Management of the device is manual and cumbersome, lacking the ability to manage many servers at once, and with no kind of remote management. There are no maintenance contracts available other than what was offered with the vendor's consumer products. No on-site support or parts by mail means in the event of a hardware failure, the entire system will have to be packed up and carried to a local electronics store. Technical support is provided via phone to people accustomed to dealing with consumer-grade products and includes only consumer-grade responses. The support personnel can't handle highly complex technical issues, nor can they offer solutions beyond “wipe and reload”—an option that may be fine for a laptop, but is insufficient for fixing a business's primary server.

Select vendors that are known for building reliable hardware. Some vendors cut corners by using consumer-grade parts; others use premium-quality parts.

The difference between consumer- and premium-grade parts is often one of testing. RAM manufacturers do not have one assembly line that makes consumer-grade RAM chips, and another that makes premium-grade RAM chips. They have one factory that makes RAM chips. If a chip passes all the quality assurance tests, it is sold as a premium-grade product. If it passes most of the quality assurance tests, it is sold as a consumer-grade product at a lower price. If it doesn't pass enough tests, it is sold as scrap.

The MIL-SPEC Scam

Some vendors make a big deal out of the fact that they use MIL-SPEC parts. The implication is that these parts are higher quality. That is misleading.

MIL-SPEC is the U.S. military's requirements specifications for parts. MIL-SPEC does not require parts be of high quality; rather, it specifies the required quality and tolerances for deviation. In other words, what is on the label has to be true. What MIL-SPEC really means is that the manufacturer is more certain that the item being purchased will be the quality expected, whether that is high or low quality.

That said, manufacturing is improved because the vendor can spend less time worrying about the variations in the parts being used, which allows it to focus on overall quality.

14.7 Summary

This chapter discussed what makes server hardware different from workstation hardware. It is not just the fact that the hardware fits in a rack or has “server” in its name.

Server hardware is engineered with a bias toward performance. *Enterprise* hardware tends to emphasize reliability through hardware, improving reliability so that applications do not need to concern themselves with handling hardware failures as much. *Cluster* hardware tends to emphasize reliability through software, reducing the cost by eschewing expensive RAID controllers and other hardware.

Server hardware is different in that it tends to emphasize CPU and I/O performance. It is more expandable and upgradable. Maintenance is made easier by physical design, such as no side access required, and remote management features such as IPMI.

Server hardware is configured differently. Operating systems have special server editions that add server features and remove client features.

$N + 1$ redundancy means that any one part of a system can fail and the system keeps running, although performance may be degraded. This is a common feature of storage, power, and network designs.

Servers should be housed in server rooms with proper cooling and power. They should be remotely managed, so that maintenance doesn't require a walk to the machine room or a flight to a datacenter.

Servers generally have maintenance contracts that include a defined response time and the ability to cross-ship spare parts.

Exercises

1. How is server hardware different from workstation hardware?
2. How are servers configured differently than workstations?
3. What is a maintenance contract? What are its major elements?
4. What is $N + 1$ redundancy?
5. Does having two of the same component in a system automatically make the system $N + 1$ redundant? (Hint: There are two very different reasons for having two of a given component.)
6. If a vendor said that a device's cooling fans were $N + 1$ redundant, what would that mean?
7. What is data integrity and what are some ways to achieve it?
8. Why do servers need to be managed remotely?
9. What is the difference between components that are hot-plug and hotswap?
10. Why would someone want hotswap parts on a system that didn't have $N + 1$ redundancy?
11. Why would someone want $N + 1$ redundancy if the system does not have hotswap parts?
12. Why don't workstations typically use RAID?
13. What are some of the ways servers can be remotely managed?
14. In your environment, what are the major and minor differences in how workstations and servers are configured?
15. How are servers and devices in your machine room remotely managed? Which devices cannot be remotely managed?
16. Which critical hosts in your environment do not have $N + 1$ redundancy or don't allow hotswapping parts? Estimate the cost to upgrade them to be $N + 1$.
17. Select a machine in your environment that provides a service such as email, DHCP, web access, or some other application. Describe how it is configured differently than a laptop or desktop in your environment based on the criteria used in this chapter.

- 18.** Describe your site's strategy for purchasing maintenance and repair contracts. How could it be improved to save money? How could it be improved to provide better service?
- 19.** Pick an important machine in your IT organization and describe the details of its maintenance contract, or how broken parts would be replaced.
- 20.** Select an important server in your environment and describe the data integrity features of the hardware used. Does it use RAID? Which parts are hotswappable or hot-pluggable?
- 21.** A server is low on disk space but has drive bays available. The SA chose to wait until the next maintenance period to add a disk rather than do it while the system was running. Why might this be?

Chapter 15. Server Hardware Specifications

This chapter is about the decisions one makes when purchasing an individual server. Designing hardware is all about making different tradeoffs. Exactly what vendors offer changes from month to month and from year to year. Nevertheless, certain principles remain constant.

Vendors have different product lines. Each is biased toward cost or performance in different ways. If we can understand what went into the designer's decisions, we can be smarter customers.

Within a product line there are many hardware choices: CPUs, RAM, storage, and so on. Fundamentally we should begin with the application requirements in mind and use those requirements to guide each decision, including the amount of RAM, the amount of storage, the number and types of CPUs, and so on. Even if we are adding general-purpose compute power to a virtualization cluster before we know which actual applications will be running, the requirements can be specified in general terms of what virtualization clusters need.

Bear in mind that with each different type of model you purchase, the support overhead becomes higher. For most companies, it is better to settle on a couple of standard server models than to try to optimize each server for the particular application that will be running on it. We do not want to end up with a blizzard of beautiful snowflakes.

Server performance is very complex, but in general an application is usually waiting for one of four things: disk, CPU, memory, or network. If we had perfect knowledge of our system, it would have just enough of all four without any wasted capacity. Unused capacity is money spent but not used. However, the reality is that we don't have perfect knowledge of our systems, and we can't purchase these resources in highly granular increments. Nor would we want to: Spare capacity is needed for the occasional bursts of activity, plus it offers room for future growth.

Instead, we usually generalize the application as being bound by one or two of the four limiting factors and purchase a server that is optimized in that aspect. Database software is often limited by the speed of disk I/O, so we start our purchasing decision-making process by looking at models with the fastest disk subsystem. One application may require large amounts of RAM, all other factors being inconsequential. Another application may be a CPU hog, churning on data for hours before spitting out a small result.

15.1 Models and Product Lines

Most vendors have multiple product lines. They are similar to the workstation product lines described in [Section 6.1.3](#): one that emphasizes lowest initial cost or purchase price, one that emphasizes lowest total cost of ownership (TCO), and another that emphasizes high performance.

The line that emphasizes initial purchase price is often called the “value line” and trades expandability to achieve a lower initial purchase price. Like its workstation analog, this line is appropriate only when saving money now is more important than long-term manageability. It might be appropriate for a small company with a few employees that does not expect to grow. For example, a small law practice might have one lawyer and one legal secretary—the same structure it has had for years and will have in the future.

The line that emphasizes TCO has models that may cost more initially, but save money in the long term. These machines have higher performance and additional expansion capabilities that prevent obsolescence. They also have management features that become more important when one has many machines, especially if they are remote. This line is appropriate for a company large enough to have dedicated IT staff who maintain the fleet as a unified whole.

The line that emphasizes performance provides access to the latest technologies, such as next-generation CPUs, or amounts of RAM that seem ludicrous by today’s standard but will be normal expectations in a few years. Vendors also have specialty product lines for vertical markets, such as high-end graphics, numerically intensive computing, carrier-grade equipment (suitable for use in telecom environments), and so on. You pay a premium for early access to such technology. This is appropriate for demanding applications such as genomics, large database applications, numerical simulations, and other applications that require high performance.

15.2 Server Hardware Details

There are many hardware options available. While the specific sizes, quantities, and speeds might change over the years, some of the basic decisions stay the same. To make the best purchasing decision, you need to understand the technical details of each component, its limitations, and how the components interact with one another.

15.2.1 CPUs

The most fundamental decision in selecting a server is the CPU—in particular, the choice of a few high-speed cores versus many slow cores. Depending on the architecture of the software that will run on the machine, this decision can determine whether your service is fast and efficient or underutilizes the capacity of the machine.

In the old days, typical computers had a single CPU and we could count on vendors offering faster CPUs each year. Moore’s law predicts that CPU performance will approximately double every 18 months, permitting either the same CPU to be purchased for half the price, or a CPU that is twice as fast to be purchased for the same price.

More recently, this trend has reached certain physical limits. With no other option available, CPU manufacturers started putting multiple CPU cores on the same chip. Instead of a CPU that is twice as fast, you can get a single CPU with two CPU **cores**, each able to run a program or execution thread in parallel.

In aggregate, the computing power has doubled, but any single-threaded program will run more slowly. There is a difference between being able to run one program twice as fast and being able to run two programs at the same time.

Multiple CPUs and Cores

Your purchasing choice is often between a few fast cores or many slower cores. For example, when this book was being written, a Dell R730 server could be outfitted with the CPU options listed in [Table 15.1](#).

Processor Model	CPU Cores	CPU Frequency	Aggregate Speed (Cores × Speed)	Retail Price (Dell, Aug. 2015)
E5-2699 v3	18	2.3 GHz	41.4 GHz	\$8,900
E5-2698 v3	16	2.3 GHz	36.8 GHz	\$7,040
E5-2697 v3	14	2.6 GHz	36.4 GHz	\$5,850
E5-2690 v3	12	2.6 GHz	31.2 GHz	\$4,560
E5-2660 v3	10	2.6 GHz	26.0 GHz	\$3,130
E5-2667 v3	8	3.2 GHz	25.6 GHz	\$4,490
E5-2643 v3	6	3.4 GHz	20.4 GHz	\$3,360
E5-2637 v3	4	3.5 GHz	14.0 GHz	\$2,160

Table 15.1: Intel Xeon Processor E5-2600 v3 Product Family

If an application could utilize all of the cores, the fastest result would be achieved with the processor that has the most aggregate horsepower. In [Table](#)

[15.1](#), the model E5-2699 v3, with 18 cores at 2.3 GHz each, has nearly three times as much aggregate speed as the model with the fastest individual cores.

In contrast, if the application was single-threaded, the fastest result would come from using the model with the fastest core—in this example, the 3.5 GHz cores on the E5-2637 v3. It has single-CPU performance more than 50 percent faster than the E5-2699 v3. While this model has four cores, the application would use only one of them. The other three would either be idle or be used by other processes on the machine.

A server might also have multiple CPU sockets; a low-end server might have a single socket; a high-end server might have 12 or more. Most computers are limited in that each CPU must be of the same kind and speed. One can add capacity only by adding matching CPUs to empty sockets, or replacing all the CPUs with faster ones. The Dell R730 has two sockets; the second can be empty or be filled with an exact match of the first CPU module.

Multitasking and Multithreading

With a single CPU, an operating system can seem to run multiple programs at the same time, but this multitasking is only an illusion. A CPU can really run only one program at a time. Since each program is sharing the CPU, each program runs for a few milliseconds, and then the OS puts it to sleep and runs another program. Each time one process is put to sleep so that another can run is called a **context switch**. This cycle, when repeated, permits many programs to share a CPU. The illusion is complete.

If two programs share the same CPU and require 10 seconds to run each, the total run time will be about 20 seconds. If one program pauses to wait for a disk read or other I/O, the other program will have full use of the CPU. This can reduce that 20 seconds significantly.

With multiple CPUs each program can be assigned to a different CPU and they actually do run in parallel. In the earlier example, the total run time for the two programs, each using its own CPU, will be about 10 seconds as expected, depending on overhead and other factors. If there are more processes than CPUs, sharing begins again.

Programs can also spawn multiple threads of execution. Some web server software is designed to create one thread for each incoming web request. The OS spreads these threads over all the available CPUs—a practice that better utilizes the multiple CPUs.

However, if two or more threads need access to the same resource (for example, the same variable), it is risky for them to do so at the same time. If one is changing the variable, the other may read it in a half-changed state. To eliminate such conflicts, a program locks the variable, accesses it, and then unlocks the variable. Only one thread can hold a lock at a time; the others wait until the lock is released. Threaded programs can spend a lot of time waiting on locks, which reduces efficiency. For example, a program with eight threads running on an eight CPU machine may utilize less than the full resources of the machine.

Threading is explained in greater detail in Volume 2 of this book series in [Chapter 5](#), “Design Patterns for Scaling.” We also recommend that you search for articles on the web about Amdahl’s law, which is a more formal way of determining whether an application would benefit from increased parallelization.

Example Applications

In this section we discuss various applications and identify whether they would benefit from having a greater number of cores overall versus a smaller number of faster cores.

A single-threaded legacy application would run fastest on a single fast core. Any additional CPUs would go mostly unused. We say “mostly” because there most certainly will be auxiliary programs running concurrently: monitoring tasks, backups, periodic maintenance tasks, cron jobs, and and so on.

High-throughput web server software systems usually manage a pool of threads, assigning each incoming HTTP request to a different thread. Such software will get the most aggregate throughput on a machine with many cores, even if they are relatively slow. Conversely, some varieties of web server software are engineered to provide high throughput on a single fast core. Check the software’s documentation.

Graphics computation may or may not benefit from many CPUs. There are too many factors to establish a general rule. One should check with the developers to learn how the software was designed, then verify their statements by running your own timing experiments or benchmarks. For example, one application might take advantage of parallelism inherent to the calculations being performed and be designed to use many CPUs; it would benefit from the aggregate performance. Another application may be single-threaded and run best on a single fast CPU. But what if this single-threaded application is used many

times over? For example, suppose the application was used to render each frame of a film. If unrelated frames can be rendered in parallel, the total throughput may be optimized with many cores. Again, benchmarks will reveal what is best.

A virtualization cluster generally runs better with many cores. VMs can be allocated dedicated cores or they can share cores. Dedicating a core to a VM guarantees a certain performance level for the VM. However, what makes VMs more efficient is to let the virtual machine manager (VMM) dynamically allocate cores to VMs. The amount of CPU power a VM needs changes over time, and the VMM dynamically schedules resources to make the best use of the system. Suppose a VM is allocated four virtual CPUs (vCPUs) but is using only two of them. The VMM can notice this and allocate the extra CPUs to another VM that is in need. Later, if the first VM starts running heavier tasks, it might receive more CPU time. Allocating more resources than exist is called **oversubscription**. For example, allocating 16 vCPUs to 8 real CPUs is a 2:1 oversubscription. This may be sufficient for applications such as VDI, where the typical VM spends most of its time idle. In contrast, a CPU-bound numerical analysis application will suffer if it shares CPUs.

If the virtualization cluster has mixed use, a hybrid solution may be best. For example, suppose the cluster is used for 100 VDI users (light-weight users who can share CPUs) plus dozens of application servers (heavy-weight use that needs dedicated, high-speed CPUs). The physical machines that make up the cluster should include a combination of many-core and faster single-core configurations.

Other CPU Types

The most commonly used family of CPUs is the x86-64 architecture, but other systems do exist. The popularity of the Sun Microsystems SPARC architecture is waning but the ARM architecture is on the rise.

The ARM architecture is particularly interesting because it has extremely good power efficiency. Because of this, the ARM chips dominate the smartphone market. However, since the 64-bit ARM architecture was announced, servers based on ARM have become a possibility.

ARM is incompatible with x86, so it doesn't run operating systems or software that were not specifically ported to the ARM architecture. As of this writing, Microsoft Windows is not available for ARM. Sites that have experimented with ARM chips use Linux and compile their own software.

On the one hand, ARM servers tend to have lower performance than x86

chips, so you need more of them to make up the disparity. On the other hand, they use less energy, and may be a net win overall. It depends on the application. Studies have benchmarked the two options to determine which is most cost-effective when considering all the costs of ownership: initial cost, energy consumed by the chips, energy consumed for cooling, floor space consumed, and so on. For large-scale web applications, ARM-based servers often win. Over time this once-niche architecture will likely grow to be a dominant player.

15.2.2 Memory

Another critical component to consider in server selection is memory. Random access memory (RAM) is where running programs and their data are stored. A server needs to have enough RAM to support the applications that are running, plus any RAM used by the OS and any support programs that are also running.

Until the mid-1990s CPUs and RAM were about the same speed. The RAM system could feed a CPU with data at full rate. In the mid-1990s this changed. CPU speed outpaced RAM speed. CPUs were starved for data because the RAM subsystem could not keep up.

At the same time the entire system got so fast that the latency (the time it takes information to travel from one place to another) between the CPU and the RAM started to dominate system performance. Latency can't be improved because it is limited by the speed of light, or electrons, and nobody has figured out to make either of those move faster. Therefore, as CPU and RAM speed improved but latency stayed the same, performance suffered because the CPU could not be fed data fast enough.

To improve this situation, CPU designers created a number of solutions such as various caching architectures and NUMA.

Caches

One way CPU designers overcome the problem of latency is through caching. A cache stores frequently used or soon-to-be used data close to the CPU for faster access. It uses a small amount of high-speed (expensive) RAM to improve the performance of the main system's lower-speed (less expensive) RAM.

In an enterprise environment, with shrink-wrap software, the main thing to know about the CPU caches is the cache per core ratio. The more cache there is per core, the better the performance.

To understand how caches improve performance, it is necessary to look into some of the details of how computers operate at the most basic level. A block of

memory is called a page. The pages of memory in active use by a process are called the working set, or the process's memory footprint. For best performance, the working set of a process must fit entirely in the cache. Otherwise, the cache is always evicting pages to bring in new ones.

Modern processors have multiple levels of caches, called Levels 1, 2, 3, and 4, or L1, L2, L3, and L4, respectively. Advances in CPU development can result in additional cache levels.

The L1 cache is on the core itself. It is dedicated to a particular core, and becomes the core workspace. It is usually very small (a few kilobytes). There is typically very little variation in L1 cache sizes among different CPUs.

The L2 cache is near the core, often shared among cores that are on the same chip. The L1 cache feeds from the L2 cache. It is larger than the L1 cache but smaller than the L3 cache. L2 cache sizes vary slightly between CPUs. The variation may be important to developers who are optimizing code for a particular CPU, but not so much for enterprise SAs who have to run shrink-wrap code.

The L3 cache is shared among all CPUs of the machine, and often with other functions that are part of the chip package. Where an L4 cache exists, it may take on this role, leaving the L3 cache to be dedicated to the CPUs. The L3 cache can have segments dedicated to particular CPUs, graphics processing units (GPUs), or other functions. The L2 cache feeds from the L3 cache. The L3 cache is larger than the L2 cache, and there is significant variation between processors in the size of the L3 cache. This is the number that you typically see quoted in the CPU specifications. What you want to look at here is the amount of cache per CPU core.

The L4 cache, for CPUs that have it, is also incorporated into the CPU package. It feeds the L3 cache. It is larger and slower than the L3 cache. Currently, L4 caches are quite new and rare, but we expect that situation to change rapidly.

Caches improve performance but add complexity. When two cores access the same area of RAM, copies of data may appear in two different caches. If one core writes to its RAM, the data in the other cache is invalid and must be removed so that the other core is not accessing obsolete data. Doing this operation accurately without reducing performance is very complex.

In a multitasking environment, anytime the OS switches from one process to another (a context switch), the L1 cache is unlikely to have the data needed by the new process. The program will run slowly until the appropriate data is paged

in. A large L2 or L3 cache helps this happen faster. If these caches are large enough, it is possible that they might contain the working sets of two different processes. A similar situation occurs when two VMs share the same CPU. If switched between two processes or VMs rapidly, the value of the cache may be negated unless the working set of each fits in the L3 cache.

This kind of performance issue may not surface until you begin testing real workloads. When performing such tests, try running the same workload with larger and larger amounts of data. You will eventually see performance stall or plateau when the amount of data reaches a certain size. This is an indicator that the working set has exceeded the size of the cache. Use this information to tune the system so that the working set stays within the cache. Developers can use this information to restructure their code to have a smaller footprint. For example, the information could be read by row instead of by column.

Over time, caches get larger and more sophisticated. By the time this book is printed, new generations of chips will undoubtedly be shipping with larger, more sophisticated caches. More levels may be added, some may be removed, and which levels are shared or not shared by cores may change.

NUMA

Another way that CPU designers have improved RAM latency is by introducing Non-Uniform Memory Architecture (NUMA) RAM systems.

In the old days, all CPUs had equal access to all RAM in the system. The speed at which a CPU could access any page of RAM was the same, or uniform. RAM was one big pool to which everyone had equal access.

To improve RAM latency, chip designers created a new RAM architecture. Each partition, or bank, of RAM would be very close to a particular CPU. The other CPUs could access the bank of another CPU, but such access would be slower or at higher latency. Access speed to RAM was no longer uniform; it depended on which bank of RAM was being accessed. There would always be a bank of RAM that was more local, faster to access, for each particular CPU.

Designers modified operating systems to take advantage of this improved access speed. The OS would coordinate between the CPU scheduler and the virtual memory system such that a process running on a particular CPU had its memory stored in that CPU's memory bank. When everything came together, performance was stellar. In fact, this development gave Silicon Graphics (SGI) a huge advantage in the scientific computing arena in the mid-1990s.

This kind of process scheduling is quite complex. In the past, if a CPU was

idle, moving a process to it was easy. Now, to optimize performance, all of its data has to be copied to the receiving CPU's bank, too. Moving a process from one bank to another is very expensive. Avoiding moves, especially for complex workloads, is as difficult as predicting the future.

Once an obscure feature of high-end systems, NUMA is now commonplace in server hardware and server operating systems. Both Windows Server and Linux optimize for it.

When specifying the amount of RAM for a new server, one must take into account whether NUMA is in use. The processes that will be running on each core must fit within the bank size. Usually all banks must be the same size, and there is a minimum size for each. Therefore, an 8-core server that requires at least 16 GB of RAM for each core cannot be configured with less than 128 GB of RAM. If the processes that will run on each CPU require more than 16 GB of RAM, performance will suffer as the OS tries to move processes between CPUs, or simply gives up and permits a CPU to run a program from a distant bank.

Swap Space

Swap space is a feature of an OS's virtual memory system whereby if the system runs out of RAM, the OS can overflow to disk. The disk space used for this is usually a preallocated partition or file known as swap space. When the system runs out of RAM, the least recently used pages of RAM are copied to swap space and those pages are freed for use by programs that actively need RAM. The next time that swapped-out page of RAM is touched, the OS copies it from disk into memory, possibly expelling some other page of memory to disk.

Disks are approximately 100 times slower than RAM, so the process of pausing a program to bring a swapped page in from disk is a performance nightmare. However, swap space is like an auto's safety air-bag: You never plan to use it, but it is good to have during an emergency.

Swap space is not a purchase-time option. Normal disk space is used. Reconfiguring a machine to use more or less swap space can be done dynamically in most operating systems.

Most servers don't use virtual memory because performance while swap space is used is terrible. However, an application crashing because the system ran out of memory is even worse. It is a good idea to allocate a small amount of swap space to handle the rare situations where swap space will prevent a crash.

Swap Space at Stack Overflow

Stack Overflow's default policy for Linux and Microsoft Windows servers is to allocate 4 GB of swap space, but to treat actual use of the swap space as an error condition. The company's monitoring systems raise an alert if swap space is used. These alerts are analyzed to determine if this was a one-time thing or a warning sign that the machine needs more RAM.

15.2.3 Network Interfaces

Servers often have multiple network interface cards (NICs) because they are connected to different networks, or because they are ganged or grouped together for increased bandwidth or resiliency. A server needs network bandwidth proportional to the number of clients accessing it. A server with many clients needs more bandwidth than one with fewer clients, assuming all clients generate the same amount of bandwidth. A server with 100 active clients, each with 1 Gbps network connections, could in theory receive 100 Gbps of traffic, but most likely there will be other bottlenecks in the network that prevent this situation from happening. That said, in general servers require faster network connections than the clients they serve.

We already discussed the concept of connecting servers to separate service, backup, and administrative networks in [Section 14.4](#). Servers may also have separate fiber-optic interfaces for attaching to dedicated high-speed storage networks, both to speed up data access and to avoid having high-volume data access interfere with the service traffic.

Sometimes multiple NICs are grouped together to provide more bandwidth or more resiliency. For example, two 10 Gbps NICs might connect to the same switch, providing 20 Gbps aggregate bandwidth. This practice is often called bonding, or teaming, the available bandwidth. Alternatively, the same two NICs might each be connected to different switches so that if either switch fails, traffic still flows. This improves the resiliency, or survivability, of the system.

The number, speed, and configuration of NICs required depend on the functions performed by the server and the architecture of your datacenter network.

15.2.4 Disks: Hardware Versus Software RAID

Disks, particularly those with moving parts, are the most failure-prone components in most systems. In the previous chapter we discussed using RAID for data integrity. RAID can be achieved through hardware or software. You need to understand the difference before making a purchasing decision.

Software RAID

Software RAID is part of the operating system. This setup provides RAID via device drivers. The benefit of this approach is that it is inexpensive, because it is included in the price of the operating system.

The downside is that it is usually not as fast because it is running on the same CPUs as the rest of the system. RAID Levels 5 and 6 require many CPU cycles to calculate parity.

Software RAID often has fewer features. For example, it may support only RAID 1 mirrors. Older RAID software did not support RAID on the boot disk, because of the Catch-22 involved in loading the RAID drivers from the boot disk. This problem has been fixed since Windows Server 2008R2 was launched and in most Linux distributions since 2013.

Software RAID on the boot disk brings a certain amount of complexity when a disk fails. Generally the system must boot off half of the RAID mirror to get started. If that is the disk that failed, the BIOS may not be smart enough to try the second disk. Typically you have to manually reconfigure the boot drive to be the remaining good disk.

For this reason, it is a good idea to test the failure and recovery situations before the system goes into production. Keep good notes on what worked and didn't work, and what was required to recover the system after a disk failed. It is better to know the limits of the system ahead of time than to discover them during an outage.

Hardware RAID

With hardware RAID, RAID is implemented in a hardware device, usually a card that is plugged into one of the expansion slots of the server. The hard disks plug into this device, and it manages the disks as one or more RAID groups. The server sees each RAID group as a hard disk. It has no idea that the hard disk presented to it is actually a bunch of disks in a RAID configuration. That is all done behind the scenes.

Hardware RAID usually provides higher performance and more features than software RAID. The typical hardware RAID controllers implement a wide range of RAID levels. They usually have dedicated hardware that performs parity calculations.

The downside of this approach is the higher cost. RAID controllers can be very expensive. For a simple system that does not demand extreme performance but simply needs the basic protection of a mirrored disk, hardware RAID may be

overkill.

The term “hardware RAID” is slightly misleading. A RAID controller has plenty of software: It is just running on the controller itself. Because it is a self-contained system, it does not hog the computer’s CPU and memory as software RAID does. The hardware itself is specially designed for performance. For example, each disk usually has its own data path so that multiple disks running at the same time do not compete for resources. Such optimizations cannot be guaranteed in software RAID on a standard motherboard.

Some motherboards have integrated RAID controllers. These are often simple RAID controllers that provide RAID 1 mirroring only. This is sufficient for many applications.

Previously RAID was an expensive, luxurious option that only the most high-end applications could afford. Now it is inexpensive and commonplace.

Even Mirrored Disks Need Backups

A large e-commerce site used RAID 1 to duplicate the disks in its primary database server. Database corruption problems started to appear during peak usage times. Neither the database vendor nor the OS vendor was willing to accept responsibility. The SAs ultimately needed to get a memory dump from the system as the corruption was happening so that they could track down who was truly to blame.

Unknown to the SAs, the OS was using a signed integer rather than an unsigned one for a memory pointer. When the memory dump started, it reached the point at which the memory pointer became negative and started overwriting other partitions on the system disk. The RAID system faithfully copied the corruption onto the mirror, making it useless.

This OS software error caused a very long, expensive, and well-publicized outage that cost the company millions in lost transactions and dramatically lowered the price of its stock. The lesson learned here is that mirroring is quite useful, but never underestimate the utility of a good backup for getting back to a known good state.

15.2.5 Power Supplies

The second-most failure-prone component in a server is the power supply. It is very common to have $N + 1$ redundant power supplies so that if one fails, the system can keep running.

Each power supply provides a certain amount of power, measured in watts. Generally a second power supply is enough to provide $N + 1$ redundancy, but sometimes a third or fourth is required for large servers or network equipment with many fiber interfaces. The vendor can advise on how many power supplies a particular configuration requires both at a minimum and to provide $N + 1$ redundancy.

Each power supply should also have a separate power cord. Operationally speaking, the most common power problem is a power cord being accidentally pulled out of its socket. Yet, some vendors inexplicably provide dual power supplies with a single power cord. Such vendors demonstrate ignorance of this basic operational issue.

Each power supply should draw power from a different source: a separate circuit or uninterruptible power supply (UPS). Generally each power distribution unit (PDU) in a datacenter is its own circuit, so plugging each power cord into a different PDU assures two power sources.

Another reason for separate power cords is that they permit the following trick: Sometimes a device must be moved to a different power strip, UPS, or circuit. In this situation, separate power cords allow the device to move to the new power source one cord at a time, leaving the system up during the entire transition.

Separate Power Cords Save the Day

Tom once had a scheduled power outage for a UPS that powered an entire machine room. However, one router absolutely could not lose power, because it was critical to systems outside the machine room. This dependency was discovered only moments before the scheduled power outage.

Luckily, there was a wall socket near the router that would be unaffected by the outage. It had been installed for lights and other devices that did not require UPS support. While it wasn't UPS protected or an independent, clean circuit, some power was better than no power.

Shortly before the planned datacenter outage, Tom moved one of the router's power cords to the wall socket. During the outage the router ran on a single power supply. The day was saved.

15.3 Things to Leave Out

What you don't want on a server is anything you won't use. This includes fancy video cards, keyboards, monitors, and mice. Servers do not display graphics locally, so fancy video graphics cards are a waste. As discussed previously, monitors and keyboards take up precious datacenter room. A KVM permits better use of space.

A mobile phone or laptop is part computing device, part fashion statement. It really does matter what color it is. Servers, in contrast, should be used and not seen. They should be in datacenters where people generally won't see them. Servers don't need beautifully designed front panels and cases that look like they should be on display at the Museum of Modern Art. Cases should be designed to provide proper airflow at the lowest price possible. Beautiful, fancy cases are expensive to design and manufacture, yet the only value they provide is to impress you during the sales presentation. In a perfect world, vendors would offer a discount to customers who forego any purely decorative plastic bezel that attaches to the front of servers. Sadly, they don't.

If you look at pictures of Google datacenters, you'll see rack after rack of servers that are cost reduced to an extreme. No pretty front bezel. In fact, the original Google servers had no case at all. They consisted of bare motherboards that sat on metal shelves, separated only by a layer of insulation. You can see one such rack at the Computer History Museum in Mountain View, California. A little money saved on each server multiplies into big savings when you have many servers.

15.4 Summary

When you are purchasing a server, there are many options and many choices to be made. We begin by considering the application requirements and base our decisions on them.

Vendors offer different product lines, some focusing on lowest initial purchase price, lowest total cost of ownership (TCO), or highest performance. The lowest TCO is achieved by including facilities that reduce the cost of IT management, mostly by supporting remote management and features that enable the fleet to be managed as a whole instead of as individual hosts.

There are many types and models of CPUs. The Intel x86-64 is the most common today. Most servers have the option of a few fast cores or many slower cores; each is appropriate for different applications. Applications that take advantage of many cores include threaded applications, virtualization, and programs that are designed to distribute work over many cores. Applications that

work better with fewer fast cores include single-threaded legacy applications and certain mathematical computation systems.

Servers need a sufficient amount of RAM to run their applications. Modern CPUs improve RAM speed through caches and NUMA. Caches may be on the core (L1), between cores (L2), or on the motherboard (L3), although such distinctions are changing over time. NUMA places certain banks of RAM near certain CPUs; the operating system then attempts to optimize performance by placing programs running on a particular CPU in the appropriate RAM banks.

Servers usually need reliable, high-speed networking. The server needs network bandwidth proportional to the number of clients that access it. A server may use faster NICs or bond many slower NICs together so that they act as one larger channel. Multiple NICs can be used to improve resilience by connecting each to a different switch, thereby creating a system that can survive failed switches.

Different servers have different storage requirements. Disks fail, so servers should use RAID to ensure that they can survive a failed disk. The exception is servers that store caches of data that can be found elsewhere. Hardware RAID means the RAID processing is done on a separate controller; it is generally faster and has more features. With software RAID, the RAID processing is part of the operating system; it is generally slower but has the benefit of being included in the cost of the operating system.

Exercises

1. How do vendors typically differentiate their product lines?
2. Which features are different between the product lines that focus on best initial cost versus best total cost of ownership?
3. Identify three features that improve a server's total cost of ownership. How would we know at purchase time if the extra cost is worth it? How would we measure afterward if the goal was achieved?
4. Within a CPU family there are often models with fewer cores or more cores. What are the benefits to each?
5. What are L1/L2/L3 caches? What are their benefits?
6. What is NUMA? Why does it require changes to the operating system?
7. What are the pros and cons of software RAID versus hardware RAID?
8. Is there no software in hardware RAID?
9. Does RAID eliminate the need for backups?

10. Obtain a server hardware vendor's list of product lines. Classify each as having a focus on lowest initial cost, total cost of ownership, or performance.
11. Which servers are used in your environment? How many different vendors are used? Do you consider this to be a lot of vendors? What would be the benefits and problems with increasing the number of vendors? Decreasing the number?
12. [Section 15.2.2](#) mentions that programs may be more cache-efficient if they process data by reading information a single row at a time, rather than by a column. Explain how this change would affect caching.

Part IV: Services

Chapter 16. Service Requirements

Services are the applications that customers need and use. For the purpose of this book, the term **service** means all the parts that make an application work: the software, the hardware, and the operations that bring it all together. A service may be a single piece of software on a single machine, or many different software components across many machines that work in cooperation to provide a service, or something in between. A service may have direct customers, such as a web-based application, or it may be invisible to most people, such as a database that is used by other services.

Fundamentally, system administration is about providing services. Computers and software are useful only when they are actively providing a service, not sitting in a box. Services do not run themselves; it is a human process to create a service as well as to maintain it. A good service meets customer requirements, is reliable, and is maintainable.

A service begins life as a set of requirements. Customers and stakeholders have needs, and the service is created to fulfill them. These requirements are used to create the technical design. The service is deployed and launched. Launching a service is more complex than one would expect.

It is tempting to think of the launch of a service as the end of this process, but it is really just the beginning. Services usually run much longer than it takes to create them.

A running service requires maintenance and upkeep. As demand grows it is scaled to handle the larger workload. Requirements change over time, and new software releases must be deployed to fix bugs and add features. Because systems fail, disaster planning is part of any service. This includes basics such as data backups and more advanced practices such as service relocation.

Services need to be supported. They must be monitored, so that problems are detected and fixed. There needs to be a team that handles alerts from the monitoring system as well as end-user reports of problems. For many services, the customers can request moves, adds, changes, and deletes. There needs to be a mechanism for them to do so, and a team that is responsible for handling those requests.

In general, this book treats this subject from the perspective of an organization that buys third-party software. Volume 2 of this book series focuses on services where software is developed in-house. These approaches require very different

strategies and techniques.

We will discuss services in four parts. This chapter is an overview of services and service requirements. The next chapter covers engineering the solution once the product is selected. The subsequent chapters in this sequence are a deep dive into the service launch process. The issues related to actually running and supporting the service are covered in the remaining chapters of this book.

16.1 Services Make the Environment

The richness and utility of the environment we provide to our users depends on the quantity and quality of services we provide.

Services are what distinguish a structured computing environment that is managed by SAs from an environment in which there are one or more stand-alone computers. Homes and very small offices typically have a few services; often they simply rely on their ISP for foundational services. Larger organizations have a rich environment of services: from the applications that run the company, to the foundational services and infrastructure on which they depend. A large organization's fundamental services are run like an in-house ISP.

Foundational services create the platform that other services rely on. Examples include DNS, DHCP, directory services, network access (WAN and LAN), and Internet gateways. While they are generally invisible to users, failures are highly visible because they affect many services.

Basic services are user-visible applications that most people have come to expect in an organization. Examples include printing, email, file storage, chat/IM, and VoIP/phone service. Because of their visibility and pervasiveness, people have developed expectations that these services will be reliable and always available.

Lastly, there are **primary applications** and the **back-office systems** that support them. Applications generally drive business functions such as payroll, inventory management, enterprise resource planning (ERP), supply chain management, and so on. Back-office services include the databases and other behind-the-scenes services that support applications. Together they form the company's critical path: The company cannot function without them.

Companies No Longer Sell Atoms, They Manage Bits

In the old days computers were not in the critical path to accomplish the business's objectives. Rather, the computer was viewed as an ancillary device. You could sell a car or perform surgery on a sick patient even if

the computer was down. The computer was required after the fact for billing, inventory, and other things after the primary task was complete.

Today matters are much different. Computer-based services are in the critical path. You can't sell a car today without performing a credit check, entering the order into a supply-chain system, and so on. Surgery requires computer-based scheduling of people, facilities, and equipment; much of the surgical equipment itself is controlled by a computer. The supply chain that brings food to New York City holds about a three-day inventory; if the services that make up the supply chain failed, Manhattan would have to be evacuated lest three million people starve.

All aspects of modern society require that our services are available and performant. Improving a process, whether it is to make it more efficient, better, or more profitable, requires the involvement of the IT department. IT has a duty to be an agent of change and an ethical responsibility to participate in the constant process of change and improvement.

No matter which business a company thinks it is in, it is really in the software and operations business. If its management does not realize this foundational fact, the company is doomed.

16.2 Starting with a Kick-Off Meeting

Creating and launching a new service is difficult. It cannot be done alone or in a vacuum.

Most of the requirements gathering and project management are done via email, IM, teleconference, and video chat. Nevertheless, it is critical to start out with a face-to-face meeting—the kick-off meeting.

A kick-off meeting should have all the stakeholders present. **Stakeholders** are all the key people affected by or involved in the project. Use this meeting to get agreement on the goal of the new service (i.e., which problem is being solved), a timeline for completion, and an approximate budget. You won't be able to resolve all these issues, but you can bring them out into the open. Delegate each unresolved issue to a participant who will be accountable for its resolution.

Having all the stakeholders see one another's faces and learn one another's names has a bonding effect that improves collaboration in the future. Taking the time to do this is an investment in the success of the project. Everyone should introduce themselves and explain their role in the project.

Although painfully low-tech, in-person meetings work better. People ignore

email. Phone calls don't convey people's visual cues. A lot of people on a conference call press mute and don't participate. A video conference is sufficient if a physical meeting is impossible or too costly.

Once the kick-off meeting has put everyone on the same page, status meetings can be held by phone and updates provided via email.

16.3 Gathering Written Requirements

The next step is gathering requirements. Requirements are a list of what the service will be able to do. Requirements should list desired functionality, features, and capabilities. Focus on the end goal: what the system will enable people to achieve. The list of requirements guides all the other steps: design, engineering, implementation, testing, and so on. It avoids confusion and finger-pointing. It sets expectations with customers. It can even be a checklist that lets you know when you are done.

Requirements should not include engineering and implementation details. Requirements are not the place to list which language the software will be written in or which font will be used.

When defining the requirements, don't try to boil the ocean. Some requested features will be too difficult or expensive to implement. Some may not be appropriate for this service to provide, but may be better suited to a different service. Some may be required by too few people to be worth implementing, at least in the initial release.

Some requirements may have external dependencies on other groups, projects, or services. Make sure to flag each of these external dependencies, and have a plan for what to do if the external dependency is not delivered on time.

There are different kinds of requirements. **Customer requirements** describe the features and functionality of the service to be built. **Operational requirements** describe what is needed to make the system work in the organization's IT infrastructure and be maintainable. Management may have budget requirements, security may have access requirements, and so on.

Requirements are written down. They are not simply agreed to in a verbal discussion, tracked on a dry-erase board in your office, or kept in your head. Writing them down in a shared requirements document has many benefits:

- **Transparency:** Unlike your brain or the dry-erase board in your office, everyone can read the requirements document.
- **Fewer gaps:** Writing down requirements reduces errors caused by a request being misheard or forgotten. People can verify that their request

was recorded properly.

- **Fewer misunderstandings:** Two people often think they have verbal agreement when they do not. Seeing the decision in writing verifies that everyone is agreeing to the same thing.
- **Buy-in and approval:** Written requirements provide the ability to get buy-in and management approval in an accountable way. People know what they are agreeing to. People can't claim they didn't sign off on the requirements if they've literally signed their name on a printout. They may claim they didn't understand the document or that circumstances and budgets have changed, but at least you have baseline agreement.
- **Fixed scope:** A formal requirements document provides a mechanism for handling additional feature requests that arrive after the agreement has been reached, preventing feature-creep. New requirements need to be formally agreed to, documented as a scope change, and subjected to buy-in and management approval before they can be added to the scope.
- **Accountability:** Accountability is a two-way street. Customers can point to the document when a requirement you've agreed to turns out to be missing or incomplete. Documentation of requirements also helps delineate features versus bugs. A bug is a requirement that is incorrectly implemented. A feature request is a new requirement that wasn't previously agreed to. While a bug should be fixed, a feature request requires approval, resource allocation, and possibly budget approval.

A requirements document does not need to be hundreds of pages long. A bullet list may be sufficient for small projects. The essence of business writing is brevity. A short, concise document is easier to read and understand.

Define terminology early in the document. Getting agreement to the ontology that will be used is very important. **Ontology** is the system of terms and definitions that define the system and its parts. Often during a heated debate one realizes that everyone is using the same words but meaning different things. Pausing to get agreement on terminology helps everyone see eye-to-eye. At that point we often realize we were closer to agreement than we had previously realized.

Annotate the importance of each requirement. Marking each as “must have,” “desired,” or “optional/nice to have” communicates intent, gives the system designers flexibility, and helps implementors prioritize work when deadlines are looming and features must be cut.

Applied Ontology: Clarifying Definitions

A very inefficient and contentious requirements-gathering process was improved when someone pointed out that everyone had a different definition of a computer. Is a computer the virtual machine, the machine that the virtual machine runs on, or the entity in a IPAM database, which might be out-of-date and describe a machine that used to exist or will exist in the future?

In this company, it was decided that a computer is a physical machine. A node is a physical machine used to host virtual machines. A virtual machine (VM) is a virtual machine that runs on a machine. A host is the IPAM database entry that can describe either a physical machine, a virtual machine, or a node.

Once this ontology was defined, the confusing discussions were greatly reduced. These may not be the definitions used at other sites or in the industry, but they were what was needed for this particular company.

Ontology resolves confusion regarding many things, including the names of various customer roles, types of transactions, and the phases of a process. Ontology is the bedrock of communication.

16.4 Customer Requirements

Services are built for the customers. If a service does not meet the customers' needs, building the service was a wasted effort. If we don't gather customer requirements, we don't know our customers' needs. We may *think* we know their needs, but the ESP skills of most system administrators are surprisingly overrated.

Gathering customer requirements is an art and a skill. It requires patience and listening, and many iterations going back and forth until the list is complete.

16.4.1 Describing Features

The requirements should focus on the list of features, stated from the perspective of what the customer should be able to accomplish using business terms, not technical terms. Record the "what," not the "how."

It is better to record a requirement such as "the user should be able to send email" than "there should be a button on the left that the user clicks to send email." The latter assumes a lot about the interface. Why should the designers be required to include a button even though they have a better idea?

This also means not proscribing particular technology. For example, when an email system is being designed, users might request that it be compatible with the IMAP protocol (RFC3501). They should, instead, be stating their needs at a higher level of abstraction. Ask them why they need such a specific thing. For example, maybe they need to be able to read their email from their smartphone and they know that their phone supports IMAP. It would be better to record the list of smartphones that must be supported, since there may be better ways to support that feature than IMAP. Requiring IMAP support, therefore, may over-specify the requirement.

Conversely, specifying IMAP support may under-specify the feature. Imagine the user's surprise when the IMAP support is available but his or her particular smartphone is unable to access email. The feature is complete as specified—IMAP is supported—but the user is unable to read email. Requesting that this problem be fixed would be a feature request, not a bug. It would be rejected much to the surprise and dismay of the customer. Technically speaking, the product works as requested.

It is this kind of situation that makes perfect sense to a technical person but is frustrating and unreasonable to a typical user. This is one reason why users view IT departments as difficult to deal with. They're being told they got what they asked for, but in their mind that's not true: They can't read email from their phone. Phrasing requirements at the right level of abstraction is one way that we can prevent this problem.

16.4.2 Questions to Ask

Ask how, why, where, and when, as if you are a journalist conducting an interview.

How do customers intend to use the new service? Why do they need each feature? Where, when, and how will the system be accessed? How critical is the service to them, and which levels of availability and support do they need for the service?

Determine how large the customer base for this service will be and what sort of performance they will expect from it. For example, when building an email system, try to estimate how many emails, both inbound and outbound, will be flowing through the system on peak days, and how much disk space each user would need. How fast should users be able to open, send, and delete messages? And so on.

Another way to record requirements is through use cases that tell a story. As

long as the story can be completed by the final design, the use case is sufficiently covered. In Agile methodology, the format is “As a <type of user>, I want <some goal> so that <some reason>.” For example, one might specify: “As a payroll clerk, I want to be able to enter future payroll adjustments so that I don’t need to do them all on the day they take effect.” Another example: “As a user, I want to indicate which folders should not be backed up so that my backup drive isn’t filled up with things I don’t need saved.”

16.4.3 Service Level Agreements

The requirements document should include the service level agreement (SLA), or at least some general specifications that can be used to build an SLA.

An SLA enumerates the services that will be provided and the level of support they receive. It typically categorizes problems by severity and commits to response times for each category, perhaps based on the time of day and day of the week if the site does not provide 24/7 support. The SLA usually defines an escalation process that increases the severity of a problem if it has not been resolved after a specified time and calls for managers to get involved if problems are getting out of hand. In a relationship in which the customer is paying for a certain service, the SLA usually specifies penalties if the service provider fails to meet a given standard of service. The SLA is always discussed in detail and agreed on by both parties.

The SLA creation process is a forum for the SAs to understand the customers’ expectations and to set them appropriately, so that the customers understand what is and isn’t possible and why. It is also a tool to plan which resources will be required for the project. The SLA should document the customers’ needs and set realistic goals for the SA team in terms of features, availability, performance, and support. It should document future needs and capacity so that all parties understand the growth plans, and their approximate costs. The SLA is a document that the SA team can refer to during the engineering process to make sure that they meet customers’ and their own expectations and to help keep them on track.

16.4.4 Handling Difficult Requests

Requirements gathering is a collaborative process. The ultimate goal is to find the middle ground between what the customer ideally wants, what is technically possible, what is financially affordable, what can be accomplished by the date that the solution needs to be available, and what the SA team can provide. A feature that will take years to develop is not reasonable for a system that must be

deployed next month. A feature that will cost \$1 million is not reasonable for a project with a budget that's in the thousands of dollars. A small company with only one or two SAs will not get 24/7 support, no matter how much the company wants that.

Your job is to educate as much as it is to record features.

Don't become upset when a customer asks for something technically unreasonable; if the customer knew technology as well as you do, the customer would be an SA. Try to understand the end goal and see how that can be achieved instead. Customers might ask for something that is outside the development budget or sounds technically impossible. For example, they may ask for offline access on a system that is highly interactive, or they may want something with a 24-inch screen to fit inside a pants pocket. Rather than say "no," focus on the need, not the technology. Ask which benefit or need they are trying to fulfill. It may simply be that sales representatives need to access price information when they are in front of their customers, which is usually in places without network access. That's a reasonable request when recorded that way. The end result may be a single-page price list that is downloaded to the person's phone, or a laminated price sheet that the sales rep puts in a wallet.

A common request from non-engineers is that a service be always available or have 100 percent uptime. Non-engineers do not realize how unrealistic that is. Some SAs get extremely upset at such requests. This is a moment to step back and take a few deep breaths. You're going to spend the next few minutes or so educating your customers about system reliability.

Remember that to a layperson, "always" doesn't mean 24/7; rather, it means that the system works when the user needs it. This may be from 8 AM to 6 PM. Take a moment to explain to the person that there is "always" and then there is "always."

Explain that the difference between a system that is up 99 percent of the time and 99.9 percent may be in the thousands of dollars. Improving it to 99.99 percent uptime often costs tens of thousands of dollars, while 99.999 percent uptime often costs in the millions of dollars, and 99.9999 percent uptime is more than the company can afford. Write this on a whiteboard so the person can see the progression. Each step is not 10 times more expensive, but rather 10 times 10 times more expensive.

For comparison, explain that WiFi in a coffee shop has about 98 percent uptime, so striving for better than that doesn't make sense if most users are going to be working remotely.

At this point you have a common basis for understanding and can have a more reasonable discussion about uptime requirements. Break the week into different time slots: office hours, weekends, late nights, and so on. Define different uptime requirements for each of these segments.

You'll find taking a few minutes to educate a customer is well worth the effort.

16.5 Scope, Schedule, and Resources

A project plan generally has three elements: scope (the features), schedule (the deadline), and resources (the budget and the staffing). If a project is at risk of not making its deadline, one or more of these will have to change. Features will need to be dropped, the deadline will need to be extended, or the budget (in the form of money or staff allocation) will need to be increased.

For example, if the project needs to be complete in time for the Christmas shopping season, the schedule is inflexible—so the features and/or budget needs to be flexible. If the budget is inflexible, it must be possible to remove features or extend the schedule.

If the project is to send humans to the moon and safely return them, the features are inflexible; a one-way trip is not an option. In this case, either the budget or the schedule must be flexible.

Get buy-in from management as to which of these elements are flexible and which are inflexible at the start of the project. The buy-in process helps communicate priorities to all stakeholders, prevents a lot of confusion all around, and helps focus people on the right priorities. It also has the benefit of discovering early on if management is unwilling to be flexible in any of the three areas. In that case, you have time to send out resumes and find a new job because your management is crazy. Again, that's something you'd rather know early on in the project.

If the project does look like it will be late, and you have agreement that the schedule is more flexible than the features, then you've already set expectations with management. Giving bad news is easier when such expectations have already been set. In an ideal situation, the conversation will not be about what to do, but rather will simply request permission to act on a previously made decision.

16.6 Operational Requirements

Operational requirements are the things that make a system maintainable. These

are the features required if we are to have a system that is reliable, scalable, and secure. These features are often invisible to the customers—yet reliability is the most visible aspect of a service.

16.6.1 System Observability

The most fundamental, and therefore most important, key to a manageable system is that you can see what it is doing. That is, it must be observable or we cannot manage it, fix it, or scale it. The system must provide enough visibility and introspection to debug it when there are problems, optimize it when it is slow, and detect when resources are low before they become a problem. We must have situational awareness to reason about it, predict its future behavior, and do capacity planning. We laugh hysterically when salespeople tell us that their system runs itself, or that their company’s goal is to build a system so that it doesn’t need maintenance. Yes, that should be their goal, but until it is achieved vendors must build observable systems.

The most basic kind of visibility is logging. All parts of the system should log events, transitions, and operations. Logging should be adjustable to provide more detail when debugging. The level of detail may include logging every API call or function call.

The next most important visibility is monitoring. It should be possible to monitor not just whether the system is up or down, but also internal resources and timing: How much bandwidth is being used? How much free disk space is available? How long are requests taking?

If the only thing that can be monitored is whether the system is up or down, then there is no way to prevent outages; you can only respond to them. In other words, you will be warned when it is too late to prevent an outage.

The new system should integrate into existing monitoring systems, dashboards, trouble-ticket systems, and other appropriate visibility tools.

Monitoring is discussed in greater detail in [Chapter 38](#), “[Service Monitoring](#).”

16.6.2 Remote and Central Management

Services should be remotely managed. If you have to be on the console of a machine to do rudimentary maintenance and configuration, each task will be more time-consuming. This is also an indication that automating management tasks may not be possible.

Servers are generally located in machine rooms where power and cooling can be managed more effectively. Remote management permits servers to be located

anywhere. It is less expensive to rent colocation space elsewhere than to build a computer room in your office. It is often also more reliable.

Services that are replicated in many places must be manageable in a centralized fashion. Remote management opens up the possibility of deploying instances of the service around the world. Configuration changes and upgrades should not require manually updating each instance.

Remote management systems should work well over long-distance (high-latency) connections. For example, some remote management utilities fail to work if the management client is in California and the service itself is in Europe. In this situation each network packet takes more than 100 ms to reach its destination. Protocols may time out. User interfaces may work but can be unusably slow. This is not a limit of the technology, but rather bad software engineering. Plenty of APIs and user interfaces work well over high-latency links—and so should management tools.

Management Over High-Latency Links

When Tom complained to Dell that iDRAC's web-based user interface ran unacceptably slowly from as little as 12 miles away, he was told that iDRAC wasn't designed to work at long distances. We'll assume the representative misspoke, since it would be hilarious if a remote management system wasn't designed to work well remotely.

16.6.3 Scaling Up or Out

If a service is successful, more people will want to use it. Over time it will need to scale to handle more users, more transactions, or more capacity. Services generally scale in one of two ways: up or out.

Scale-up means getting a bigger system. To process more requests, more users, or more disk space, the main system is replaced with one that is bigger and faster. For example, to scale up a web server so that it can process more requests per second, you might replace the computer with one that has a faster CPU and I/O system.

Scale-out means the system is expanded by adding more replicas. For example, a web-based service is often made up of web servers behind a load balancer. Scaling out means adding more redundant web servers (replicas). Each web server scales out the system to handle more requests per second.

16.6.4 Software Upgrades

A system must have a reasonable way to upgrade the software and firmware. Ideally, it should be possible to test a new software release outside of production and upgrade the service without interrupting it. Typically, commercial software requires some kind of service interruption when it is upgraded. In that case the interruption should be minimized such that it fits in a scheduled maintenance window.

Software is never done. There will always be the need for new software releases, even if every feature is complete and every bug is fixed. New security holes are discovered constantly, and may not even be in the vendor's code but in a library or framework it used. For example, in April 2014 a major vulnerability in the OpenSSL cryptography library was announced. The bug, termed Heartbleed, existed in thousands of software packages. Any company that thought its software was "done" got a rude awakening.

More importantly, software is malleable and vendors should be providing new features and improvements over time. There's always room for improvement.

Software Is Not a Hammer

A hammer is manufactured and is ready to be sold to the customer. At that point the product is complete. Software is never complete. Anyone who disagrees should leave the software business and go back to making hammers. Be wary of vendors that provide upgrades rarely, sparingly, or only begrudgingly. If a security hole is discovered, the vendor should be able to provide a software patch within hours or days, not leaving you to wait for the annual software release. If software releases aren't frequent, periodic, and scheduled, then this is a sign that the vendor's management thinks their company is making hammers. Such companies should not receive our support because they fail to support us.

Upgrade processes should exist and be automatable. The upgrade process should integrate into the organization's software patching automation. It should not involve walking to each desktop machine for a manual update. It should be possible to upgrade some clients, but not all, so that we can mitigate risk by rolling out the upgrade slowly. If all clients have to be upgraded simultaneously, then there is no way to test the upgrade. As the number of clients grows, the concept of upgrading clients at the same time becomes more and more inconceivable.

The more people depend on a service, the more visible a failed upgrade will

be. It should be an operational requirement that the service is designed in a way such that it is possible to deploy the service to a test environment for the purpose of both testing the upgrade process and testing the new release itself.

A **production assurance test (PAT)** or **user acceptance test (UAT)** environment can be a good option for preproduction testing. PAT and UAT environments are run at production standard, but upgraded ahead of the main production environment. Some sites select volunteers who use the PAT or UAT environment, rather than the production environment, at all times, to ensure that when an upgrade is rolled out in that environment, it is really used and production tested.

16.6.5 Environment Fit

The better a service fits into the existing IT environment, the easier it is to adopt it in your environment. Integration issues are reduced and less training or skill development is required. For example, if you use ActiveDirectory, then using a product that interfaces with it will be easier than using a product that has its own directory service. If the service runs on an operating system with which the team is unfamiliar, using it will require not just additional training, but entirely new procedures that have to be developed for tasks such as data backups, account creation, and configuration.

For a small project, it is reasonable to make it an operational requirement that the service fit into the existing OS infrastructure, directory service, and so on. Larger projects require more flexibility. For example, a large SAP ERP deployment is a self-contained environment unto itself. Therefore it is more acceptable that it may introduce a new operating system to your environment. Most likely it will have its own system administration team.

Here are some environmental factors to consider:

- Operating system
- Backup/restore facilities
- Monitoring system
- Network equipment vendor and routing protocols
- Security auditing systems
- Trouble-ticket systems
- Dashboards and consoles
- Network directory services
- DNS and other name services

- Printing systems

16.6.6 Support Model

A key component in the success of a new service is support processes. Proper support is a requirement that everyone takes for granted, so no one will think to mention it explicitly. This aspect is often forgotten or neglected by the engineers, who are intent on developing the technical solution.

Once the service goes into production, people will start using it. When they have problems with it, they will raise tickets or call the helpdesk. If the helpdesk does not know about the new service, the users of the service will receive very poor support, and will have a bad impression of the service, no matter how well it has been implemented. Define the support model in advance:

- What should the helpdesk personnel do when they receive a ticket related to this service?
- How can they identify that a problem is related to this service?
- Which diagnostic steps should they take?
- Which documentation do they need?
- Which access and authorizations will they need?
- To whom do they escalate the problem, and when?
- Which information should be included in the escalation?
- Will the service have a dedicated support team, or will support for this service be integrated into another team's portfolio?
- Do any of these support teams need additional resources to take on the support of this service?
- Which training or certifications should the support teams receive before the service goes live?

The project plan will need to include time and budget resources to implement the answers to these questions. Documentation may need to be written, new staff hired, training completed, helpdesk operations updated, monitoring systems expanded, and so on.

16.6.7 Service Requests

Most services will include standard service requests that people can raise. Depending on the service, these may be access or authorization requests, data changes, configuration changes, resource requests, and so on. A checklist for

building requirements might look like this:

- What should people be able to request?
- How do people raise these requests?
- Which changes need to be made to the service request system to support these new service requests?
- Which approvals are required for each of these service requests?
- Who will process the requests?
- Which access and authorizations will that team need?
- Which runbooks does that team need to correctly service the requests?
- To what extent can, or should, these requests be self-service or fully automated?

The answers to these questions will provide some new requirements. For example, depending on whether the requests should be self-service or automated, there are API requirements that need to be taken into consideration. The requirements and dependencies on other systems, such as the service request system, also need to be documented, agreed to, signed-off on, budgeted, and planned.

16.6.8 Disaster Recovery

Failures happen. Hardware fails, buildings lose power, Internet access gets cut, and so on. Acting like everything will always function perfectly is irrational. Instead, we need to prepare for failures so that we know how to recover from them.

The most basic disaster recovery requirement is that there must be a way to perform backups and restores. This includes total system restores, as well as the ability to restore an individual file or customer data point.

The most common data restore requirement is for something that was deleted by accident. As a result, the operational requirement of restoring individual data points has been implemented in most systems, often with no SA intervention required. For example, the original version of Microsoft Exchange could only back up and restore the entire system. To restore an individual lost email, one needed an otherwise unused machine with enough storage capacity to restore the entire message database. Then the individual email message would be cherry-picked from it. More recent versions of Exchange have the ability to restore an individual's email messages or a specific message, and most of the time people can help themselves by pulling messages from the recycle bin or "sent" folder

before they are cleaned out.

It can be useful to be able to separately back up the system configuration and the user data. Configuration files that are human-readable are preferable to large binary blobs. Being human-readable makes these files easy to store in source code repositories, makes it easy to produce historical `diff` listings, and so on.

As discussed in [Chapter 14](#), “[Server Hardware Features](#),” enterprise applications tend to assume data integrity and resiliency issues are handled at the hardware layer. In other words, working around hardware faults is your problem. Therefore, operational requirements should be specified to meet your needs. For example, you may require that the service run from multiple datacenters, assuming that at least one will be up at any time. If the service will be single-homed, the service can be only as reliable as the datacenter it is in. That may be sufficient for some services.

16.7 Open Architecture

Wherever possible, a new service should be built around an architecture that uses **open standards** and open protocols. This means protocols, file formats, and APIs that are publicly documented so that others can write to those standards and make interoperable products without having to worry about royalties or patent restrictions. Any service with an open architecture can be more easily integrated with other services that follow the same standards.

By contrast, a **closed service** uses standards, protocols, APIs, and file formats that are owned by one company, are controlled by that one company, and do not interoperate with other products. Other products are prevented from using the standard because the standard is not publicly documented, because it requires licensing, or because the vendor forbids it. Vendors use proprietary protocols when they are covering new territory or are attempting to maintain market share by preventing the creation of a level playing field. In the former scenario, the vendor should then work with the appropriate standards bodies to define and document open standards, and many vendors do so.

The Protocol Versus the Product

SAs should understand the difference between the protocol and the product. For example, there is a difference between the IMAP protocol and the software that implement it. IMAP is not a product but rather a document, written in plain English, that explains how bits are to be transmitted over the wire. This is different from a product that uses IMAP

to permit clients to access email. The confusion manifests itself frequently by SAs who conflate the protocol and the software that implements the protocol.

Part of the confusion comes from the overuse of the word “standard.” Companies often refer to a product as being “the standard,” meaning that it has market dominance and is the product to which other products are compared. This usage does not mean there is an open standard which permits other companies to make products that interoperate with the so-called “standard” product.

The source of this confusion is understandable. Before the late 1990s, when “the Internet” became a household term, many people had experience only with protocols that were tied to a particular product and didn’t need to communicate with other companies, because companies were not interconnected as freely as they are now. No Internet? No need to interoperate. This situation gave rise to the notion that a protocol is something that a particular software package implements and does not stand on its own as an independent concept.

Although the Internet has made more people aware of the difference between protocols and products, many vendors still take advantage of customers who lack awareness of open protocols. Such vendors fear the potential for competition and would rather eliminate competition by locking people into systems that make migration to other vendors difficult. These vendors make a concerted effort to blur the difference between the protocol and the product.

Beware of vendors that *embrace and extend* a standard in an attempt to prevent interoperability with competitors. Such vendors do this so they can claim to support a standard without giving their customers the benefits of interoperability. That’s not very *customer oriented*. A famous case of this occurred when Microsoft adopted the Kerberos authentication system as part of ActiveDirectory. This was a good technical decision, as Kerberos is exceptionally secure and robust. However, Microsoft made a gratuitous change to the protocol that Windows clients depended on. Non-Windows clients could talk to any server, but Windows clients could interoperate with only those servers that included the change. Microsoft would not reveal the technical details of the change, so if you had a single Windows client you had to buy all your servers from Microsoft. Microsoft’s decision successfully forced sites to uproot

their security infrastructures and replace them with Microsoft products. Thus it enabled Microsoft to claim support for an open protocol but robbed customers of the freedom of choice.

The business case for using open protocols is simple: It creates competition that results in better products for you. By unbundling the client from the server, you can mix any client product with any server product. We have more choices. Vendors now can compete separately at the client level and the server level. We can pick the best of class product in each category.

With closed systems, our hands are tied: We must purchase the client and the server from the same vendor. We have been in many situations where one vendor has a demonstrably better client, but a server that is slow, buggy, and difficult to manage. Meanwhile, another vendor has a bare-bones client but a server that is highly scalable and easy to manage. Since the needs of the many outweigh the needs of the few, or the system administrator, this situation tends to end up with us selecting the first option. If the systems had been based on open protocols, we could have picked the best of both worlds. Instead, the system administrators are left to keep the server limping along.

A better way is to select protocols based on open standards, permitting users and operations to select their own software.

For comparison, the next anecdote illustrates what can happen when customers select a proprietary email system that does not use open protocols but fits their client-side needs.

Hazards of Proprietary Email Software

A pharmaceutical company based in New Jersey selected a particular proprietary email package for its PC user base after a long evaluation. The selection was based on user interface and features, with little concern for ease of server management, reliability, or scalability.

The system turned out to be very unreliable when scaled to a large user base. The system stored all messages from all users in a single large file that everyone had to have write access to, which was a security nightmare. Frequent data-corruption problems resulted in having to send the email database to the vendor across the Internet for demangling. This meant that potentially sensitive information was being exposed to people outside the company and that the people within the company could have no expectation of privacy for email. It also caused long outages of the email system, because it was unusable while the database was being repaired.

Because of the lack of competition, the vendor considered server management to be a low priority and ignored the requests for server-related fixes and improvements.

Because the system was not based on open protocols, the system support staff could not seek out a replacement server without also replacing thousands of clients at the same time. That was not an option at the time due to retraining and other costs. As a result, the company chose to struggle with the service for many years, since the pain was mostly invisible to the users.

This situation has improved over time, since most applications have migrated to web-based applications. In the old days we had to install a single-purpose application for email, another for calendaring, and more for bug tracking, finance, expense reports, and so on. Now most of those applications are web based, which means we must simply have a compatible web browser. However, even that convenience can be stymied by proprietary plug-ins and extensions such as ActiveX, Silverlight, and Flash, which support only certain browsers on certain operating systems. Given these limitations, it is important to certify web-based applications on all browser/OS combinations that your organization requires.

Open protocols and file formats typically change only in upwardly compatible ways and are widely supported, giving you the maximum product choices and maximum chance of obtaining reliable, interoperable products.

Another benefit from using open systems is that they don't require gateways to the rest of the world. Gateways are the "glue" that translate between different systems. Although a gateway is a lifesaver when you need to bridge the gap between two incompatible systems, not needing a gateway is even better. Gateways are an additional service that adds to the administrative workload, requiring capacity planning, engineering, monitoring, and, well, everything else in this book. Reducing the number of services is a good thing.

Protocol Gateways and Reliability Reduction

In college, Tom's email system was a proprietary system that was not based on Internet standard protocols, such as SMTP. Instead, the system was sold with a separate service that used a gateway to send email to and from the Internet. The gateway used its proprietary protocol to communicate with the mail server and SMTP to communicate with the

rest of the world, and vice versa. The gateway added to the administrative burden of running the system: It was yet another thing to manage, debug, do capacity planning for, and so on.

This gateway was slow, unreliable, and expensive. It seemed that the vendor had engineered the gateway with the assumption that only a tiny fraction of the email traffic would go through the gateway. It had very low throughput. The mail system had many outages, nearly all of which were problems with the gateway.

The vendor had little incentive to improve the gateway, because it let customers communicate with systems that were considered to be the competition.

None of these problems would have arisen if the system had been based on open protocols rather than requiring a gateway.

History repeated itself nearly a decade later when Microsoft's Exchange mail server was introduced. It used nonstandard protocols and offered gateways for communicating with other sites on the Internet. These gateways added to the list of services that SAs needed to engineer, configure, plan capacity for, scale, and so on. Many of the highly publicized Exchange problems were related to the gateway.

These examples might seem outdated, since today no one now would sell an email system that is ignorant of the Internet. However, it is important to remember these lessons the next time a salesperson tries to sell you a calendar management system, directory service, or other product that ignores Internet and other industry standards but promises excellent gateways at an extra cost or for free. Using standard protocols means using open standards, such as Internet Engineering Task Force (IETF) and Institute of Electrical and Electronic Engineers (IEEE) standards, not vendor-proprietary standards. Vendor-proprietary protocols lead to future headaches. A vendor that offers gateways is probably not using open standards. If you are unsure, directly ask which open standards the gateways interoperate with.

16.8 Summary

A fundamental role of system administration is to provide services to users. In this book we use the term “service” to mean all the parts that make the application work: the software, the hardware, and the operations that bring it all together. In some organizations the IT department provides a rich set of services;

in others, very few. Some SA teams are responsible for all services; others are responsible for one of many. Some services form the foundation of all other services, others are basic infrastructure that users require to work and communicate, and still others are applications that drive the company.

When creating a new service, begin by getting agreement from all involved about the end goal, timeline, and approximate budget. Gather requirements. Customer requirements are the attributes and features that provide value to the company. Operational requirements are the features that make it possible to run the service. Requirements should be written down to provide transparency and to assure accountability in agreements and approvals. When gathering customer requirements, specify end results, rather than the technologies or details used to get there. Ask open-ended questions. Define an SLA. Be prepared for impossible-sounding requests that will require you to investigate the need behind the request.

Specifying operational requirements creates a foundation that enables SLA requirements to match what is required to achieve the SLA. A system must be observable, so that it can be monitored and debugged. Other requirements include remote management and the ability for the system to scale, to be upgraded, to survive failures, and to be restored after a disaster. Systems based on open architectures, standards, and protocols encourage vendor competition. Systems should fit into the existing technology environment, including monitoring systems, directory services, and so on.

Exercises

1. What is a service? What are the qualities of a good service?
2. Why are services important?
3. What are requirements?
4. Which types of requirements should be gathered when building a new service?
5. What can you do if a customer requests an impossible (or impossible-sounding) requirement?
6. Why do you need to write down requirements?
7. Pick a service that you use. If you were going to make a similar service, which requirements would it have?
8. What is an SLA?
9. Pick a service you are responsible for. What is the SLA? If it doesn't have

one, what do you imagine the unofficial SLA is?

10. What are the three elements of a project plan?
11. Identify the three elements of the project plan as specified in the following quote. What does their order, and presence or absence, indicate?
“I believe that this nation should commit itself to achieving the goal, before this decade is out, of landing a man on the moon and returning him safely to the earth.”—President John F. Kennedy, announcing his proposal for the Apollo moon mission
12. Why is the ability to upgrade software important?
13. What does environmental fit mean?
14. If a new service was being created in your organization, which existing infrastructure should it leverage, rather than reinventing from scratch?
15. Why do open architectures, standards, and protocols benefit the consumer?
16. List all the services that you can think of in your environment. Which hardware and software make up each one? List their dependencies.
17. Select a service that you are designing or you may design in the future. What will you need to do to make it meet the recommendations in this chapter? How will you roll out the service to customers?

Chapter 17. Service Planning and Engineering

After a product has been selected we need to engineer how it will be configured and operated in our environment. For example, engineering a Microsoft Exchange service involves determining which hardware will be used, how the software will be installed and configured, how users will access and authenticate to the service, and how various operational tasks such as backups, monitoring, archiving, and account management will be done.

This chapter focuses on third-party products. How do we design a service around them? One might think this is a matter of following the installation instructions, but many of the decisions are actually left to the individual system administrator.

Both the **service architecture** and the **local engineering plans** need to be defined and documented. The service architecture is the high-level description and the local engineering plan is the specifics for a particular machine or site.

For example, a service architecture for an email system might describe that there will be three regional clusters, each with the capacity to serve the number of users in that region. The plan would describe, generically, the clusters in terms of types of machines, storage, and networking technologies. These details would include which brands and products would be used.

The local engineering plan for each cluster is much more specific. It might include specific product ordering codes, including codes for each cable, connector, and mounting bracket. While the service architecture includes networking speeds and IP address range sizes, the local engineering plan lists the exact IP addresses to be used for each interface.

Typically the service architecture is developed, and then the local engineering plan is created based on that architecture. In reality, though, there is a great deal of interplay between the two. One step in validating the architecture is to write out the local engineering plans to see if anything was forgotten. One or more local engineering plans might be developed and built in a lab to test reality against the plans. What is learned from this process is used to update the architecture. This process is repeated until we have confidence that the architecture is sound and the local engineering plans are optimal.

At the same time the operational model or support model needs to be defined. The support model defines the operational level agreement (OLA) and the service level agreement (SLA) for the service, as well as who supports the

various components of the service.

The remainder of this chapter discusses some general engineering basics and philosophy and then ways to optimize such plans.

This chapter focuses on designing a generic service. [Part VIII](#), “[Service Recommendations](#),” provides specific recommendations for particular types of applications and services.

17.1 General Engineering Basics

Before this chapter goes into the more sophisticated aspects of service engineering, there are some basic issues that you should be aware of. Much of this includes the kind of engineering decisions that “go without saying,” yet are so crucial that we need to include them. Consider the following to be rules of thumb, often learned the hard way in real-world environments:

- **Follow vendor recommendations.** Determine how many machines are needed, and which type of CPU horsepower, disk space, RAM, network bandwidth, and so on is required. Most of these decisions have already been discussed in [Chapter 14](#), “[Server Hardware Features](#),” and [Chapter 15](#), “[Server Hardware Specifications](#).”
- **Mirror your boot disk.** Disks fail. Often the most difficult disk to replace is the boot disk; it requires reinstalling all the software. Mirror the boot disk (RAID 1) so that when (not if) a disk fails, the system will keep running and give you time to replace the failed disk. The only exception to this is when the machine can be rebuilt from scratch in an automated fashion and the machine stores no state, as discussed in [Chapter 3](#), “[Pets and Cattle](#).”
- **Run services on servers, not workstations.** It is bad practice to run a service on a workstation under your desk. There’s nothing more permanent than a temporary solution.
- **Run services from computer rooms, not offices.** Computer rooms also provide physical security, high-speed networks, reliable power, and cooling that is not available in the typical office.
- **Restrict console access.** The general public should not be able to log into the console of a server.
- **Leverage the existing infrastructure.** As much as possible, new services should be integrated into existing systems and processes. The existing monitoring, alerting, and service request systems should be used, as well as backup systems, and configuration and patch management systems.

17.2 Simplicity

When engineering a service, your foremost consideration should be simplicity. Strive to create the simplest solution that satisfies all the requirements. A simple service will be the easiest to maintain, easiest to expand, and easiest to integrate with other systems. Undue complexity leads to confusion, mistakes, and potential difficulty of use and may well make everything slower. Complex systems are also more expensive in setup cost and maintenance costs.

Computer scientist Brian Kernighan famously stated that debugging is, by definition, more difficult than creating a system. Therefore, when you create a system that is the pinnacle of your engineering abilities, you have, by definition, created a system that you cannot debug.

Case Study: Cornered by Complexity

A small Pennsylvania-based social network web site had an extremely complicated network designed by the company's lead technical person. It used complex routing protocols in unusual ways to achieve load balancing and rapid failover in case of failure. The system practically ran itself, except when it didn't. Only its designer understood how it worked and could debug it. This led to a number of problems when this person became overloaded and was pulled in many directions.

The site tried to hire a full-time networking person to offload the designer's responsibility for maintaining the network. Because the company was located in rural Pennsylvania, the candidates interviewed mostly had corporate network skills, whereas what the site really needed was a combination of ISP-style networking and advanced load-balancing knowledge.

After more than three years of interviewing candidates, the position remained empty. The company had painted itself into a corner.

During the engineering phase, create multiple design proposals. Consider each of them for its simplicity, manageability, cost, and so on. Revise, revise, revise. Good engineering is not about having the ability to create perfection in the first draft, but rather about being able to iterate over many revisions, improving each one until we have the design we want.

Case Study: Branch Office Prototypes

The network group at a company with many buildings around the world

found it difficult to maintain the ad hoc network solutions that had been deployed at each building. They decided to standardize.

To simplify the design, the team reduced the number of possible configurations to three: small, medium, and large. Next they created a standard template for the equipment and configuration that would be used in each scenario. Each configuration was deployed in a lab and connected as if it was an actual building. Complex routing protocol configurations were verified and documented. Operational procedures were tested and documented, including the processes for turning up a new building, turning one down, performing failovers between ISPs, repairing components, and so on.

In the prototyping stage, the team members kept asking themselves how things could be simplified further. Operational procedures were unified across the three configurations, naming and labeling standards were made uniform, and many more opportunities to simplify were found.

Each simplification made the systems easier to understand and manage.

17.3 Vendor-Certified Designs

Often third-party products have the engineering dictated to you. The vendor proscribes minimum CPU, RAM, and disk requirements. Often vendors publish **certified engineering designs** or **vendor best practices**. There may be multiple recommendations—for example, separate engineering recommendations for use with up to one dozen, one hundred, or one thousand users.

Even if one uses the certified engineering design exactly, some local decisions always need to be made: hardware vendor, model number, specific configurations, IP addresses, network switch connections, deployment racks, optional configuration parameters, and so on. This is the **local engineering design**.

We like when vendors provide best practice engineering designs because they've done much of the hard work for us. The designs are usually based on real-world experience. They have been tested in labs and the real world. The vendor has performed load testing to determine the maximum number of users or other capacity. It is best to test these assertions in a lab environment. The best approach is to trust the vendor recommendations, but verify them. Often, however, only larger companies have the resources to do so. Smaller companies just have to trust the vendors.

Straying too far from the vendor's recommendations will result in the loss of support. The vendor won't help you if you call customer support with a problem because you are using the product in a configuration the vendor does not certify.

Usually we must engineer how the system will be installed ourselves. Vendor-certified designs are relatively rare. Even when a vendor design exists, creating the local engineering design requires a lot of expertise and effort.

17.4 Dependency Engineering

The reliability of a service is greatly influenced by its dependencies. A system is never more reliable than the most unreliable system it has a hard dependency on. For example, if an application server can be accessed only via an unreliable network, the application will never be more reliable than that network.

The larger and more complex a system becomes, the more dependencies it has. We can manage these dependencies during the engineering process in ways that greatly improve the resulting system's reliability.

17.4.1 Primary Dependencies

Primary reliability refers to the core system reliability excluding any external dependencies. As system administrators, we often do not have much control over the reliability of the software itself, especially for third-party products.

What we can control are our decisions about the hardware it runs on. As discussed in [Chapter 13](#), "[Server Hardware Strategies](#)," enterprise software tends to be written assuming the hardware will never fail. In turn, we choose reliability features such as mirrored boot disks, ECC RAM, dual NICs, and so on. These issues have already been discussed in previous chapters and [Section 17.1](#).

17.4.2 External Dependencies

We can control many of the external dependencies that influence a service's reliability.

First we need to know what those dependencies are. A **dependency matrix** is a list of which subsystems depend on other subsystems. Usually the dependency matrix resembles a tree diagram, with each level relying on services in the levels below, and no cross-dependencies. When engineering a new service, you should create a dependency matrix to document the dependencies so that everyone reviewing the plan understands them. Such a matrix can be used to find high-risk dependencies and mitigate them. Once the service is launched, this matrix is useful for operational tasks such as basic debugging as well as disaster planning.

All dependencies are not the same. A **hard dependency** means the dependent service will fail if the other service fails or is unavailable. A **soft dependency** is one that does not directly cause a failure, though it may reduce performance.

For example, an application may have a hard dependency on an external storage server or SAN. If the SAN dies, the application dies with it.

In contrast, a DNS server is usually a soft dependency. Systems are configured to depend on multiple DNS servers. As long as at least one is reachable, the client is able to perform DNS lookups. There may be a performance penalty, but the client is able to operate. Note that the DNS *service* is usually a hard dependency, but a single DNS *server* is usually a soft dependency.

Applications often have soft dependencies on other applications. An accounting application may depend on another server to provide foreign currency exchange rate information. However, if that server is unavailable, the application may work around it by using cached data, disabling a specific feature, or requiring the data to be manually entered. This kind of soft dependency is called **graceful degradation**.

As we engineer a system and review the dependency matrix, we can improve the reliability by identifying hard dependencies and either eliminate them or reengineer them to be soft dependencies. For example, earlier we mentioned a service that was only as reliable as its least reliable network connection. That network connection could be reengineered to be a dual-connected connection, thereby ensuring the system can survive one network failure at any time.

NFS Dependencies Outside the Datacenter

The network file system protocol commonly used on Unix systems is called NFS. It has a feature whereby if the server goes down, the clients will freeze until it comes back up. This feature is useful in situations in which one would rather wait than have the client software lose data because a file system magically disappeared. Under this arrangement, an NFS server is a hard dependency for any client.

When Tom was at Bell Labs, a customer configured his desktop machine to be an NFS server and started providing some useful data there. Soon, many machines had automounted the disk volume from his machine, including some very important servers in the datacenter.

Then the customer powered off his desktop machine and left on vacation. All the machines that had accessed the data froze, waiting for the desktop machine to become live again. The SAs had to decide whether

to get corporate security to open his office door to boot up his machine or to reboot some very important servers.

In hindsight, the servers should have been configured to mount NFS volumes only from machines directly managed by the SA team, or possibly just configured not to mount from other NFS servers at all.

17.4.3 Dependency Alignment

Another engineering trick is to realign dependencies into smaller failure domains. A **failure domain** is all the services, locations, and users that are adversely affected when a particular component fails or is taken out of service.

Good engineering requires taking the time to think through how we can shrink failure domains, or divide large failure domains into smaller, isolated failure domains.

Hardware Failure Domains

Suppose three servers were being set up. Each server was made up of the CPU chassis, an external disk chassis, and its monitor. The original plan was to have one power strip for all the CPUs, one for all the disk chassis, and one for all the monitors. This made for a very uniform and elegant wiring diagram. However, it also meant that if, for example, the power strip for the disks died, all three servers would die. A better design was to divide the system into three smaller failure domains: Each power strip would connect to the CPU, disk, and monitor of a particular server. Each combination of power strip, CPU, disk, and monitor would be an isolated failure domain. In the original design the entire lot was one big failure domain.

A similar situation often happens in network engineering. Often a network team sets up VPN servers in locations around the world. At each location there is a redundant pair of VPN servers along with supporting devices such as network switches, power strips, and so on. The engineers configure each half of the pair to be an independent failure domain: one VPN device, power strip, UPS, and switch. A careless design or installer can create cross-dependencies that widen the failure domain to the size of the entire installation.

Service Failure Domains

The next example involves a common best practice used by Microsoft Windows administrators. When a user logs into an ActiveDirectory domain, an MS Windows login script runs that sets up the user's environment, configuring

printers and other needed resources. The best practice is to have this login script download from the same server as the user's home directory. This aligns the dependency on the file server with the dependency on that startup script. In fact, everything the script depends on should also be replicated to that server.

By aligning all the dependencies to the same machine that hosts their home directory, we know that the login process will have all of its dependencies available, or the process won't start at all. If the server with the home directory is unavailable, it doesn't matter that the other dependencies are also unavailable; the users can't log in anyway. Without such alignment, a single outage would affect all users. Replicating the script every time it is updated seems like a lot of work, but DFS Replication Service makes such replication automatic.

This pattern is repeated in other places. Consider a server farm that hosts many web servers, each of which depends on database servers. We can minimize the external dependencies of a web site by making sure that the web server and the databases it accesses are all on the same machine, or are all on machines on the same switch. It may be easier to simply land each new web site on whatever servers are available. By taking the time to consider the dependencies, we build a more reliable service.

Such alignments do not require tracing every cable and power cord. We can make matters easier by simply planning ahead. First we engineer each rack to be independent—self-sufficient in its networking and power by having its own network switch and power distribution units.

Now we make sure that all servers for a given service that have hard dependencies on each other are **rack aligned** or located in the same rack. Doing so limits the number of external hard dependencies that the service has and reduces the failure domain of the rack, because there are fewer services in any given rack.

If a service has components that are replicas or are in a master/slave configuration, we make sure that the replicas are in different racks, and that the master and slave are not in the same rack. Doing so means that the service has a soft dependency on any given rack, rather than a hard dependency.

Location Failure Domains

A common design pattern is to make each office self-sufficient for a limited and specific set of services. For example, one might define that if an office loses connectivity to the rest of the company, the following services should still work: DNS, DHCP, ActiveDirectory, home directories, email, and printing. Service

would degrade in predictable ways in this scenario: Email sent outside of the office would queue up until connectivity was repaired, and so on.

Once these requirements are defined, we can engineer those services to depend on only local services. All other services can rely on services outside of the office.

When introducing a new service, we must decide if it will be part of the self-contained realm. For example, a new VoIP service might be so fundamental to the company's operations and safety that it would be part of the self-contained list.

The same applies to applications. A new accounting-related service would be engineered to depend on only services in the same building in which it is deployed. If all of its users are in a single office, it might be deployed in a computer room in that office.

Sometimes, however, self-sufficient sites might not be economical. For example, some companies might set up hundreds of tiny sales offices, often with fewer than a dozen people in each. It would be cost-prohibitive to have each office set up with its own local services. Instead each office may be designed with no servers at all, relying on regional hub datacenters to provide DNS, DHCP, and all other services. With this approach, the amount of server equipment needed in each office is eliminated. If the office would be dead in the water when its WAN connection is down, having services working may be unimportant.

Catch-22 Dependencies

A startup web company found itself short of disk space and, to save money, exported some free disk space from a Unix desktop via NFS. This disk ended up being mounted on all the servers, since it provided a common component. After a block-wide power failure that outlasted the UPS, the company's network would not come up. The workstation couldn't finish booting without the DNS server running, and the DNS server required the NFS partition to be available to complete its own boot process. Shortly afterward, the company hired an SA.

17.5 Decoupling Hostname from Service Name

Picking the names for machines often becomes a hotly contested debate rivaling the controversies over topics such as Apple versus Microsoft, Vim versus Emacs versus Sublime, and whether sandwiches should be cut lengthwise or diagonally.

We won't enter into this debate. However, once you've picked a naming scheme, we have one bit of advice: Decouple the purpose of the machine from the name of the machine.

For example, consider an email service running on a machine called `mail`. What do you do when it is time to replace the mail server with a new one? You could call the new machine `newmail` or `mail2`, but then users would need to reconfigure their email clients, which would be a hassle. You could call the new machine `mail` and rename the old machine to be `oldmail`, but now everyone would start referring to the new host, possibly before it is ready.

A better choice is to design your systems to use aliases for the service name. For example, name the machines simply `server1`, `server2`, `server3`, and so on, but provide aliases such as `mail`, `calendar`, `wiki`, and so on. These aliases can be changed to point at new machines as needed.

In DNS, aliases are configured using their `CNAME` record, which lists the actual machine name, or `A` and `AAAA` records, which list the IP address of the machine. The benefit of using a `CNAME` is that it does not need to be updated if the IP address changes.

Very old software packages might not support the use of aliases. We consider this to be a bug. Also note that SSL certificates are based on the hostname that appears in the URL, not the hostname. Therefore web-based services that use HTTPS need additional planning.

Case Study: Splitting the Central Machine

As a small company, Synopsys started with the typical configuration of one central administrative machine. It was the Network Information Service (NIS) master, DNS master, time server, print server, console server, email server, SOCKS relay, token-card authentication server, boot server, NetApp admin host, file server, Columbia Appletalk Protocol (CAP) server, and more. It was also the only head—keyboard and monitor—in the machine room, so it was the machine used to remotely access all other machines. Plus SAs used it to check their email rather than run back to their desks.

As the SA group grew and new SAs were working at this machine's console, a particular error began to happen. Occasionally someone using the administrative machine to access other hosts would get confused and shut down the administrative machine instead of the intended host. Because everything relied on the central machine, this accident effectively

brought down the whole company network at once.

The time had come to split the functionality of the machine across multiple servers, not only because of those occasional slips but also because the machine was becoming increasingly unreliable and overloaded. At this point, the central machine had so many services running on it that just figuring out what they all were was a large task in itself.

Some services moved relatively easily because they were associated with a service-based name. Others were more difficult because they were tied to IP addresses.

Likewise, do not hardcode services to particular IP addresses. Listing an IP address in configuration files leads to trouble. List hostnames instead. If the IP address of a machine must be changed, we can change it in one place—DNS—and no configuration files must be altered. There are countless situations where the IP address space of a particular VLAN must change, usually due to mergers, reorganizations, or the need to physically move machines. In each case the project is made unnecessarily complex due to hardcoded IP addresses. If a service doesn't work, debugging becomes a hunt for the mystery location of a hardcoded IP address buried deep in a configuration file.

There are some exceptions to this rule. First, DNS clients must be configured using IP addresses, due to the Catch-22 that would happen if DNS clients needed to use DNS to find the IP address of a DNS server. NICs and network equipment are another exception for similar reasons. However, there are still a couple of ways that you mitigate against an eventual need to change IP addresses. Look for mechanisms that allow you to use a different service-based IP address in the configuration files, rather than the primary IP address of the machine itself.

Using IP Aliases

There are some other cases where IP addresses are commonly hardcoded into configurations. For example, IP addresses are often used in firewall rules. Likewise, the IP addresses of DHCP servers are often found in the router configurations. Routers are configured to relay broadcast DHCP requests from a particular network to a specified set of DHCP servers. This is called a DHCP relay or IP helper configuration.

DHCP servers can usually be configured with a second IP address (often known as a secondary IP address or an IP alias) that is used for the

service, and can be easily moved to another machine. A similar approach can often be used for other services that require firewall rules, with the secondary IP address being used for the service and in the firewall rules, making it much easier to move the service to another machine, along with the IP alias. When this secondary IP address is a loopback address that is advertised onto the network as host route, then it is truly independent of the VLAN, and can be easily moved, even to a completely different datacenter.

17.6 Support

During the planning and engineering phase, consider how the service will be supported during its lifetime. Support includes technical aspects such as detecting problems, patching, upgrading, performing backups, and scaling the service. It also includes people and processes, such as identifying and training the support groups, developing mechanisms for customers to report problems and submit requests (and how those problems and requests should be handled), and providing documentation for the customers and the support teams.

You need to know when there are problems with the service and when you should be thinking about scaling it up—therefore the service must be monitored. You will need to put some thought and effort into engineering the appropriate monitoring solution. You should also consider which requests users may make in relation to this service, and how those will be handled. Which automation and processes should be in place? Which documentation is needed to support the service?

17.6.1 Monitoring

It isn't a service if it isn't monitored. If there is no monitoring, then you're just running software. A service should be monitored for availability, problems, performance, and capacity-planning. Monitoring is covered in more detail in [Chapter 38](#), "[Service Monitoring](#)."

The helpdesk, or front-line support group, must be automatically alerted to problems with the service so that they can start fixing them before too many people are affected. A customer who notices a major problem with a service and has to call in to report it before the service provider discovers and begins to fix the problem is getting a very low standard of service. Customers do not like to feel that they are the only ones paying attention to problems in the system. Conversely, problems you can detect and fix before they are noticed are like

trees that fall in the forest with no one around to hear them. For example, if an outage happens over the weekend and you are alerted in time to fix it before Monday morning, your customers don't even need to know that anything went wrong. (In this case, you should announce by email that the problem has been resolved, so that you receive credit. See [Section 49.2](#).)

Likewise, the SA group should monitor the service on an ongoing basis from a capacity-planning standpoint. Depending on the service, capacity planning can include network bandwidth, server performance, transaction rates, licenses, and physical-device availability. As part of any service, SAs can reasonably be expected to anticipate and plan for growth. To do so effectively, usage monitoring needs to be built in as a part of the service.

During the planning and engineering phase of the project, you need to define what should be monitored, and how. You need to define which events should trigger alerts, and what the priority of those alerts should be. You need to work with the monitoring team to understand how this service gets integrated into the existing tools, and which work is required on both sides to make that happen.

You also need to define the process for adding new components into the monitoring system as the system is scaled up or upgraded. Likewise, you need to define how components are removed from the monitoring system as they are retired or repurposed. Ideally these changes should be automated, and driven from the inventory system. However, that means that this automation needs to be developed, and the processes need to be clearly defined and documented.

17.6.2 Support Model

The support model needs to specify who supports the various components of the service, and what the OLAs and SLAs for each component are. The OLA and SLA for the service need to take into account the OLAs and SLAs of the constituent parts, as well as the design of the service itself.

For example, if a service has components in multiple datacenters around the world, who provides hands-on support in the datacenters, and what are their SLAs? What are the vendors' SLAs for the various hardware components—do these differ in different locations? What is the SLA for WAN connectivity? Who provides local network support, and who provides WAN support? Who supports the OS that the application runs on? Who supports the application itself? What are the processes, contact information, SLAs, and escalation paths for all of these groups?

In a small or midsize company, many of these roles will be performed by the

same people, who all know each other. In large companies, there will be different teams supporting each component, they won't all know each other, and the processes and communication paths may not be clear. Make sure that all of this information is tracked down and documented in advance, rather than finding out in the midst of an incident that you do not know how to contact the on-site support staff in a remote location.

17.6.3 Service Request Model

For most services, the local engineering plan should also include a service request model (SRM). The SRM defines which requests users can make relating to this service, who can make those requests, how they make those requests, which approvals are required, who acts on the requests, which change control policies apply, and which SLA is used for turning around those requests.

Because each service is different, work with the vendor and stakeholders to define which tasks are required, document them, and then practice them in a test environment. Try to estimate the volume of requests of each type, and understand from the stakeholders what would trigger a request of each type and what kind of turnaround time they would expect on each request.

Consider whether it makes sense to automate these requests. For example, if a user-add request will always be generated as a result of a new hire, can you link to the HR system, so that these requests are automated? This will eliminate the need for someone to remember to submit that request, and someone else to process it. Likewise, user-deletes should occur automatically when someone leaves the company, and perhaps when someone moves to a different role in a group that does not need access. Alternatively, moves could generate access review tasks for the new manager and the service owner.

17.6.4 Documentation

As part of support planning, operational procedures must be defined. These are different for every service, but generally include backups and restores, business continuity or disaster recovery plans, tasks related to onboarding new users, disconnecting users who are leaving, performing periodic tasks, and anything else required to keep the service running.

For each routine task, there must be a runbook that details how the task should be performed. Define the runbooks by performing each task in a test environment. What is learned through this process may lead to changes in the architecture or local engineering plan. For example, you might discover that

backup and restore procedures were not considered until after the architecture was defined. This may require changes to the architecture. Alternatively, what is learned may be a minor point, but one that requires changes in the local engineering plan—for example, to define that each machine must have an additional fiber interface on a dedicated backups network. This is an iterative process.

The documentation must also include the steps to take when debugging the service. This documentation should cover which data needs to be gathered, and how, as well as steps to examine that data and use it to debug the problem. There should be documentation for the helpdesk telling these personnel how to perform basic health checks on the service, verify problems that might be reported by end users or the monitoring system, and debug and resolve the most common problems. [Chapter 29, “Debugging,”](#) describes the steps involved in debugging problems. The documentation should describe how to follow those steps for this service.

The helpdesk documentation should also specify which information needs to be gathered and passed on to the next support level if the issue remains unresolved and needs to be escalated. The documentation for the more senior SA team must include the vendor’s contact information and support contract number, all the components that the service depends on, and contact information for all the other teams that support those components. The contact information for other teams should be provided through a link to a well-known team support page that each team maintains itself, rather than duplicating information that may get stale into many different support documents.

17.7 Summary

Service engineering involves designing how a product will be configured and used in your environment. With a simple service, it is a matter of installing the software. Of course, most services are not simple. In such a case, we must design the service architecture and local engineering plans.

The service architecture is the high-level design. It describes how the parts will work together, which resources are required, and so on. Local engineering plans describe specifics for a particular device, such as its model, name, and IP address.

While a lot of science and methodology goes into such engineering, a few rules of thumb are helpful: Follow vendor recommendations, mirror your boot disk, use service-class hardware, don’t run a service on a machine under your

desk, restrict console access, and leverage the existing infrastructure as much as possible.

Service architecture should strive for simplicity. Revise the design over and over, tweaking and improving it. Consider how the dependencies will affect the service. Consider alternative plans that reduce dependencies by examining failure domains. We can align dependencies to make a system less susceptible to failures. We can align along hardware failure domains, service failure domains, and location failure domains.

When possible, do not expose actual hostnames to users. Instead, expose hostname aliases that can be moved to other machines. Do not hardcode IP addresses unless doing so is unavoidable.

The service architecture should be supportable. It should indicate how the system will be monitored, how it will be accessed for administrative purposes, and so on. Documentation should include a runbook for common tasks, expected failure modes, and debugging typical problems.

Exercises

1. What is a service architecture? What is a local engineering plan? How are they related and/or different?
2. Why is simplicity important in service engineering?
3. What are primary and external dependencies?
4. Which support services improve our ability to operate a service?
5. Pick a service you are familiar with, and list its primary and external dependencies.
6. What is dependency alignment? How can it be used to make a system less susceptible to failures?
7. For a service you are familiar with, describe how its dependency alignment could be improved.
8. Of the rules of thumb listed in [Section 17.1](#), which are the most important? Which were the most surprising to you?
9. Pick a commercial or open source product you are familiar with, and design a service architecture for deploying it to your organization.
10. Suppose you were part of the team described in the “[Branch Office Prototypes](#)” case study in [Section 17.2](#). How would you apply the small batches principle of [Chapter 2](#), “[The Small Batches Principle](#),” to that situation? Remember that the small offices are scattered around the world.

Chapter 18. Service Resiliency and Performance Patterns

This chapter includes a number of design patterns related to improving the resiliency of a system or its performance. These two issues are often related.

Products and services are often designed with some kind of internal redundancy to make the service more resilient or to enable it to scale to larger capacity. These techniques follow certain patterns. You will see them in many systems. This chapter's goal is to introduce you to the most common patterns so that you will be familiar with them when you see them in products you may be designing, purchasing, or administering.

First we discuss how extra hardware is used to provide resiliency. In this pattern, if one piece of hardware fails, there will be other hardware that can take its place. When there are two systems acting this way, the result is often called a master/slave relationship because one system is authoritative and in control.

Another design pattern involves having many systems that all provide the same kind of service. These are called replicas. In this case, each system receives and processes a fraction of the incoming workload. Clients send requests to a device called a load balancer, which then forwards a fraction of the requests to each replica.

While these patterns solve many problems, new issues arise. For example, how we share information, or state, between these machines is not as simple as it sounds.

Lastly, no one likes a slow service. This chapter ends with advice about making sure systems meet the performance expectations for the service. The first step is to understand how many simultaneous users (or workload in general) a system can handle. We can do a dataflow analysis to understand the theoretical limits of a system and load testing to test or prove our assumptions. We then explain how network latency can affect total system performance in real systems.

We cannot cover performance engineering and scaling in depth in one chapter. Volume 2 of this book series includes a broad overview of techniques. *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise* by Abbott and Fisher ([2010](#)) contains in-depth explanations and techniques.

18.1 Redundancy Design Patterns

Redundancy can provide greater performance or better reliability. Sometimes it can provide both. This section discusses common redundancy patterns. Then it covers how they can be used to provide either more performance or better reliability.

18.1.1 Masters and Slaves

The first category of patterns involves two equivalent sets of hardware. One is called the master and the other is called the spare or slave. The master provides services and the slave is, in general, idle but ready to take over if needed.

The master is in control and the slave follows orders. If there is data, or state, stored on these machines, the master holds the authoritative version of the data. If the slave holds the data as well, it is a copy, and often slightly out-of-date. Master/slave arrangements are generally created to improve a system's ability to survive or recover in case hardware fails.

While this sounds simple, implementing master/slave relationships is quite complex and there are many variations of how it can be done. Here are some of the most common patterns:

- **Master plus cold spare:** In this configuration the service runs on the master machine. The cold spare is a redundant set of hardware that is powered off, but can be powered on and configured to replace the master in the event that the master fails. The time it takes to bring up the cold spare could be hours or days, especially if one must restore data from the most recent backups. This pattern used to be very common when some downtime was permitted and the technology to provide a seamless failover was either nonexistent or too costly.
- **Master plus hot spare:** In this configuration the service runs on the master. The hot spare is a separate machine that receives updates such that if the master failed, it could take over the master's role with little or no data loss. Usually the master updates the hot spare in real time. Switching to the spare is a manual process.
- **Master/slave:** When switching to the hot spare is automatic, we call this a master/slave configuration. The slave does nothing but receive updates and waits for the master to die, in which case it takes over. It may assume the master's IP address so that clients see no outage, or simply retry a request and continue. When the failed machine comes back to life, it typically becomes the slave.

- **Master/read-only slave:** In this configuration the slave can handle some requests but not others. Typically it handles read requests—that is, requests that do not change the stored data or state. For example, an application would send database updates (writes) to the master but database reads to either the master or the slave. This improves performance since it reduces the amount of work being done by the master. The disadvantage of this pattern is that the slave may have data that is slightly out-of-date. An application may become confused if it performs an update and immediately sends a read request to the slave.
- **Master/master:** In this case both sets of hardware are able to perform both read and write operations. This is also known as a **load sharing** configuration. If either master fails, the other runs as normal. When the broken machine returns to life, it must sync up with the existing master before it begins providing service again. Coordinating such activity is very difficult. For example, if the failure occurs on the network between the two masters, both masters will think the other died and accept updates. As a consequence, their databases will diverge. When the network connection returns, both masters will be confused to find a master that already exists, with very different data. This is known as a **split brain** situation. Often the only way to resolve this conflict is for a human to pick one master and force the other one down. Because of the complexity of a master/master system, such software tends to be very expensive.

18.1.2 Load Balancers Plus Replicas

Another way of providing redundancy is to use a **load balancer plus replica** configuration. In this pattern, all clients send their requests to the load balancer. The load balancer is a host that simply forwards the request to one of many replicas. Each replica is configured with the same software and provides the same service. Thus, the workload is distributed over the many replicas. Each replica processes a fraction of the workload. If a replica dies, the load balancer stops forwarding requests to it. If a new replica is added, the load balancer adds it to the mix.

A typical example is a web server. Each replica has the same set of web pages and application software. The load balancer can forward any request to any replica.

Another benefit of such redundancy is that it makes upgrades easier. You can perform a **rolling upgrade**. One at a time, each replica is disconnected, upgraded, tested, and brought back into service. The outage of the single host

does not stop the entire service, but just reduces the total capacity of the system.

18.1.3 Replicas and Shared State

Since each replica needs to act the same way, the replicas cannot store any information that would make them unique. For example, if the service permits someone to upload a photo for his or her profile page, the photo cannot simply be stored on that replica. If it did, future requests that go to other replicas would not have that photo or be able to display it.

Thus, if a web-based service stores or changes information, the replicas usually store this information elsewhere, usually on a database server. The information that may change is called **state**. Replicas are stateless, meaning that they store no state. Databases are stateful, meaning that they store information that changes.

Many applications require a small amount of state to be shared. This is often shared in an in-memory database such as Memcached or Redis. These databases are extremely fast for storing small amounts of data that is temporary in nature.

Volume 2 of this book series goes into greater detail about load balancers and replicas.

HTTP Cookies Are Web State

When logging into a web site, a user enters a username and password. After that, how does the web site know it is the same user? Does the web browser send that username and password with all future HTTP requests? Not at all.

Suppose Alice enters her username and password to log into a web site. This gets sent as form-data to the HTTP server. If the password is valid, the HTTP server replies with a web page or message that informs Alice of her successful login. That reply also includes a unique cookie such as `LOGIN=1723832343207`. Alice's web browser will include that cookie with all future HTTP requests to that domain until it expires. When the server sees this cookie, it knows that the user is logged in and can determine which user it is. It can, therefore, generate web pages that are specific to Alice. In theory, the cookie is generated randomly and difficult to guess, which provides some measure of security.

When the cookie is initially created, it must be stored on the server so that later the application can use the cookie value to find which user is associated with it. Initially web sites stored this information on a local

disk. This pattern worked because it was fast and easy—but broke when the load balancer plus replica design pattern was invented. In that pattern, each replica created its own cookie list. If the next HTTP request went to a different replica, that replica wouldn't recognize the cookie value and the user would receive the web page for logged-out users.

The solution is to store the cookie list somewhere that all replicas can access, as depicted in [Figure 18.1](#). Initially sites used SQL databases since they were what was available. However, SQL databases can be slow. Since nearly every web page requires a cookie lookup, this slowed down the entire site, and put a huge burden on the SQL servers. Eventually sites switched to in-memory databases such as Memcached or Redis.

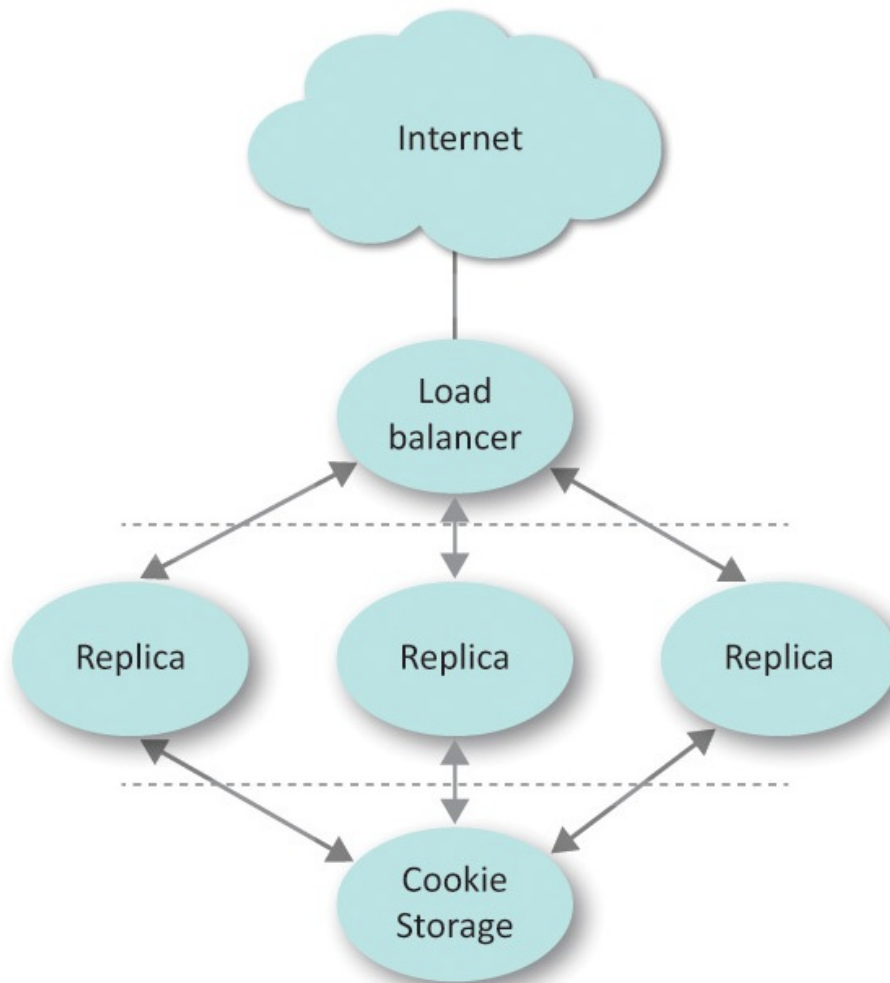


Figure 18.1: Web application with replicas and cookie storage

18.1.4 Performance or Resilience?

Performance is about making the system process work faster, or scaling to process more work. Resilience is about improving the system's ability to work around failures. It is also called survivability.

As we've seen, redundancy is sometimes used to improve performance and sometimes used to improve resilience. When used to improve performance, the work is distributed across the added components. When used to improve resilience, the added components take over if another component fails.

People often make the mistake of assuming that redundancy makes a system more reliable *and* faster. Do not make this assumption. Sometimes the extra reliability will come at a performance cost. Moreover, an incorrectly scaled solution with replicas may be no more reliable than a single-server solution.

For example, a database using the master/master pattern may be slower than a single server. The masters may share the load for read operations, but writes will require additional overhead to perform synchronization and other housekeeping. If a large percentage of the operations performed are writes, the system will be more reliable, but slower.

A web service using the load balancer plus replicas may not be resilient to failures. Suppose there are three replicas, each running at 100 percent utilization. If one fails, the remaining two replicas will receive the workload that had been processed by the failed replica. Each replica is now receiving 150 percent of the requests it was designed to handle, which usually will cause the replicas to become overloaded and crash or have other problems.

A system that has enough capacity to be fully functional with one component removed is called $N + 1$. If two components can fail, it is called $N + 2$, and so on. If an $N + 1$ system fails, the system is now $N + 0$. When the failing component is repaired, the system returns to $N + 1$.

Any design with redundancy should indicate whether the redundancy is used to increase performance capacity or to improve resilience. Otherwise, there will be confusion. If some members of the team think the redundancy is for resilience but the system is actually running at $N + 0$, someone might decide to take one replica down to do maintenance. At that point the system will be overloaded.

If we know that the redundancy is intended to improve survivability, then the system should be monitored to detect whether the system is $N + 1$ or better, and alert us if the system has become (or will soon become) $N + 0$. With a master/slave system this is easy to monitor. If one component goes down, we know we are at $N + 0$ and should work to rectify the problem. This is more

difficult with $N + M$ systems, because to know the value of M , we must know how much capacity each machine adds to the system. This is usually determined by load testing. That said, you should at least monitor how many replicas are down so that they can be repaired before the system is less than $N + 0$.

18.2 Performance and Scaling

No one likes a slow service, even if it has every imaginable feature. From a customer's perspective, two things are important in any service: Does it work, and is it fast? *Work* covers such areas as reliability, functionality, and the user interface. When designing a service, you also need to pay attention to its performance characteristics, even though many other difficult technical challenges need to be overcome. If you solve all those difficult problems but the service is slow, the people using it will not consider it to be a success.

Performance expectations increase as networks, graphics, and processors get faster. Performance that is acceptable now may not be viewed so favorably a year from now. Bear that in mind when designing the system. Often a small added cost to over-provision a system can avoid a large upgrade in the future.

To build a service that performs well, you need to understand how it works and perhaps look at ways of splitting it effectively across multiple machines. From the outset, you also need to consider how to scale the performance of the system as usage and expectations rise above what the initial system can do.

With **load testing**, you generate an artificial load on a service and see how it reacts. For example, you might generate 100 hits per second on a web server, and measure the latency, or the average time to complete a single request. Then you might generate 200 hits per second, and see how the behavior is affected. You would continue to increase the number of hits until you see how much load the service can take before the response time becomes unacceptable.

Your testing may show that the system runs well with a few simultaneous users, but how many resources—RAM, I/O, and so on—will be consumed when the service goes into production and is being used by hundreds or thousands of users simultaneously? Your vendor should be able to help with these estimates, but conduct your own tests, if possible. Don't expect perfect accuracy from a vendor's predictions of how your organization will use its product.

Bad Capacity Planning, Bad First Impression

Always purchase servers with enough extra capacity to handle peak utilization, as well as a growing number of customers. A new electronic

voucher system deployed at one site was immediately overwhelmed by the number of users accessing it simultaneously. Customers trying to use the system found it impossibly slow and unreliable and soon switched back to the old paper-based method. This situation gave a bad first impression of the service. A memo was sent out stating that a root-cause analysis was being performed and that the system needed more RAM, which would arrive shortly. Even when the new RAM arrived, however, uptake was slow due to the bad first impression. Users preferred to stick with what they knew worked.

This new system had been projected to save millions of dollars per year, yet management had skimped on purchasing enough RAM for the system. The finance group had hoped that the new system would be wildly popular and the basis for numerous future applications, yet the performance was specified for an initial small capacity rather than leaving some room for growth. This new service had been given a lot of internal publicity, so the finance group shouldn't have been surprised that so many people would be trying the service on the very first day, rather than having a slowly increasing number of customers. Finally, the finance group had decided to flash-cut the new service rather than gradually introduce it to more divisions of the company over time. (Flash cuts are something we discuss more, and advocate against, in [Section 21.7](#).)

The finance group learned a lot about introducing new electronic services from this experience. Most important, the group learned that with customers, the old adage “Once bitten, twice shy” holds true. It is very difficult to get customers to accept a service that has failed once already.

Performance of the service for remote sites may also be an issue because of low-bandwidth and high-latency connections (we give advice about latency later in this chapter). Some applications enable you to place servers in different locations, and to use a global load balancer to direct traffic to the closest instance. For applications that are not designed that way, you may need to become very creative in providing reasonable remote performance if the service has a lot of network traffic. In some cases, quality of service (QoS) or intelligent queuing mechanisms can be sufficient to make the performance acceptable. In others, you may need to look at ways of reducing the network traffic.

18.2.1 Dataflow Analysis for Scaling

If you understand the individual components of a typical transaction in a service, you can scale the service with much greater precision and efficiency. Strata's experiences building scalable Internet services for ISPs and ASPs led her to create a dataflow model for individual transactions and combine them into spreadsheets to get an overall dataflow picture.

A dataflow model is simply a list of transactions and their dependencies, with as much information as can be acquired about resource usage for each transaction. That information might include the amount of memory used on the server hosting the transaction, the size and number of packets used in a transaction, the number of open sockets used to service a transaction, and so on.

In modeling an individual service transaction with dataflow, all the pieces of the transaction necessary to make it happen are included—even such pieces as Internet name lookups via DNS—to get a true picture of the transaction. Even things technically outside your control, such as the behavior of the root-level name servers in DNS, can affect what you're trying to model. If a transaction bottleneck occurred in the name-lookup phase, for instance, you could internally run a caching name server, thus saving some time doing external lookups. Sites that keep and analyze web service logs or other external access logs routinely do this, as it speeds up logging. For even faster logging, sites may simply record the external host by IP address and do the name lookups in a post-processing phase for analysis.

A nice thing about a service is that it is generally transaction based. Even file sharing consists of multiple transactions as blocks are read and written across the network. The key part of dataflow modeling to remember is that service transactions almost always depend on infrastructure transactions. It's fairly common to investigate a scaling problem with a service and discover that the service itself has encountered a bottleneck somewhere in the infrastructure.

Once the dataflow model is accurately depicting the service, you can address performance and scaling problems by seeing which part of the dataflow model is the weakest point, and then monitoring each piece, under real or simulated conditions, and seeing how it acts or how it fails. For example, if your database can handle up to 100 queries per second, and if you know that every access to your web site's home page requires three database queries, you can predict that the web site will work only if there are no more than 33 hits per second. However, you also now know that if you can improve the performance of the database to be 200 QPS—possibly by replicating it on a second server and dividing the queries between the two—the web site can handle twice as many hits per second, assuming that no other bottleneck is involved.

Resources on the server can also be an issue. Suppose that a server provides email access via IMAP. You might know, from direct observation or from vendor documentation, that each client connected to the server requires about 300K of RAM. Looking at the logs, you can get an idea of the usage patterns of the server: the number of users who are on simultaneously during certain parts of the day versus the total number of server users.

Knowing how many people are using the service is only part of the process. To analyze resources, you also should consider whether the IMAP server process loads an index file of some type, or even the whole mailbox, into memory. If so, you need to know the average size of the data that will be loaded. The data size can be calculated as a strict average of all the customers' index files. The average can be taken as a mean or median, based on where in the size curve most index files occur. Another viable solution is to add up only the index files used during peak usage times and perform the mean or median average calculations on that sum. Pick what seems to be the most realistic case for your application. The monitoring system can be used to validate your predictions. It might turn up unexpected findings, such as that the average mailbox size grows faster than expected. This growth rate might affect index size and thus performance.

Finally, step back and do this kind of analysis for all the steps in the dataflow. If a customer desktop makes an internal name-lookup query to find the mail server rather than caching information on where to find it, that fact should be included in your dataflow analysis as load on the name server. Maybe the customer is using a webmail application, in which case the customer will be using resources on a web server, whose software in turn makes an IMAP connection to the mail server. In this case, there are probably at least two name lookups per transaction, since the customer desktop will look up the webmail server, and the webmail server will look up the IMAP server. If the webmail server does local authentication and passes credentials to the IMAP server, there would be a third name lookup, to the directory server, and then a directory transaction.

Dataflow modeling works at all levels of scale. You can successfully design a server upgrade for a 30-person department or a multimedia services cluster for 3 million simultaneous users. It might take some traffic analysis on a sample setup, as well as vendor information, system traces, and so on, to get exact figures of the type you'd want for the huge-scale planning.

An Example of Dataflow Analysis

Strata once managed a large number of desktops accessing a set of file

servers on a network. A complaint about files being slow to open was investigated, but the network was not clogged, nor were there unusual numbers of retransmits or lags in the file server statistics on the hosts serving files. Further investigation revealed that all the desktops were using the same directory server to get the server-to-file mapping when opening a file and that the directory server itself was overloaded.

No one had realized that although the directory server could easily handle the number of users whose desktops mapped to it, each user was generating dozens, if not hundreds, of file-open requests to compile large jobs. When the requests per user figures were calculated and the number of simultaneous users estimated, it was then easy to see that an additional directory server was required for good performance.

18.2.2 Bandwidth Versus Latency

The term **bandwidth** refers to how much data can be transmitted in a second; **latency** is the delay before the data is received by the other end. A high-latency link, no matter what the bandwidth, will have a long round-trip time: the time for a packet to travel and the reply to return. Some applications, such as non-interactive (streaming) video, are unaffected by high latency. Others are affected greatly by this factor.

Suppose that a particular task requires five database queries. The client sends a request and waits for the reply. This is done four more times. On an Ethernet network, where latency is low, these five queries will happen about as quickly as the database server can process them and return the result. The complete task might take a second. But what if the same server is in Mumbai and the client is running on a machine in New York City? Suppose that it takes half a second for information to travel from New York City to Mumbai. Data can travel only as fast as the speed of light, and usually travels much more slowly due to delays added by routers, repeaters, and other devices along the network path. Now the task will take six seconds (one-half second for each request and each reply) plus the same one second the database takes to do its work. That's a lot slower than the original Ethernet time. It may not sound like a lot, but to an interactive user, it is noticeable and annoying. Now consider that when this kind of task is done thousands or millions of times each day, the many tiny delays add up to a significant amount of time.

Suppose that the link to Mumbai is a T1 connection (1.5 Mbps). Would upgrading the link to a T3 connection (45 Mbps) solve the problem? No. A T3

link adds bandwidth but does not change the speed of light. Such an upgrade will not improve the situation (unless there are additional queuing delays because the link is saturated).

One solution might be to launch all five queries at the same time and wait for the replies to come back as each of them completes. If each query is independent of the others, this is an easy solution. Better yet is when five queries can be replaced by a single high-level operation that the server can perform itself. For example, if the code performed a series of SQL queries and then summed the result, we could send one long query that gathered the five pieces of information and summed the result at the server. With this strategy, we reduce the number of round trips from five to one.

Mathematically speaking, the problem is as follows: The total time to completion (T) is the sum of the time each request takes to complete. The time it takes to complete each request is made up of three components: sending the request (S), computing the result (C), and receiving the reply (R). This is depicted mathematically as

$$T = (S_1 + C_1 + R_1) + (S_2 + C_2 + R_2) + \cdots + (S_n + C_n + R_n)$$

In a low-latency environment, $S_n + R_n$ is nearly zero. Thus, programmers may forget that it exists, or worse, think incorrectly that the formula is

$$T = C_1 + C_2 + \cdots + C_n$$

Programs written under the assumption that latency is zero or near-zero will benchmark very well on a local Ethernet network, but perform terribly once put into production on a global high-latency wide area network (WAN). This can make the product too slow to be usable. Most network providers do not sell latency, just bandwidth. Therefore their salesperson's only solution is to sell the customer more bandwidth, and as we have just shown, more bandwidth won't fix a latency problem. We have seen many naive sites unsuccessfully try to fix this kind of problem by purchasing more bandwidth.

There are a number of ways we can change the algorithm to work around latency issues:

- **Move work to the other side.** As in the earlier example, combine work done in small steps into one larger request performed by the server in one round trip.
- **Send requests in parallel.** If the requests are independent of one another, they can all be sent at the same time, and we can wait for the replies to arrive in any order.

- **Create windowed requests.** If sending all the requests in parallel would overload the server or the network between the two, a more sophisticated algorithm can limit the flow to n outstanding requests at any given time. This n is called the maximum window.
- **Query lists instead of individual items.** Change the protocol to accept a list of data instead of requiring one request per datum. For example, one graphic system required clients to verify that the fonts it had locally were also available on the server one font at a time. This added minutes to the startup time when used on high-latency links. The protocol was changed so that the client could send the list of fonts and receive a list of status results.

Applications such as streaming video and audio are not as concerned with latency because the video or audio packets are being sent in only one direction. The delay is unnoticed once the broadcast begins. However, for interactive media, such as voice or video chatting, the latency is noticed as a long pause after one person stops speaking and before the other person starts.

Even if an algorithm sends only one request and waits for one reply, how those messages are sent can make all the difference.

Case Study: Minimizing Packets in High-Latency Networks

A global pharmaceutical company based in New Jersey had a terrible performance problem with a database application. Analysis found that a single 4,000-byte SQL request sent over a transatlantic link was being sent in 50 individual 80-byte packets. Each packet was sent only when the previous one was acknowledged. It took 5 minutes for someone to log in.

The developers had been demanding additional transatlantic bandwidth, which would have taken months to order, been very expensive, and proved disappointing when it didn't solve the problem. When the system administrators reconfigured the database connector to send fewer larger packets, the performance problem went away.

18.3 Summary

This chapter introduced a number of design patterns and scaling techniques with which all SAs should be familiar. You will see them when evaluating products, designing services, and administering services.

Master/slave is a pattern where one system provides service and an equal system takes over if the first one fails. Load sharing between many redundant

systems (replicas) is done via a load balancer that directs proportional amounts of traffic to each replica.

Each additional replica adds a certain amount of capacity or horsepower to the service. If any replica is down, the load balancer divides its work among the remaining systems. This helps work around failures and also makes it possible to do maintenance or upgrades. However, this scheme requires spare capacity on the remaining replicas equal to the capacity lost from the failed one. One should know how much spare capacity is required to withstand one or more failures. A system that can tolerate one failure is called $N + 1$. The utilization should be monitored and alerts should trigger when the spare capacity is dangerously low.

Sharing state between replicas is difficult. In a web-based service, if a user logs in while connected to one replica, the other replicas need to know that the user is logged in. Otherwise, each request may or may not respond appropriately. Usually the replica that the user logs into sends a cookie to the HTTP client. The replica must communicate this cookie's information to the other replicas. This shared state is usually put in a high-speed database that all replicas access.

An overloaded system can be slow or fail. Determining how much load a system can handle while still providing an acceptable response time can be done a few different ways. Dataflow analysis involves modeling the system to understand how many transactions subsystems will perform for each transaction. Load testing involves sending requests and measuring the resulting response time.

Latency, or the time information takes to travel, is often an issue when a system that talks to a subsystem at great distance, or a web browser talks to a distant server. Some designs tolerate high latency better than others.

Exercises

1. When would two systems be configured as a master/slave pair?
2. In your environment, what are some examples of master/slave, master/master, and load balancing?
3. What is a load balancer? When do replicas need to share state?
4. Compare load testing to dataflow analysis.
5. How would you load test a service you run?
6. What is the difference between latency and bandwidth? If a system is slow because of a high-latency link, why can't a faster network connection improve the situation?

7. Measure the latency between your home computer and various work servers.
8. Explain to a non-SA the difference between latency and bandwidth.
9. How can you mitigate a performance problem caused by high latency?
10. Log into a web site and view the cookies stored in your web browser as a result. On another computer, verify that you are not logged into that web site, and then copy those cookies to this other computer. Are you logged in? Why or why not? What does this mean about how cookies should be stored and transmitted?
11. Suppose we save money by having the slave in a master/slave pattern be a slower, less expensive machine. When the failed master came back to life, we could switch back to it instead of letting the slower machine remain as the master. What would be the benefits and disadvantages of this?

Chapter 19. Service Launch: Fundamentals

This chapter is about launching a new service such as an email system, a VPN server, an OS installer, or a web-based application. The way that a new service is launched is every bit as important as the way that it is designed. The customers' first experiences with the service will color the way that they view the service in the future. First impressions last, especially bad ones.

Launching a new service can be the most exciting part of a job. It is the culmination of intense work by many people. Once launched, the service is something you point at and are proud of. While running the service will fill more calendar months, the launch itself is more intense and requires sustained focus.

This chapter focuses on the initial launch of a service, using our recommended six-step process. [Chapter 20](#), “[Service Launch: DevOps](#),” also deals with launches but covers more advanced techniques. [Chapter 21](#), “[Service Conversions](#),” covers the special case where a launch requires users to be migrated from an old service to the new service. Volume 2 of this book series covers related topics such as techniques for upgrading live services.

19.1 Planning for Problems

While a service launch is exciting and fulfilling, the problem with launching a new service is that something always goes wrong. The road to launching a service is full of tricks and traps that catch us in the most non-obvious ways: Users with a particular web browser can't access it, the service doesn't work for remote users, the production environment doesn't have enough space, the documentation is confusing, and so on.

We should be optimistic that a newly launched service will be a success. However, it is irrational to expect a flawless launch. That said, we tend to be surprised when there are problems, and we don't plan countermeasures for common problems. This chapter is about the work (and it *is* work) we must do to make the reality match our optimism.

Luckily, the more we launch services, the more skilled we become. We accumulate knowledge that helps us anticipate and overcome obstacles. We look for potential problems when the system is being designed rather than after it is built. We improve our ability to test edge cases. We find better ways to communicate, delegate, and collaborate. Every problem has a solution. The more we plan, the better the outcome. The more we launch, the easier it gets.

The popular perception of the NASA Apollo program is that only the Apollo 13 mission had problems. People typically think the other missions were trouble-free. In fact, this is not true. Every Apollo mission was fraught with problems. Apollo 7 experienced an assortment of minor hardware problems. During Apollo 8, parts of the computer's memory were erased, causing it to forget its position. During Apollo 12, the instrumentation panel lost power 36 seconds after launch, and had to be manually switched to backup power.

The possibility of each of those problems had been foreseen and NASA had worked out step-by-step plans for what to do. Astronauts had practiced those plans. What made Apollo 13's problem special was that it was unforeseen. There was no playbook that the astronauts and NASA engineers could turn to. It was an unplanned problem and there wasn't much time to invent a solution. The difference between a problem and a crisis is preparation.

As we gain experience we become better at foreseeing potential problems. We keep a checklist of potential issues to plan for. A typical system administrator accumulates this checklist in his or her head and uses past experience to be better prepared in the future. A good system administrator keeps the checklist written somewhere for reference. A great system administrator maintains this list in a way it can be used and updated by everyone in the organization. Each new launch problem results in updates to the list. As a result, with each new experience the entire organization gets smarter.

19.2 The Six-Step Launch Process

The general launch process relies on two major concepts: the **ready list** and iterations of launches and learning.

The ready list defines what it means to be ready to launch. We work on these items either directly or via delegation. The list is dynamic, with new items being added as bugs or regressions are found. Once all the items on the ready list are satisfied, we can attempt to launch the service.

The launch itself is a loop whereby we do a small launch, learn from it, then repeat. For example, the initial launch may be a beta release rather than an actual launch that is visible to customers. Each iteration may result in adding more items to the ready list based on what we learned.

After the system is launched and fully in production, there is more learning to do. We pause to perform a **retrospective** where we look back and reflect on the entire process to see what we learned and how we can improve future launches.

The entire process can be summarized as follows:

- **Step 1: Define the ready list.** Define a list of tasks that must be complete before the launch may proceed.
- **Step 2: Work the list.** Schedule, work on, and mark as complete each item in the list.
- **Step 3: Launch the beta service.** Launch to one of many test areas where the assembled pieces are verified.
- **Step 4: Launch the production service.** Launch the service into production.
- **Step 5: Capture the lessons learned.** Capture the lessons learned for posterity and to educate the organization.
- **Step 6: Repeat.** Begin the process again.

19.2.1 Step 1: Define the Ready List

The ready list is a list of tasks that must be completed or assertions that must be satisfied to indicate that we are ready to launch. In other words, this is the list that tells us when we are done.

Maintaining this list helps team members keep focus and avoid work that is unrelated to the shortest path to launch. It communicates your progress to management, other teams, and customers.

What to List

Items on the ready list generally fall into four categories:

- **“Must have” feature set:** There may be other features requested, but the launch cannot happen without these features. This part of the list serves as a contract between the customer and the provider. Setting expectations in this way prevents the problem where, after launch, stakeholders are surprised that certain features are missing.
- **“Would be nice” feature set:** All features that are not must-haves fit in this category.
- **Bugs to be fixed:** Testing will reveal bugs that must be tracked. If not tracked, they will not be fixed, or there will be a duplication of effort as multiple people try to fix the same issues.
- **Assertions and approvals:** This is the list of things that must be true before launch. For example, the legal department has approved the trademark, security audits are complete, the CEO and head of product management have signed off, and so on. This checklist contains technical

items such as verifying disk capacity or confirming that a particular ISP connection is up.

List Attributes

At a minimum each item in the list should have a brief title, a priority, the name of who is responsible for completing the task, a link to the history of the task, and a status.

The priorities should be standardized among all projects to avoid confusion, to make it easier for people to move from project to project, and to prevent employees from spending time reinventing the wheel with each project. For simplicity's sake, we focus on three priorities: must have, low priority, and not required.

If there has been a decision to not include a feature, it may be less confusing to list it as “not required” than to delete it from the list. If it is deleted, people will be unaware of the decision to not include the item in the launch.

If your organization has a standardized priority scheme for bugs, adopt it for use on the ready list to avoid confusion. For example, at Google there is a carefully defined set of bug priorities from P0 to P6. Of those, the definitions of P1, P2, and P5 map well to the must have, low priority, and not required priorities. Therefore we would adopt those priorities even though they are not contiguous numbers.

Items Should Be Specific and Unambiguous

The description of an item should be measurable. “Make things better” is ambiguous. “Improve sort performance by 20 percent” and “System enables RAID controller” are better. One way to make sure goals are well written is to follow George T. Doran's advice in his 1981 article, “There's a S.M.A.R.T. Way to Write Management's Goals and Objectives.” One is in a better position when goals are specific, measurable, achievable, relevant, and time-bound:

- **Specific:** The goal addresses a specific area for improvement.
- **Measurable:** The goal's success (or progress) can be quantified.
- **Achievable:** It can be realistically achieved given available resources.
- **Relevant:** It is worthwhile, and in line with broader goals.
- **Time-bound:** The goal has a deadline.

Doran's article discourages us from trying to achieve all five qualities for

every item. For example, listing a deadline is overkill for simple bug reports.

Storing the Ready List

The list should be viewable to all stakeholders—both people on the team and external people. This transparency enables everyone to understand what is going on outside their silo. It also helps management see progress and spot bottlenecks. Updates to the list should be made by those doing the work rather than requiring all updates go through a central coordinator. This improves agility.

Following are the most common places to store and maintain the list. Pick whatever fits your team and corporate culture the best.

- **Bug or ticket system with tags:** One method is to assign a bug ID or ticket to each item in the list and label the items in a way that one can easily query for all the items. For example, some systems permit tagging. One assigns a tag to all items in the ready list. For example, if the project is called “email2017,” each item is tagged as `tag:email2017launch01`. Publish a launch status page URL that queries for items with that tag, sorted first by priority and then by status.
- **Bug or ticket system with watchlists:** Some issue tracking systems permit one to create a “watchlist” or “hotlist” of items, which provides similar functionality to tagging. Issues are added to the watchlist. People who are subscribed to that watchlist can treat the list as a unit, with the ability to sort and filter the items within it.
- **Kanban board:** Kanban boards can be either index cards on a wall or electronic equivalents such as LeanKit or Trello. Cards are placed in columns: a “backlog” column for issues that aren’t being worked on yet, an “active” column for issues that are in progress, and a “completed” column for items that are finished. The history of the item is stored on each card, and may include a URL to a bug or ticket if needed.
- **Online or shared spreadsheet:** One can list each item in a row of a spreadsheet, using columns for title, priority, and so on. Web-based spreadsheets can be viewed by anyone at any time, and with the right permission system can be updated by anyone on the team. For simple items, you can keep the history in the spreadsheet; for more complex items, you can list a URL of the related bug or ticket.
- **Offline status spreadsheet coordinator:** With this option, a designated

coordinator maintains a spreadsheet listing the items and their statuses. All updates go through this person, and he or she emails the updated spreadsheet to all team members periodically. For example, if you have completed an item, you would email the good news to the coordinator, who then updates the spreadsheet. Once a week the coordinator emails the latest spreadsheet file as an attachment to everyone involved. We recommend against using this technique, for several reasons. People will always be working from outdated information. Having a third party update the spreadsheet can be a source of errors. The coordinator becomes an information bottleneck. Transparency is reduced. This method may have made sense before the web, but now sharing information is easy and convenient. We list this technique here only so you can recognize it when you see it.

The benefit of tagging or using a watchlist is that it keeps both the items and their history in one place. The Kanban system is growing in popularity as new online tools are making it easier to use. It provides transparency and a more modern way to set the cadence of the process.

Consider using systems that give a visual sense of accomplishment. Completed items should stay on the list but be marked as complete by changing their color or font, or by moving them to a list of completed items. If the system you use makes any completed items disappear, you are literally hiding your accomplishments.

Making accomplishments visible motivates the team. The team will see the completed portion growing over time, which gives them a sense of accomplishment. It is emotionally draining to have your accomplishments hidden, and to not see the progress you and the team are making.

A visual indication of progress also helps communicate with management. You want to accurately communicate progress to management, especially executive management. If they peek at the list weekly, they should see more and more items being marked complete. If what they see is a list of what's left to do, the subliminal message is that you haven't even started. Upper-level managers have too much on their plate to remember what the list looked like last week. Expecting someone to remember that last week there were 20 items and this week there are only 15 is expecting too much.

19.2.2 Step 2: Work the List

Now we work through the items in the list. Some we do ourselves, others we

delegate.

If you have never managed a project before, it may surprise you how much work is required to keep people accountable. If they do not feel accountable for the work they agree to do, they will not do that work. This is not because they are evil or lazy, it is simply that they are generally overloaded. Urgent items get attention and others slip. If we overdo it, however, we are perceived as nagging or micromanaging. The ready list helps keep people accountable and on schedule without requiring you to nag or micromanage.

First, make sure that everyone knows what is expected of them and their deadlines. Don't assume they "just know." You can do this by making sure the ready list includes the specific deliverable rather than being vague. For example, "Installer works on Dell 740DX hardware" is better than "Improve installer reliability."

People should be aware of a task's deadline. Therefore deadlines should be agreed to by all and recorded in the ready list so there is no ambiguity.

People are more likely to stick to deadlines when they have control over them. For this reason, it's best not to set a deadline, but rather ask the person taking the task to suggest a deadline. Nine times out of ten, that person will suggest a deadline that is the same or sooner than what you expected. You can accept this suggestion and the person will feel good because he or she was in control. You will have to negotiate an earlier date only if the individual suggests something that would be incompatible with the overall schedule. Even then, because the date was agreed to by negotiation, the person will feel some sense of control.

Second, keep people on track. Now that people know what they must do and the deadline, it becomes your job to make sure they are held accountable.

Some teams use weekly status meetings to monitor progress. The purpose of the meeting is to verify that deadlines are being met and to troubleshoot any roadblocks that might prevent deadlines from being met.

Some teams start each day with a very fast status meeting called a "stand-up." This technique is borrowed from Agile methodology. People stand in a circle and take turns announcing three things: what they completed yesterday, what they plan on doing today, and which roadblocks they are facing. To keep the meetings brief, there is no discussion, just the individual announcements. Because people are standing, there is a physical reason to keep the meeting short. Assisting co-workers with their roadblocks is done only after the meeting ends. As the meeting breaks up, people socialize a bit, as people often do. However, now it is productive socialization, because people initiate ad hoc discussions about the

various roadblocks raised. It is more efficient to have two people discuss a roadblock together than to have the entire team pause and wait for the length of the discussion during the meeting.

Single-Person Teams Need a Ready List, Too

You might not think you need to maintain a ready list if you are a solo SA or the only person on the project. The opposite is true. Keeping other people apprised of your status can be even more important in this situation. It keeps the work you do visible to management, and keeps customers aware of the status in a way that prevents them from bothering you to ask, thereby distracting you from your work.

19.2.3 Step 3: Launch the Beta Service

Before launching to real customers, the service should be launched to an isolated environment for testing. The isolated environment is called a **staging area**. The bugs and issues found there should be added to the ready list.

The concept of staging areas usually applies to software releases, but you can actually apply it to any service launch. You may be introducing a new OS release to your automated OS installation system; a beta test would involve testing it with friendly users. You may have a process for onboarding customers that is being reengineered; a beta test would involve walking through the new process with a person who pretends to be a newly hired employee.

There are many different types of staging areas. Each has a different name and a very specific purpose.

- **Dev:** The Dev environment is the Wild West of staging areas. It is where developers can deploy to at any time for whatever testing purposes they have. There are no gates.
- **Quality Assurance:** The QA area receives a release candidate from the developers. The QA process tests the release and either accepts or rejects it, and provides a list of bugs found. New releases enter this stage based on QA process schedules.
- **User Acceptance Testing:** The UAT area is a sandbox for external customers to test a release. They bring real-world data to the system and verify their procedures work, or learn how to adapt them to the new release.
- **Beta or Early-Access:** The Beta area is where approved customers have

early access to a release so that they may provide feedback or report bugs.

- **Production:** This is where live users use the service.

Each stage has entry and exit criteria. The entry criteria determine when the testing starts. For example, the QA stage receives a release candidate from development, whose intention is to deliver a software package that the developers believe is ready for use. The exit criteria restrict or **gate** whether the release passes or fails. For example, the gate to successfully leave QA may be that all automated and manual tests pass. The gate to successfully leave UAT may be zero P0 bugs and no more than three P1 bugs. The larger the project, the more important it is that these exit and entry criteria be explicit and written.

19.2.4 Step 4: Launch the Production Service

Finally we are ready to launch. The must-haves are done. As many of the nice-to-haves as possible are complete, too.

Launching is both a technical task and a communication task. The technical task involves pushing code to the production environment and any other step required to expose the new release to users.

The launch may require some kind of final approval. This often takes the form of executive approval, a product manager who signs off, or a senior team member's nod of agreement. Later we'll discuss replacing such mechanisms with comprehensive automated testing.

Communication is equally important. We need to communicate that the launch is happening, what is new since the last release, and most importantly whether any items are incomplete (these need to be clearly enumerated).

You should never launch to all users at once. That is too risky. Launch to a few users and expand. Go beyond a simple beta user process. Release to waves of users, such as different departments or teams, starting with the most risk tolerant and ending with the most risk averse. Performing gradual rollouts is a theme you will see throughout this book.

19.2.5 Step 5: Capture the Lessons Learned

Congratulations on the launch! The next step is to learn from this experience. A retrospective is a live meeting where the launch process is reviewed. It is called a retrospective because it involves looking back at what happened. We determine what went well, what didn't go well, and what needs to be improved. The purposes of the retrospective are fourfold:

- To get agreement about needed improvements from the project members.
- To educate the entire team about issues visible to only a few.
- To communicate problems to management, particularly ones large enough that their resolution will require additional resources.
- To spread what you learned to the entire organization so all can benefit.

It is best to do a retrospective in a live meeting with all the stakeholders and team members in one room, or in a video conference. Use this meeting to build a document collaboratively. This document should then be disseminated to all involved, and placed on a wiki so it can be reviewed. A retrospective document should have sections including an executive summary, a timeline, successes and failures, and short-and long-term changes that should be made.

- **Executive Summary:** What happened, when, and briefly what went well and what should be improved in the future. This section should be written in nontechnical, business language that any executive could understand.
- **Timeline:** A high-level summary of when the project started, when it ended, and major milestones in between.
- **Successes:** What went well. Celebrate what you did right.
- **Failures:** What didn't go well. Be honest about areas of improvement. Do not phrase them in terms of blaming individuals, but describe the facts of what happened and how it affected the project.
- **Short-Term Changes:** What should be done differently; the low-hanging fruit that can be achieved simply.
- **Long-Term Changes:** What should be done differently; changes that are large enough that making the change may require the creation of a project, assigned resources, budget approval, and so on.

Once a retrospective document is complete, each of the short-term and long-term changes should be entered into the bug tracking system so they can be triaged along with other kinds of requests. This is how we make sure that operational concerns are not forgotten.

Creating a retrospective document and publishing it for the entire company to see is actually quite a radical move. Traditionally businesses announce their accomplishments and hide or deemphasize their problems. Individuals tend to hide their mistakes, struggle through their problems silently, and confide in only a small group of people about their shortcomings. Here we not only discuss them but also celebrate them by documenting them for everyone to see.

We do this because it is the best way to learn. If we keep our problems to

ourselves, we restrict ourselves to only the solutions that we think of. We rob ourselves of learning from other people's suggestions and experiences. If a team doesn't share its problems and learning with other teams, we miss out on an opportunity to educate the entire organization.

Most companies have many IT teams. Imagine a company with ten IT teams. If they do not share their issues and learn from one another, one could imagine ten launches in a row where each team has the same problem. External customers do not realize there are many different teams. They simply see a big company that can't seem to get anything right since it keeps repeating its mistakes.

In a traditional model, our problems define our weaknesses. A better model is one where problems and solutions are discussed openly. Problems are a means by which we learn something valuable. Each problem is an opportunity to share this new wisdom as widely as possible. Instead of defining our weaknesses, the discovery and communication of a problem makes the entire organization smarter.

19.2.6 Step 6: Repeat

There's no such thing as a final launch. There will be new releases with new features, bug fixes, and so on. Each requires another launch.

Each launch should have its own ready list and use a similar process, though future launches tend to be smaller and do not require as much bureaucracy.

As discussed in [Section 16.6.4](#), software requires upgrades. Even if you do not add features or find bugs to be fixed, the underlying libraries and frameworks will need to be upgraded as security holes are discovered in them. They start out fresh but they go bad. Joshua Corman ([2015](#)) explained it this way: "Software ages like milk."

19.3 Launch Readiness Review

A launch readiness review (LRR) is a process created to assure successful launches by comparing the state of the service against launch readiness criteria (LRC).

Google's Production Readiness Reviews

The term LRR comes from Google SRE culture, which is described in the book *Site Reliability Engineering: How Google Runs Production Systems* ([Beyer, Jones, Petoff & Murphy 2016](#)).

19.3.1 Launch Readiness Criteria

The LRC include basic tasks such as verifying that stakeholders have approved the launch, that there is a disaster recovery plan in place, that data backups are working and have been tested, and that various security audits have been completed. The LRC also include items that come from the lessons learned during past launches.

A major search engine once launched a new analytics tool. It was unexpectedly popular on its first day and attracted ten times as many users as had been planned for. It quickly became overloaded. There was no plan in place for how to handle this situation, so the service was embarrassingly slow as the operations team scrambled to make adjustments and scale the system. The launch became a big black-eye for the company.

The organization already had launch criteria that product management must produce usage estimates and load testing must be performed to verify that the system could handle such a load. However, as a result of the unexpected popularity of this one service, the organization added a new criterion: The team must have a plan for what to do in the unlikely event that the service turns out to be ten times more popular than expected. Whether this means that the service will be disconnected (and who can approve this decision), or that the service will be restricted to a small subset of users, these decisions must be made ahead of time so that there is prior agreement about what to do, and operations can test these plans ahead of time.

Note that the launch criteria list is not the same as the launch procedure. The latter may itself be a checklist of technical steps required to actually launch, such as DNS updates, firewall updates, load-balancer changes, and so on.

When all the criteria are met, we are ready to launch. Management should be strict in applying the criteria. If exceptions are permitted, the business risk should be quantified so that managers understand the implications of what they are agreeing to.

19.3.2 Sample Launch Criteria

Here are some sample items that might appear on launch review criteria lists:

- **Service is monitored.** Service is monitored both for uptime and performance. Internal systems are measured for resource utilization, access counts, and error counts.
- **Monitoring alerts are configured.** Alerts trigger for SLA violations, and

for situations that would lead to SLA violations.

- **Backups and restores are tested and working.** Data storage has a backup mechanism, and restores have been tested.
- **Standard authentication, authorization, and access control have been tested and are working.** The service uses the company's standard AAA service, and access has been successfully tested.
- **SLA is defined.** The service SLA is defined in writing.
- **Service request model is in place.** Customers will be able to submit service requests using the standard tools, and these requests will be properly handled.
- **User documentation is complete.** The documentation required by users is complete and verified.
- **Operational documentation is complete.** The documentation required by operations to run the service is complete, and there exists a written procedure for what to do for each exceptional situation (alerts).
- **User training is complete.** Users have received training. If training cannot begin until the system is launched, how will people be trained before their need for the service is critical?
- **Load testing is complete.** The system has been load-tested using simulations of the expected and optimistic traffic levels.
- **Ten-times response plan is developed and tested.** If on the first day the service receives ten times as many users as expected, which steps will be taken? This plan must be written and tested.

19.3.3 Organizational Learning

Sometimes problems identified during a retrospective become candidates to add to the launch readiness checklist. The LRC is how we capture learning in an organizationally beneficial way.

Without an LRC process each team will probably make similar mistakes with its launches. They're all in the same environment, with the same trips and traps, and all probably equally naive when doing their first launch; or perhaps their last launch is such a distant memory that this launch may as well be the team's first launch.

LRC are a way to improve the organization's collective knowledge. They communicate learning from one group to another so that people do not repeat one another's mistakes.

19.3.4 LRC Maintenance

As the LRC list grows, it can become a bureaucratic burden. We can prevent that problem by being cautious about adding new items, and by removing obsolete items.

Each item on the list incurs a cost to the company. Whether the cost is just answering a question on the LRC form or the actual mitigation, it is work that someone must do for each launch.

As the list gets longer, fear of the list grows. If that happens, the LRC become an excuse not to launch, and the company becomes calcified, unable to adapt and change and meet new business needs. Alternatively, people may find ways to work around the LRC, which means unofficial launches occur that don't benefit from the learning of others. This change will not happen overnight. It is a long, slow process. One day we wake up and find that launches are going wrong because the LRCs are no longer being used.

The list should grow slowly through experience. Growing the list in this way keeps the criteria based in reality, and prevents well-intentioned individuals from adding hypothetical situations that are unlikely to happen. It avoids the temptation to add every seemingly good idea to the list.

Items should be updated and rewritten as needed. Obsolete items should be removed. This is difficult because there is personal risk in shrinking the list but no liability in adding to it. People fear that they will be personally blamed if they suggest removing an item that later enables a problem or outage. The bigger risk is that the list will become so big and scary that launches become less frequent. The benefits of the small batches principle will disappear: Each launch will include a larger batch of changes, people will get out of practice in doing launches, and so on. This will discourage launches, which further delays or blocks new features. The result is that the company becomes immobilized. Therefore it is everyone's duty, and management's responsibility, to assure discipline in list maintenance.

The best way to remove something from the list is to make it obsolete because it has become automatic. For example, suppose one LRC item is to explain how the service survives single-host failures. Each launch designs and implements a failover mechanism that then is documented, tested, reviewed, and evaluated as part of the LRC. If the company creates a centralized load balancing service that can be used on a subscription basis ("load balancer as a service"), the labor involved in satisfying this LRC item becomes significantly easier. The criteria can be satisfied by simply showing that the system uses the platform's default

load balancer.

19.4 Launch Calendar

Large organizations that engage in many simultaneous launches require some kind of coordination of these activities. Usually this takes the form of a launch calendar. The launch calendar may be a simple chronological list of future launches on a wiki or something more complex, such as a groupware calendar created via Microsoft Exchange or Google Calendar.

As organizations grow larger, it is more likely that some people will be out of the loop, which can lead to errors. The calendar creates awareness of future launches. In a small organization it is easy for everyone to know about future launches since they are probably involved. Having a shared launch calendar prevents conflicts. It also makes it easier to schedule isolated launches. Some launches are better done in isolation for technical reasons, or simply to avoid confusion.

The calendar also helps teams to plan around one another's launches. For example, the launch of something externally visible may require marketing announcements and other work that needs to be done ahead of time. A calendar enables this activity to be planned and coordinated.

Some items on the calendar will be listed on their final planned date; other launch dates are approximate or aspirational. Items in the far future are more likely to be estimates, while items happening soon should be more accurate. It is useful to indicate items whose dates are still changing with a tag, icon, or different color.

Each team should be held responsible for keeping its entries on the launch calendar up-to-date.

19.5 Common Launch Problems

This section describes the most common launch problems. An observant reader may notice that most of these problems can be avoided by using the DevOps methodology described earlier. DevOps culture also encourages the early involvement of operations, which would prevent the other issues.

19.5.1 Processes Fail in Production

Deployment processes that worked in other stages may fail in production. Often the deployment process is different in each of the dev, test, beta, and production environments. This often happens when a different team is responsible for each

stage, and they don't communicate or share code. The members of each team develop their own installation process. Often the production environment is completely different, and the production team finds itself developing a procedure from scratch, without the benefit of experience from deploying to the other stages. Or, perhaps the beta and production environments are similar enough that one process should work for both. However, the production environment has changed in the many months since the last time code was pushed to it and the process breaks. Either way, the lesson here is that the best launch experience comes from having a unified deployment mechanism across all stages. It should be developed as part of the code, not as something left to do the day the service is pushed to production. One of the goals of containers such as Docker is to provide a single delivery format that works in all stages.

19.5.2 Unexpected Access Methods

The way users access the service is different in all stages prior to production. Many service launches fail because tests were performed using one access method but customers actually access it another way. In one case, an SSL VPN was going to be how 80 percent of the users would access the service but the developers and testers used the service only without the VPN. In another case, the system was tested only by local users with fast networks; no one considered that actual users would be accessing the system over high-latency links, which would cause certain protocols to time out. The lesson here is to make sure the testing is as similar to the access method that real users will use.

19.5.3 Production Resources Unavailable

Assumptions are typically made about resources available in production. Suppose shortly before launch the developers announce that the new service will require a large amount of disk space, CPU, or bandwidth resources. Developers simply assumed that these resources would be available without making their requirements known ahead of time. The real-world logistics of installing additional servers or acquiring more bandwidth are complex and often take weeks to months of pre-work. Developers are often shocked to learn that there isn't a petabyte of spare disk space they can use, or that adding ISP capacity can easily take 12 or more weeks. The lesson here is to involve operations with resource and capacity planning early in development, not at the end.

19.5.4 New Technology Failures

The first time a new technology is used, the launch takes longer. The first time

you use a new technology, it will take considerably longer to get it working in production. This is true for a new language, framework, database, or other service. For example, if your organization is a C# shop, the first time you try to use NodeJS in production you will discover dozens of unexpected issues, each requiring a week of experimentation, design, and testing. If you have great operational expertise in one brand of database, and a developer decides that some new data should be in a different database system, the developer may be surprised to discover that adding this database requires developing new expertise and procedures. Sadly, the true believers in the new technology can't possibly imagine that any of these tasks will take any time because the new technology solves all problems (or so they think). As a result, the use of these new technologies may not even be communicated until the last minute and the launch will be delayed. The lesson here is that new technologies require weeks and possibly months of operational work before they can be used in production. This factor should be weighed as part of the decision to adopt the new technology. Descriptions of such failures have been documented in articles such as "Why MongoDB Never Worked Out at Etsy" ([McKinley 2012](#)) and "Providence: Failure Is Always an Option" ([Punyon 2015](#)).

19.5.5 Lack of User Training

No training for users leads to disaster. For most new services, users require training or at least simple screenshots that walk them through the steps to access the system. Many times the excuse to delay training is the following Catch-22: Training requires that the system be launched, but the launch can't be done until the training is complete. The organization, therefore, decides to launch without training and ends up in a sticky situation where people can't go back to the old system (it has been disabled) and don't know how to use the new system. One solution is to build a staging area used exclusively for training purposes. This provides a platform for training without disrupting the legacy production service. The lesson here is that the training process must be planned for just like any other feature.

19.5.6 No Backups

Suppose that shortly before launch, operations points out that the new system has no disaster recovery plan. The system is constructed in a way that makes doing backups and restores very difficult. More importantly, it can take weeks, perhaps months, to test the restore process until it is sufficiently reliable. The lesson here is that disaster recovery features must be built into a service and can

not be designed as an afterthought.

19.6 Summary

Launching a service requires a lot of coordination. We must be prepared for unexpected surprises, especially the first time we use a new technology or are in a new environment.

There are six steps to launching a new service: We define the end goal in terms of a ready list, or tasks that must be completed. We work this list. We launch into a beta test environment, and then we launch into production. We capture lessons learned, and we repeat the process.

The ready list should be stored where everyone has access to it to improve transparency and communication. Consider using a bug or request-tracking system, a Kanban board, or an online shared spreadsheet. Such coordination is important even for a single-person team.

As we launch more services, we start to accumulate best practices and guidelines that will help future launches be successful. We can share these lessons learned by assembling them into a launch readiness checklist used for all launches.

Some common launch problems to be aware of include differences between test environments and production environments, operational difficulties the first time a new technology is used in production, and lack of resource and capacity planning.

The more launches you are involved in, the better you will get at launching new services.

Exercises

1. Describe the six-step launch process.
2. Steps 5 and 6 of the launch process are not part of the actual launch. Why are they important?
3. What is a ready list?
4. In a previous project you worked on, how was the ready list maintained? What were the pros and cons of maintaining it this way? What would you have done differently?
5. Why does a one-person team need a ready list?
6. If a system is sufficiently tested, why does it need a beta stage?
7. What is a retrospective document? What are its primary components?

8. Which launch readiness criteria (LRC) would you apply to launching a new email server?
9. [Section 19.5](#) describes some common launch problems. Which have you experienced? What could have been done to prevent them?
10. Project: Five chapters in this book are about managing complex processes that bring something new to the organization: launching a new service (this chapter); launching via rapid iterations ([Chapter 20](#)); moving users from an old service to a new service ([Chapter 21](#)); upgrading a single, complex machine ([Chapter 33](#)); and launching an organizational change ([Chapter 37](#)). Make a chart with a row for each major recommendation and a column for each chapter. Mark an X in the box to indicate the recommendation was made in this chapter. Which recommendations are the most common? Which are the most unique to particular situations?

Chapter 20. Service Launch: DevOps

Now that we understand the fundamentals of launching a new service, how can we optimize the process?

The DevOps movement has a number of principles that we can adopt to improve any operational process. DevOps is a way to improve IT that involves developers and operations working in an integrated fashion. Yet, DevOps practices apply to any operational situation. You don't need developers to benefit from the DevOps principles!

This chapter focuses on two ways of replacing a risky big launch with many smaller launches. By doing so, we enable the benefits described in [Chapter 2](#), “[The Small Batches Principle](#).” The first way is to use continuous integration and deployment techniques, which create the ability to rapidly and confidently launch new releases into production. The second way is to first develop a minimum viable product (MVP) and then enhance it with many small improvements over time.

DevOps or Rapid Release

The IT department of one *Fortune* 100 company started a project to adopt DevOps practices but found huge resistance at every level. People had many preconceived notions of what DevOps was. The myths and misconceptions held by employees confused the issue and derailed the entire project.

The team took a step back to analyze their situation. They realized what was important to them wasn't DevOps per se, but rather that the company was being killed by the fact that it was so damn difficult to deploy new software releases. If software was never improved, the company couldn't improve. It was being beaten by its competitors, which were able to rapidly improve everything from their web site's shopping cart, to how price tags were printed, to how products were shipping to their stores, to making sure their snack shops didn't run out of hotdogs. All of these things were driven by software.

Members of the IT department went back and rewrote all their presentations. They removed the phrase “DevOps” and instead used the phrase “rapid release.” They extolled the virtues of being able to rapidly and confidently release new software upgrades. They spotlighted one

project that had successfully adopted the new testing and deployment processes that meant a new feature could go from concept to the field in a week instead of a year.

Other teams accepted the concept now that the focus was on a benefit they could appreciate. Soon this Rapid Release Initiative was transforming the company.

20.1 Continuous Integration and Deployment

The six-phase launch process discussed in [Chapter 19](#), “[Service Launch: Fundamentals](#),” included various staging areas that a software release flows through: development, testing, beta, and production. Each stage reduces the risk of surprises and failures at the production launch. We can think of those stages as practice for the actual launch. If practice makes perfect, shouldn’t we practice a lot?

While the staccato movement from one staging area to another is commonly used throughout the industry, it is non-optimal and possibly toxic. It is very labor intensive. Each stage has a team of people associated with it who must be available or the process halts. It keeps each team in a silo, optimizing for their own needs instead of the greater good. It encourages people to game the system, because the focus becomes getting work past the gate rather than creating good output. It also encourages large batches. If QA receives a new release every quarter, the number of changes since the last release will be complex enough that this team must test everything from scratch. As an optimization it is tempting to send a new release to QA less frequently, but this simply creates a bigger testing burden. If a bug is found, the developers need to isolate it to one of the changes, which is now an even larger list.

A more modern technique is called continuous integration and continuous delivery (CI/CD). This practice automates software building, packaging, and testing so that it can be done frequently. In fact, the cycle is repeated for each change to the source code. Thus, instead of disconnected tests done periodically, it is a continuous process. Each release is automatically installed in a stage and put through a testing process. If all the tests pass, the release is installed in the beta staging area, where more tests run automatically. In some environments these tests gate automatic deployment to the production environment.

20.1.1 Test Ordering

The order in which tests run is important. The most expensive tests should be

placed at the end of the testing pipeline. A less expensive test failing early prevents us from having wasted resources on a more expensive test. For example, if there are manual tests, do them at the end. These tests need to be done only if all the automated, and therefore less expensive, tests are successful. Imagine spending a morning running manual tests and then running the automated tests, only to find one failed. You just wasted your morning.

If there are manual tests, you may choose to run them only occasionally. In fact, some sites run the automated tests in a loop, always pounding on the newest release. The manual tests, if there are any, are run only on release candidates.

Sites with fully automated testing can achieve truly continuous integration and delivery. That is, each change triggers a release that is built and tested. This way, if there is a failure, team members know exactly which change introduced the problem—the most recent change! Batching up tests in a daily, weekly, or monthly release means a failure could be present in any of a large batch of changes and is more time-consuming to debug.

20.1.2 Launch Categorizations

We classify the way organizations perform launches into four general categories: chaotic, periodic, continuous, and push-on-green.

- **Chaotic:** Releases are brought into testing at random times. The building, deployment, and testing of a release are not automated and may be ad hoc. Therefore testing may be done differently depending on who does it.
- **Periodic:** Releases are done on a schedule such as daily, weekly, or monthly. Usually the entire build and deployment processes are fully automated. Testing is generally automated or mostly automated with a few exceptions that are done manually if all automated tests succeed.
- **Continuous:** Each change to the source material triggers an automatic creation of the next release, build deployment, and testing. Because these steps happen so frequently, they must be automated. The actual release to the production stage requires human approval.
- **Push-on-green:** Here the decision to launch to production is automated. In other words, the push to production is approved when the status of all tests is green (success). This is extremely rare and requires extraordinary measures.

The latter categories permit faster iterations and achieve better confidence through more complete automated testing. They benefit from the small batches principle. Features get to production sooner, and the improved confidence they

instill makes it easier for the organization to experiment and try new things. The ability to do rapid releases is a major goal of the DevOps philosophy. Volume 2 of this book series has more examples of this kind of methodology.

It is difficult for a chaotic project to jump all the way to push-on-green. Instead, one generally seeks to improve a project one level at a time. In fact, a reasonable goal is for all projects to aspire to periodic or continuous launches. The extra investment required to get to push-on-green should be reserved for only the most important projects. Making all projects push-on-green is a waste of resources; your time is better spent elsewhere. Push-on-green requires an extremely high level of confidence in your automated testing procedure that can be achieved only by involving everyone from product management to developers to operations.

The best way to achieve a specific level for all projects is to create a development platform that makes it easy. If projects start out using the platform, they skip the lower levels. Make right easy. If the lazy way to do something guides people to the best methodology, it becomes the default.

Etsy's development platform starts new projects at or near push-on-green. Every individual change triggers the full testing and deployment suite. As a result, Etsy launched 6,000 times in 2015. Such a large number of launches demonstrates impressively high confidence in the company's testing and deployment automation. This confidence enables team members to move faster and be bolder about launching new features.

Klein, Betser & Monroe ([2014b](#)) provide a good overview of the rigorous methodology required to make push-on-green possible at Google. The book *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation* by Humble & Farley ([2010](#)) is a deep dive into the topic.

Later in this chapter we'll discuss how these techniques can be applied to third-party software, which is commonly used in most enterprise environments.

Eliminating the Approval Committees at Google

In some organizations, there is an approval committee that signs off on a release before it is launched to production. The committee has access to the beta or UAT environment and blesses the release.

Such committees are an impediment to rapid iterations. It is difficult to iterate quickly if the approval committee meets monthly. Ideally, such committees should be eliminated.

Daniel V. Klein gave a talk at LISA 2014 describing how Google is able to launch when the testing dashboard shows all green (all tests passing) rather than having a human approval committee ([Klein, Betser & Monroe 2014a](#)).

It is not simply a matter of automating the checks done by the approval committee. Klein recommends a higher level of quality and testing at every step of the process, starting with the methods and behaviors of the developers, all the way through to how the releases are pushed into production.

20.2 Minimum Viable Product

The six-step launch process described in the previous chapter is very common—but also extremely risky. It puts all our eggs in one basket. If we aren't ready by the launch date, we are doomed.

A better method is to first launch early with the absolute minimum number of features, then add features through a series of subsequent launches. As the number of features grows, we can decide if it is time to expose the service to real users. If the deadline arrives, we can launch with what we have, even if the lower-priority features have not yet been added.

Let's return to our example of launching an automated OS installation system for our Windows workstations. We want the system to install the OS, join the domain, configure settings, install MS Office, remove certain applications, and even do some fancy network configuration such as adjusting NIC priorities if a machine has multiple NICs. Suppose we have two months to create this system.

One way to organize this project is to add all those features to the ready list and then start working on the items. Sadly, by the time the deadline arrives, we aren't quite done. Worst of all, we're 80 percent done on all items, which means we're zero percent able to launch.

A better way to organize our work is to first build a beta system with the absolute minimum features to be a viable OS installation system. This is the **minimum viable product** (MVP). We launch the MVP into the beta staging area. Then we add new features one at a time. Whenever we complete a feature, we launch the new release to the beta area. When the deadline arrives, we launch with the features that are working in beta.

Because we will be launching to beta so many times, it is important that the end-to-end system from building the pieces from the source files, to testing, to

deployment is fully automated. Otherwise, the labor involved will be a burden and will steal time away from feature development.

For example, suppose our MVP is the ability to PXE boot the machine and install the default Microsoft-supplied version of Windows. It won't be customized in any way. It won't be attached to our domain. Nevertheless, the part of the process that we have implemented will be fully automated, as will a minimum set of tests. For example, to test the end-to-end process, we'll use a tool that can act as a remote keyboard. It will start the PXE installation process, test that a user can log in and out, and so on. There may be some manual steps because of the realities of OS installation, but our goal is to eliminate human intervention as much as possible. Lastly, the process by which the beta test configuration files are promoted from the beta area to the production area is automated, too.

Now we have our minimum viable product. If we had to, we could launch it to production, though it would be fairly anemic. At least it is better than walking around the office with the installation DVD.

Now we add more features. For example, we modify it to automatically join the ActiveDirectory domain. The tests are updated to verify this feature works. When this step is complete, we use the script to promote this version from the beta to the production environment. We repeat this for each feature.

When the deadline arrives, we launch the most recent working version in beta to the production environment. We know it works because we've been testing it all along.

While we borrow some techniques from the six-step launch process, the MVP approach gives us a different paradigm for launching services:

- **Features drive the launch.** We launch when the features are ready, not when a deadline on a calendar says we must launch. In fact, we can launch early if we choose. If we carefully choose the order in which we add features, we could launch earlier for certain segments of our target population. Perhaps we might focus on desktop features first and launch for laptops later.
- **We are always ready to launch.** Instead of fearing that we won't be done on time, we are always ready to launch. What gets launched may not have every feature, but a system like this is never actually done. There is always another feature that could be added, configured, or improved. If the project is an ongoing concern with no hard definition of "done," why use a paradigm that assumes there is a hard-and-fast endpoint?

- **Launching is low risk.** Because we’ve launched to beta so many times, we get very good at it. The process is automated. We’ve seen plenty of edge cases and added code to detect them, work around errors, and so on. The process gets smoother the more we use it. There is no big, scary “launch day.”
- **The difference between beta and production stays small.** Since we are adding one or two features at a time, the delta between the beta and production versions stays small. When we do launch, we understand which new feature is being launched. This is less risky, easier to comprehend, and easier to explain than mega-launches with hundreds or thousands of new features being introduced at once. We can concisely communicate what is in each release, which makes our customers better informed and better able to understand our work.

Case Study: MVP Applied to Writing Books

This launch methodology works for many kinds of projects, not just IT services. In fact, it is how we developed the third edition of this book. We determined a minimum viable product and worked toward that goal using a ready list. We also had a prioritized list of chapters we wanted to add or rewrite. Once the minimum viable product was complete, we were always in a state where we could hand the manuscript to the publisher for editing, production, and printing.

We wouldn’t want to ship the book in that initial state, as it was largely unchanged from the second edition. However, we could if we were called to do so. From then on we worked on the new features we wanted to see in the new edition. The ones we completed by our publisher’s deadline got included. The ones we didn’t finish . . . well, you’ll never know if there were any, will you?

20.3 Rapid Release with Packaged Software

This section deals with applying the rapid release methods with externally produced software—in other words, doing DevOps with shrink-wrapped software.

Our rapid-release examples so far have been with home-grown software or systems that require frequent code updates from within our own organization. If a third-party vendor produces one release per year, building all the rapid release automation seems like a lot of work for very little benefit. However, if the

confidence gained from high-quality testing and deployment methods is good for some sources of software, it can benefit all software.

Also note that the industry trend is toward more frequent releases. Microsoft Office used to be updated every three years, with occasional bug fixes in between. Now it is frequently updated, including with security patches that are released on the first Tuesday of each month. Competition with web-based applications is driving packaged software to participate in this trend.

For all these reasons the rapid release techniques are becoming more relevant.

20.3.1 Testing Before Deployment

If we treat packaged software as we did in the rapid release method, each release would be put through a battery of tests before it is deployed.

Testing requires building an environment that replicates the existing system, then performing the upgrade, then conducting various tests. We may tear down and rebuild the environment to perform tests that simulate the variations found in the field.

This is all very much the same as the testing process for home-grown software. The difference is that often organizations do not perform such rigorous testing, or if they do it is not automated. People tend to make rationalizations: We don't have a lab environment to conduct the tests. Building a lab environment would be too expensive. We don't have the labor resources to build and rebuild the environment, or to conduct the tests. We deploy to live users and have an excellent backout plan if something goes wrong.

If we do have a testing process, it may be so labor intensive that we might skip occasional releases to reduce our labor cost. While this decision saves labor, it makes the upgrade more risky, as the jump from version n to $n + 2$ may not be as well tested by the provider. Many organizations do no testing at all and simply assume that the vendor's testing is sufficient. This is not only risky but also misses the fact that the vendor's environment cannot possibly include the infrastructure we have built around the vendor's product.

These excuses and rationalizations are widespread in our industry. Some are justified by the fact that a small shop may not have the budget to do such testing, or that the risk is small enough that a backout plan is sufficient, or that the return on investment (ROI) for automating an infrequent task makes it difficult to justify. But suppose that the company then grows and grows and grows, and we have developed bad habits. At what point in our growth should we change our ways?

As a professional system administrator, it is your responsibility to use resources wisely. On the one hand, it would be irresponsible to set up a fully automated test infrastructure for a inconsequential service that is updated rarely. On the other hand, it would be equally irresponsible to not set up a repeatable testing structure, automated or manual, for the company's most critical system. As professionals, we must hone our communication skills to make the case for such efforts in terms of risk and reward rather than pure cost.

A written checklist of tests performed manually is better than no tests. A few scripts that make it easier or less error-prone to execute some of those tests is even better. Full automation on the most basic tests, leaving a few manual tests at the end, is an improvement. It is natural for a testing regimen to start out ad hoc, solidify as a checklist, and evolve to be increasingly automated over time. We create an MVP of testing and grow it over time.

Case Study: Evolving Email Server Tests

In one case, the system being developed was an email server. The tests that we ran included sending email to test complex email routing rules that were set up. Email was sent inside-to-inside, inside-to-outside, outside-to-inside, to an internal user with an automatic vacation message reply, to mailing lists, and so on.

Originally these test messages were sent manually, followed by a mad dash to log into the various destinations to verify that the messages landed as expected.

Sending the messages was easy to automate. Testing that they reached their destinations was much more difficult to automate. Initially we simply made the manual task easier by using subject lines that made manual verification easier. Each test run was given a number, and one simply had to verify that this test number was seen (manually) in all the desired inboxes. This meant that once the inboxes were open, each in a separate window, the manual verification was easier.

Eventually the delivery verification was also automated.

20.3.2 Time to Deployment Metrics

A key metric to track is the amount of time between when an update is available and when it is in use in production. The longer this time span, the more we are vulnerable to security holes, the longer our customers are exposed to bugs, and

the longer the delay for the productivity enhancements that new features bring. If we installed every update immediately, however, we would be the first to discover problems.

We can seek a happy medium between those two extremes. For example, risk-averse organizations might delay the deployment of Microsoft Windows “patch Tuesday” updates by about a week on the assumption that bad patches will be recalled during that time as early adopters report bugs. The more organizations that do this, the fewer early adopters that will exist. An alternative method is to have a testing framework that lets us gain confidence in the updates by testing them ourselves.

Whether your goal is to have new updates deployed as soon as possible, or as soon as possible after a waiting period, recording a history of this gap between availability and use in production is useful so that we know if we are doing better over time.

Better Software Upgrades

Upgrading software in large complex systems can be difficult. The book *Ship It! A Practical Guide to Successful Software Projects* by Richardson, J. R. & Gwaltney, Jr., W. A. ([2005](#)) includes many upgrade techniques and software design patterns that can make future upgrades easier.

20.4 Cloning the Production Environment

One technique for upgrading a service is to take an ordinary upgrade process and engineer a way to do it frequently.

As will be discussed in [Chapter 33](#), “[Server Upgrades](#),” a common upgrade technique is to clone the hard drive of a system and upgrade the copy. This permits us to revert to the original system if the upgrade process is a failure.

Initially we do this process manually. First the machine is cloned. The clone may require some reconfiguration to be able to run alongside the production environment. We run the tests on the original clone to verify it works. Next we run the upgrade procedure. The tests are now repeated to verify the results are the same.

We’ve been in situations where the service was running on a virtual machine and cloning it was simple. It did, however, require downtime. Luckily the service could be taken down at specific times of the day. This limited our ability to perform the full test to once a day—at least until we realized we could clone the

clone. Now we could run the entire process repeatedly from the master clone. The master clone was updated periodically.

In another situation the service was on a physical machine. This made cloning more difficult. However, the tests could be run in a virtual environment. We used a physical-to-virtual (P2V) utility to clone the physical machine.

In both of these situations, the process was very manual but evolved to be more automated over time. The breakthrough was always the point when the clone–modify–startup process was automated and ran in a loop. This automated process left out installing the upgrade, running the tests, and verifying the test results. However, because we had the process running in a loop without manual intervention, we could add these other elements one by one and enhance the system over time. Getting the minimal loop running was the hard part. After that, a wider group of team members could get involved by adding tests and other improvements.

Legacy Systems

A legacy system is any system that cannot be upgraded, fixed, or improved because the only way to test such changes is to try them in production. In other words, it is any system with no continuous test and deployment infrastructure. The oldest mainframe application is not a legacy system if we can test changes to it and confidently launch new releases without fear. The hottest new technology is a legacy system if we are stuck with the existing version, unable to confidently upgrade it.

Organizations should seek to eliminate all legacy systems either by literal replacement or by retrofitting them with rapid release techniques.

20.5 Example: DNS/DHCP Infrastructure Software

In this example we examine a complex system that already has a long manual testing regimen and evolve it to be automated to gain confidence and agility.

The product is DNS/DHCP appliances with a central management station deployed in a large enterprise that is very nervous about any infrastructure changes. The service is global, with changes and upgrades being applied at the management station and automatically propagating to all the appliances around the world. It is complex, as it interacts with many different systems: appliance to appliance, appliance to ActiveDirectory (many versions), appliance to network infrastructure, appliance to CMDB, CMDB to appliance, and automation to

appliance, plus a wide variety of client operating systems that use DHCP and perform dynamic DNS updates.

The vendor produces new software releases frequently, but the organization deploys only one release per year.

20.5.1 The Problem

The policy of upgrading only once per year came about for many reasons:

- Customers are very cautious, perhaps paranoid, about network changes. DNS/DHCP is a fundamental part of the infrastructure. Nearly every service and function relies on DNS/DHCP. Any misstep affects many.
- The IT team has been burned by past upgrade problems in a variety of other areas that turned into highly visible outages. They really want to avoid repeating such problems.
- Testing is expensive. An isolated test lab is used to run a new release through a series of tests before a release can be deployed in the field. The tests are performed manually. If any problems are found, the vendor produces a patch and the entire testing process starts again. Thus, there is a disincentive to find a bug.
- The organization maintains a tightly controlled network. Each change goes through a Change Review Board (CRB) that has a complex and intimidating approval process, and that requires changes to critical infrastructure to be accepted by leaders across the entire company.
- The company is so large that scheduling a change to significant infrastructure requires booking a slot in the change calendar months in advance.

Sadly, the yearly update cycle creates many problems:

- Security problems are fixed with patches/hotfixes rather than software upgrades, resulting in the site running a version of code that is run at few, if any, other sites. The fear of patches is less than the fear of upgrades.
- Yearly tests are too infrequent to be automated. There is low ROI for automating something done so infrequently, and by the time the next deployment rolls around enough has changed that the automation is obsolete. Any additional tests add to the labor burden of implementing a new release. This creates a disincentive to do more tests in an environment—the opposite of what we want.
- The vendor publishes new releases every few months but most go

uninstalled. This means each actual upgrade that the site undertakes includes a large batch of changes. Big batches of changes are more difficult to test, and more risky to deploy.

- Yearly deployments mean the engineers doing the upgrade are less experienced. They haven't done the process for a year; newer members are usually doing the process for their first time.
- The Change Review Board has a certain amount of turnover each year. Yearly upgrades mean that each change requires re-educating the CRB about the upgrade process and risks from scratch. People are more nervous about a change the first time they see it. Each year half the committee is seeing changes for the first time. Being new, they try to review the entire proposal, which is a lot of work and difficult to do well. Meanwhile, the few who remember the process from the previous year could do a better job of focusing on what's new.

20.5.2 Desired End-State

The organization would be in a better place if it could benefit from new releases sooner. Incentives should encourage more testing and more frequent upgrades. The skills of both the CRB members and the engineers should stay fresh through practice.

The ultimate goal is to have high confidence in a release. The company should be able to confidently say, "Yes, this release is ready to be deployed and the deployment will go smoothly." All employees should know this because of the thorough testing process they're using.

20.5.3 First Milestone

While this organization would benefit from a fully automated release process, we begin with a few baby steps.

First we get our own house in order. The goal here is to create a test process that proves out a release and deploys it into a beta environment. This process stops one step short of deploying into production. Success in the beta environment will reduce the fears of the CRB and other employees. This will pave the way to getting more frequent upgrades approved.

A minimal test loop is created. This is a slow evolution. A script is written that builds one particular test configuration. Another script runs a set of tests. Eventually a continuous integration tool is used to automate these tests, which can now be run in a loop.

Once this loop is running, new scenarios and tests are added. A library of cloned hosts is developed for use in test cases.

Some tests are still done manually, but this improves over time. What we learn from automating the easy cases prepares us for the more difficult ones. Some tests must be done using physical hardware and require a manual step to kick them off. This improves over time: First we find we can leave the machines in a configured state and require manual intervention only to wipe and reload the machine, which doesn't need to be done every test cycle.

The test automation tests not only features but also the upgrade process itself. The tests are run on a fresh installation as well as an older installation that has been upgraded to the most recent release. This is especially important if past outages were caused by bad interactions of old and new releases.

All of this automation is hard work and takes many months. However, soon a new release can be put through a battery of tests and deployed into the beta area, with the exception of a few manual steps. Even with those manual steps, we are in a better situation than before.

When customers report bugs, each bug is not only fixed, but the customer is also shown that the automated test framework now tests that situation; thus, if the bug reappears, it won't make it into production. These are called regression tests. Also, if we openly document our regression tests, it builds confidence with our customers, who can see that their needs are being addressed because those needs are codified in a way that will keep the bug fixed. It creates positive pressure to automate other tests: Management will be inclined to expect that all testing be automated.

Soon all testing and deployment processes are automated. The beta environment is upgraded with every new release. The engineers dream of a day when customers will participate in this process with enthusiasm by not simply submitting a bug report, but also offering a VM snapshot to be part of the regression test.

20.5.4 Second Milestone

The next milestone is to convince the CRB and other stakeholders to permit more frequent updates outside of the beta area. A number of strategies are employed.

First the team enlists allies. They partner with the security group, whose members appreciate the value of security patches being deployed swiftly. The team makes a good impression by demoing the new highly automated testing

system, showing the higher degree of confidence that it brings. Then key members of the CRB are recruited one by one to get them on board. They are impressed by the idea that more frequent deployments will reduce the risk associated with the change from high to medium.

Another way to develop confidence in the new methods is to use them in low-risk situations or places where the burdensome approval process is not required. The company is growing and new networks are added all the time. New networks, before live users are on them, do not require the same CRB approval process. Problems will not be visible to live users.

Confidence in the deployment process improves as it is exercised many times. Soon it is the allies who are requesting the process change. The team leverages these allies and gets permission to begin more frequent deployments. At first this is a pilot project, but soon it is approved and becomes the standard operating procedure.

In the end new releases are reaching users faster and with better results than before.

20.6 Launch with Data Migration

Often a launch involves migrating data from the old system to the new system. This may be a traditional SQL database, data files, or application-specific data. If you are launching a new email service, for example, there may be millions of email messages to migrate.

Usually we can't migrate data while the system is running live. The database has to be frozen, converted, and then used. If you convert the database on Monday and do the upgrade on Wednesday, you will miss any data entered on Tuesday. Such migrations are full of surprises. Here are some anonymized but real examples:

- A company was migrating to new, faster hardware. After waiting 3 months to find a time that everyone could agree to do the migration, the conversion began and users were locked out of the system. Fifteen minutes later, the team discovered they didn't have a required cable to connect the new storage system and wouldn't be able to purchase one in the 12-hour maintenance window. It took another 3 months to schedule more downtime. The company lost customers as a result of the delay.
- When a company migrated a database, it broke because the new system rejected records of any users with a space in their first name. Manually massaging these records was taking hours and was error-prone. The

conversion was rescheduled.

- A Girl Scouts council's database was offline for two weeks due to a migration. Some pieces were offline even longer. No one could register for events, leaders couldn't get rosters, and the entire region's holiday season was a disaster.
- A university planned a weekend-long outage of its email system to migrate it to a different product. It had been estimated that each user would be migrated in minutes, but once the migration began each user was taking nearly an hour. The incorrect time estimate was based on a simple file copy benchmark. The actual migration used the IMAP protocol to read and write one message at a time, which was significantly slower. The upgrade plan was cancelled. The damage to the IT department's reputation made it nearly impossible for them to get large projects approved for years.

In summary, most of these problems were related to insufficient resources, processes taking longer than expected, bugs, and general unpreparedness. For any system large enough to be useful, there will always be surprises with data migration. Therefore it is irrational to make plans that assume there won't be trouble.

One way to fix such problems is to migrate individual users one at a time. However, this requires that the old and new systems can run in parallel. This approach is recommended for the email migration example but won't work for the other cases described previously. That said, it does have the advantage that it permits a slow migration. Beta users can test the system first, and the most risk-averse users can be migrated last.

Email Migration Hacks

Another way to speed up an email migration is to not migrate the messages. Here are some alternatives:

- Permit users to log into the old system if they need to access their old messages. Keep the old system running in a read-only state for a few months.
- For exceptional users who do require migration, charge them a small fee to do the migration. Only a few will ask.
- Create a control panel that lets users activate the migration process themselves.
- Create a control panel that allows users to select whether new messages

will go to the old or new system (but not both). At a certain cut-off date, point everyone to the new service. This also gives you a way to monitor adoption rates.

Do practice runs of the conversion ahead of time. Set up a new system and practice the migration there. Delete the result and repeat the process until it can be done without fail. Now you can do the actual launch by freezing the existing system, doing the conversion, and then giving people access to the new system. You'll not only be confident that the process works, but also have an accurate time estimate for how long it will take to run.

An optimization is to give people access to the practice system for the purpose of testing. Of course, they have to understand that any changes they make will be lost, but it gives them the ability to try their processes and make sure everything works. This is called user acceptance testing (UAT).

Since you are going to be doing the conversion many times, you'll want to automate it. As in the second example, there may be database records that need to be fixed. Fixing them manually the first time is reasonable, as you are debugging the process. However, such manipulations should be automated by the next practice run. Done right, soon you will be able to do the entire copy–convert–launch by typing one command.

If the process is automated, you can run it a lot. You can, for example, run it daily, fixing problems each day as they arise. People can be given access to the practice system and report bugs. Eventually things will reach a steady state. The next day's conversion can be the official one. You've reduced the risk of failure, as the risk that a new surprise will develop in the last 24 hours is low.

We teach our children that practice makes perfect. Basically that's what we're doing here. We practice every day, learning as we go, until we are ready for showtime.

20.7 Controlling Self-Updating Software

Many software packages automatically update themselves. Examples include web browsers such as Mozilla's Firefox and Google Chrome, applications such as Google Earth, most smartphone apps, and many others.

Self-updating software is a blessing, as the upgrades happen automatically without requiring any IT effort. However, it can also be a curse if an upgrade breaks. Suddenly employees can't use the home-grown corporate payroll application, for example, because the broken HTML it generates doesn't display

on the newest Chrome release.

Because of this, IT departments may choose to tightly control when applications are automatically updated. The ability to control updates is usually considered an enterprise feature. It enables companies to test new releases before they are distributed to the entire company.

At one site, the OS software could be reconfigured to retrieve updates from a custom URL instead of the default URL that pointed to the vendor's HTTPS server. The vendor provided instructions for mirroring the software repository locally.

The company maintained three mirrors of the repository: test, early, and standard. Once a week the vendor's repository was mirrored to the test repository. A lab of test machines were configured to point to this repository. The test machines ran a series of automated tests. If the tests succeeded, the test repository was copied to the early repository. This early repository was used by a random 1 percent of all machines in the company, plus by any user who requested it. Users could opt out of being selected for the early group.

If no bugs were reported by the early users, the early repository was copied to the standard repository. Within a few hours the entire company had the latest release. Actually, the way this was implemented was that the copy was automatically done a week later. If a bug was reported, someone would manually halt the copying process. The general case was automated, and the exception required manual intervention.

Over time, the system became more fully automated. As confidence in the system grew, the lag time was reduced.

An Honest Job Advertisement

Imagine if job advertisements were completely honest. Most companies advertising for IT workers would state that the job is mostly great except for twice a year when "Hell Month" arrives and everyone scrambles to deploy the new release of some major software system. This month is so full of stress, fear, and blame that it makes you hate your employer, your job, and your life. Oh, and by the way, the software releases are often late, so you can't predict which month will be Hell Month. As a result, you can't schedule any kind of vacation. Without time off to relax, stress builds and makes your life even worse.

Sadly, at many companies Hell Month is every month.

A company that adopts the DevOps principles is different. A rapid

release environment deploys upgrades to production weekly, daily, or more often. It is not a stressful event—it is just another day. There is no fear of an upcoming Hell Month.

Imagine if an auto manufacturer’s employees spent most of their time assembling cars, but when a car actually left the factory it was a fearful, stressful month of hell. It would be unacceptable to run a car company that way. It should be unacceptable to manage technology that way, too.

Adopting DevOps techniques is not just better for the company, it is better for you. Over time more and more companies will adopt these techniques not just because they are better for the bottom line, but because they will find it impossible to recruit and hire technical talent if they do not embrace this approach.

Who would want to work anywhere else?

20.8 Summary

In this chapter we discussed many ways to turn a big, risky launch process into many small iterations. With small iterations we benefit from the small batches principle introduced in [Chapter 2](#), “[The Small Batches Principle](#).”

Launching a new service is both risky and important. A service that is created but unlaunched isn’t very useful. A failed or mismanaged launch is an embarrassment that harms a team’s reputation, which robs it of momentum.

Launching home-grown software can be improved by automating the software delivery pipeline: the process of building, testing, and deploying software. Once it is automated, we can repeat the process at will.

How an organization launches software can be thought of as falling into one of four levels: chaotic (ad hoc), periodic (daily, weekly, monthly), continuous (after each change a release is deployed in a test environment), or push-on-green (continuous, to the production environment).

Another small batch technique is to launch a minimum viable product (MVP), followed by many launches, each with new features. This has the benefit of giving operations personnel time to develop their runbook and operational experience.

These techniques work with third-party software as well. A test environment is created to simulate the production environment. Tests are automated and run continuously. There is more confidence in the deployment to the live environment, so we can launch more frequently.

Similar techniques work when converting users to a new service. Rather than converting all users at once, we can convert a certain number of people per day, increasing our velocity as we develop confidence with the process.

Launching in small iterations makes the process more likely to succeed, less risky for the business, and less stressful for the people involved. Who would want to work in an environment that works any other way?

Exercises

1. Describe the rapid release strategy for launching a new service.
2. What is continuous integration and deployment?
3. Why is it important to perform the most expensive tests last?
4. What is the MVP methodology, and what are its benefits?
5. Pick a project in your environment and determine which of the four launch categorizations it falls into. Which steps would raise it to the next level?
6. Why is it unreasonable to try to achieve push-on-green for all projects?
7. Which would benefit a company most: taking a chaotic launch process and making it periodic, or taking a continuous launch process and making it push-on-green?
8. Pick a project you are involved with and describe how it could be organized using the rapid release/DevOps methodology.
9. Pick a project you are involved with and describe how it could be organized using the MVP methodology.
10. This chapter suggested permitting people to have access to an old email server to access their old messages. If you plan to keep the system running for a month, how long should you announce it will be available?

Chapter 21. Service Conversions

Sometimes, you need to convert your customer base from an existing service to a new replacement service. The existing system may not be able to scale or may have been declared to be at “end of life” by the vendor, requiring you to evaluate new systems. Or, your company may have merged with a company that uses different products, and both parts of the new company need to integrate their services with each other. Perhaps your company is spinning off a division into a new, separate company, and you need to replicate and split the services and networks so that each part is fully self-sufficient. Whatever the reason, converting customers from one service to another is a task that SAs often face.

As with many things in system and network administration, your goal should be for the conversion to go smoothly and be completely invisible to your customers. To achieve or even approach that goal, you need to plan the project very carefully. This chapter describes some of the areas to consider in that planning process.

As with many high-level system administration tasks, a successful conversion depends on having a solid infrastructure in place. Rolling out a change to the whole company can be a very visible project, particularly if there are problems. You can decrease the risk and visibility of problems by rolling out the change slowly, starting with the SAs and then the most suitable customers. With any change you make, be sure that you have a backout plan and can revert quickly and easily to the pre-conversion state, if necessary.

We have seen how an automated patching system can be used to roll out software updates ([Chapter 7](#), “[Workstation Software Life Cycle](#)”), and we’ve seen how to build a service, including some of the ways to make it easier to upgrade and maintain ([Chapter 20](#), “[Service Launch: DevOps](#)”). These techniques can be instrumental parts of your roll-out plan.

Communication plays a key role in performing a successful conversion. It is never wise to change something without making sure that your customers know what is happening and have told you of their concerns and timing constraints.

This chapter touches on each of those areas, along with two general approaches to conversions. You need to plan every step of a conversion well in advance to pull it off successfully with minimum impact on your customers. This chapter should shape your thinking in that planning process.

An Invisible Change

When AT&T split off Lucent Technologies, the Bell Labs research division was split in two. The SAs who looked after that division had to split the Bell Labs network so that the people who were to be part of Lucent would not be able to access any AT&T services, and vice versa. Some time after the split had been completed, one of the researchers asked when it was going to happen. He was very surprised when he was told that it had been completed already, because he had not noticed that anything had changed. The SAs considered this the best indicator of their success.

21.1 Minimizing Intrusiveness

When planning the conversion roll-out, pay close attention to how the conversion impacts the customer. Aim for the conversion process to disrupt the customer as little as possible. Try to make it seamless.

Does the conversion require a service interruption? If so, how can you minimize the time that the service is unavailable? When is the best time to schedule the interruption in service so that it has the least impact?

Does the conversion require changes on each customer's workstation? If so, how many changes will there be, how long will they take to implement, and can you organize the conversion so that the customer is disturbed only once?

Does the conversion require that the customers change their work methods in any way? For example, will they have to use new client software? Can you avoid changing the client software? If not, do the customers need training? Sometimes training is a larger project than the conversion itself. Are the customers comfortable with the new software? Are their SAs and the helpdesk familiar enough with the new and the old software that they can help with any questions the customers might have? Have the helpdesk scripts ([Section 27.7](#)) been updated?

Look for ways to perform the change without interrupting the service, without visiting each customer, and without changing the workflow or user interface. Make sure that the support organization is ready to provide full support for the new product or service before you roll it out. Remember, your goal is for the conversion to be so smooth that your customers may not even realize that it has happened. If you can't minimize intrusiveness, at least you can make the intrusion fast and well organized.

The Rioting Mob Technique

When AT&T was splitting into AT&T, Lucent, and NCR, Tom's SA team was responsible for splitting the Bell Labs networks in Holmdel, New Jersey. At one point, every host needed to be visited to perform several changes, including changing its IP address. A schedule was announced that listed which hallways of offices would be converted on which day. Mondays and Wednesdays were used for conversions; Tuesdays and Thursdays were for fixing problems that arose; Fridays were unscheduled, in the hope that the changes wouldn't cause any problems that would make the SAs lose sleep on the weekends.

On conversion days, the team used what they called the "rioting mob technique." At 9 AM, the SAs would stand at one end of the hallway. They'd psych themselves up, often by chanting, and move down the hallways in pairs. Two pairs were PC technicians, and two pairs were Unix technicians—one set for the left side of the hallway and another for the right side. As the technicians went from office to office, they shoved out the inhabitants and went machine to machine, making the needed changes. Sometimes machines were particularly difficult or had problems. Rather than trying to fix the issue themselves, the technicians called on a senior team member to solve the problem and they moved on to the next machine. Meanwhile, a final pair of people stayed at command central, where SAs could phone in requests for IP addresses and provide updates to the host, inventory, and other databases.

The next day was spent cleaning up anything that had broken and then discussing the issues to refine the process. A brainstorming session revealed what had gone well and what needed improvement. The technicians decided that it would be better to make one pass through the hallway, calling in requests for IP addresses, giving customers a chance to log out, and identifying nonstandard machines for the senior SAs to focus on. On the second pass through the hallway, everyone had the IP addresses they needed, and things went more smoothly. Soon, they could do two hallways in the morning and all the cleanup in the afternoon.

The brainstorming session between each conversion day was critical. What the technicians learned in the first session inspired radical changes in the process. Eventually, the brainstorming sessions were not gathering any new information; the breather days became planning sessions for the next day. Many times, a conversion day went smoothly and was completed by lunchtime, and the problems resolved by the afternoon. The

breather day became a normal workday.

Consolidating all of an individual customer's disruption to a single day was a big success. Customers were expecting some kind of outage but would have found it unacceptable if the outage had been prolonged or split up over many instances.

The complete story is documented in Limoncelli, Reingold, Narayan & Loura ([1997](#)).

21.2 Layers Versus Pillars

A conversion project, like any project, is divided into discrete tasks, some of which have to be performed for every customer. For example, with a conversion to new calendar software, the new client software must be rolled out to all the desktops, accounts need to be created on the server, and existing schedules must be converted to the new system. As part of the project planning for the conversion, you need to decide whether to perform these tasks in layers or in pillars.

- With the *layers* approach, you perform one task for all the customers before moving on to the next task and doing that for all of the customers.
- With the *pillars* approach, you perform all the required tasks for each customer at once, before moving on to the next customer.

Think of baking a large cake for a dozen people versus baking 12 cupcakes, one at a time. You'd want to bake one big cake (the layers approach). But suppose instead you were making omelets. People would want different things in their omelets—it wouldn't make sense to make just one big one (the pillars approach).

Tasks that are not intrusive to the customer, such as creating the accounts in the calendar server, can be safely performed in layers. However, tasks that are intrusive for a customer should be performed in pillars. Examples of more intrusive tasks are installing client software that requires a reboot or getting the customer to initialize accounts on a new system.

With the pillars approach, you need to schedule one appointment with the customer, rather than several. By performing all the tasks at once, you disturb each customer only once. Even if it is for a slightly longer time, a single intrusion is typically less disruptive to your customer's work than many small intrusions.

A hybrid approach achieves the best of both worlds: Group all the customer-visible interruptions into as few periods as possible. Make all other changes

silently.

Case Study: Pillars Versus Layers at Bell Labs

When AT&T split off Lucent Technologies and Bell Labs was divided in two, many changes needed to be made to each desktop to convert it from a Bell Labs machine to either a Lucent Bell Labs machine or an AT&T Labs machine. Very early on, the SA team responsible for implementing the split realized that a pillars approach would be used for most changes, but that sometimes the layers approach would be best. For example, the layers approach was used when building a new web proxy. The new web proxies were constructed and tested, and then customers were switched to their new proxies. However, more than 30 changes had to be made to every Unix desktop, and it was determined that they should all be made in one visit, with one reboot, to minimize the disruption to the customer.

There was great risk in that approach. What if the last desktop was converted and then the SAs realized that one of those changes was made incorrectly on every machine? To reduce this risk, sample machines with the new configuration were placed in public areas, and customers were invited to try them out. This way, the SAs were able to find and fix many problems before the big changes were implemented on each customer workstation. This approach also helped the customers become comfortable with the changes.

Some customers were particularly fearful because they lacked confidence in the SA team. These customers were gently escorted to the public machines and asked to log in, and problems were debugged in real time. This calmed customers' fears and increased their confidence.

E-commerce sites, while looking monolithic from the outside, can also think about their conversions in terms of layers and pillars. A small change or even a new software release can be rolled out in pillars, one host at a time, if the change interoperates with the older systems. Changes that are easy to do in batches, such as imports of customer data, can be implemented in layers. This is especially true of nondestructive changes, such as copying data to new servers.

21.3 Vendor Support

When doing large conversions, make sure that you have vendor support. Contact the vendor to find out if there are any pitfalls. This can prevent major problems.

If you have a good relationship with a vendor, that vendor should be willing to be involved in the planning process, sometimes even lending personnel. If not, the vendor may be willing to make sure that its technical support hotline is properly staffed on your conversion day or that someone particularly knowledgeable about your environment is available.

Don't be afraid to reveal your plans to a vendor. There is rarely a reason to keep such plans secret. Don't be afraid to ask the vendor to suggest which days of the week are best for receiving support. It can't hurt to ask the vendor to assign a particular person from its support desk to review the plans as they are being made so that the vendor will be better prepared if you do call during the upgrade with a problem. Good vendors would rather review your plans early than discover that a customer has a problem halfway through an upgrade that involves unsupported practices.

21.4 Communication

Although the guiding principle for a conversion is that it be invisible to the customer, you still have to communicate the conversion plan to your customers. Indeed, communicating a conversion far in advance is critical.

By communicating with the customers about the conversion, you will find people who use the service in ways you did not know about. You will need to support them and their use of the new system. Any customers who use the system extensively should be involved early in the project to make sure that their needs will be met. You should find out about any important deadlines that your customers have or any other times when the system needs to be absolutely stable.

Customers need to know what is taking place and how the change is going to affect them. They need to be able to ask questions about how they will perform their tasks in the new system and they need to have all their concerns addressed. Customers need to know in advance whether the conversion will require service outages, changes to their machines, or visits to their offices.

Even if you expect the conversion to go seamlessly, with no interruption or visible change for the customers, they still need to know that it is happening. Use the information you've gained to schedule it for minimum impact, just in case something goes wrong. If everything goes smoothly, the pre-and post-conversion communication helps to improve the customers' perception and visibility of the SA team. If you don't communicate the change, they will just assume that you are not doing anything. If you communicate the change and something fails, at

least the customers know what, when, and why you were doing something, and they will be more tolerant of the resulting disruption. See [Chapter 49](#), “[Perception and Visibility](#),” for more about managing the rest of the company’s view of the SAs.

Have the high-level goals for the conversion planned and written out in advance; it is common for customers to try to add new functionality or new services as requirements during an upgrade planning process. Adding new items increases the complexity of the conversion. Strike a balance between the need to maintain functionality and the desire to improve services.

21.5 Training

Related to communication is training. If any aspect of the user experience is going to change, training should be provided. This is true whether the menus are going to be slightly different or entirely new workflows will be required.

Most changes are small and can be brought to people’s attention via email. However, for roll-outs of large, new systems, training is critical to the success of their introduction to an organization. The less technical the customers, the more important that training be included in your roll-out plans.

Creating and providing the actual training is usually out of scope for the SA team doing the service conversion, but SAs may need to support training efforts. Work closely with the customers and management driving the conversion to discover any plans for training support well in advance. Nontechnical customers may not realize the level of effort required by SAs to set up a training room with 5 to 15 workstations, special demo machines and services, or special firewall rules.

21.6 Gradual Roll-Outs

When performing a roll-out, whether it is a conversion, a new service, or an update to an existing service, you should do so gradually to minimize the potential impact of any failures.

Using the “one, some, many” technique discussed in [Section 7.5](#), you would start by converting your own system to the new service. Test and perfect the conversion process, and test and perfect the new service before converting any other systems. When you cannot find any more problems, convert a few of your coworkers’ desktops; debug and fix any problems that arise from that process and have them test the new system. Expand the test group to cover all the SAs before starting on your customers. When you have successfully converted the

SAs, start with small groups of customers and increase the group size over time.

There are a few strategies for picking groups of customers. Random selection is one method. However, when possible be selective: start with customers who are the least risk averse, most technical, or most able to give high-quality feedback. At Bell Labs one team requested to be the first group on roll-outs. They had a particularly good relationship with the IT team. They accepted the risks because they prided themselves on being so cutting edge. They were also highly technical and diligently submitted bug reports. A school district rolling out a new student information system rolled out the grade-tracking subsystem to the high school a year before rolling it out to the rest of the district; the lesson planning subsystem was enabled for the science department a year before anyone else saw it because its members were more tech-savvy than the other departments and could support themselves during the buggy first few rounds.

Likewise, reserve the most risk-averse users for the end of the roll-out process. Some teams may be more risk averse due to business requirements or their temperament. The CEO and anyone involved in approving your budget should be last so that they see systems only after the bugs have been worked out.

Upgrading Google Servers

Google's server farm includes thousands of computers; the precise number is an industry secret. When upgrading thousands of redundant servers, Google has massive amounts of automation that first upgrades a single host, then 1 percent of the hosts, and finally batches of hosts, until all are upgraded. Between each set of upgrades, testing is performed, and an operator has the opportunity to halt and revert the changes if problems are found. Sometimes, the gap of time between batches is hours; at other times, days.

21.7 Flash-Cuts: Doing It All at Once

In a flash-cut, all users are moved to a new service at the same time. You should never do this.

Flash-cuts are highly risky. If there is a problem with the new service, all users will be exposed to the problem. The conversion will go much more smoothly if you can convert a few willing test subjects to the new system first. Avoiding a flash-cut may mean budgeting in advance for duplicate hardware so that you will have a pool of conversion machines to run in parallel. When you prepare your budget request, remember to think about how you will perform the conversion

roll-out.

In other cases, you may be able to use features of your existing technology to slowly roll out the conversion. For example, if you are renumbering a network or splitting a network, you might use an IP multinetting network in conjunction with DHCP to permit unconverted and converted hosts to operate on the same VLAN simultaneously without the need for additional hardware.

Alternatively, you may be able to make both old and new services available simultaneously and encourage people to switch during the overlap period. That way, they can try out the new service, get used to it, report problems, and switch back to the old service if they prefer. Such a strategy gives your customers an “adoption” period. This approach is commonly used in the telephone industry when a change in phone number or area code is introduced. For a few months, both the old and new numbers work. In the following few months, the old number gives an error message that refers the caller to the new number. Then the old number stops working, and sometime later it becomes available for reallocation.

The more complex a change from one service to another, the more tempting it is to do a flash-cut. This is counter-intuitive. For example, transitioning a university-wide email service from one system to another is very complex. There are many moving parts: DNS, client configuration, the server itself, data migration, and so on. It becomes very tempting to simply declare a “flag day” when the old service will disappear, messages will be migrated to the new system, and the new system will go live. We’ve seen this very example many times. All of them ended in failure: The process to migrate messages took longer than expected, the new system was incompatible with a critical application, performance was slow, training the entire organization about how to use the new system became impossible to do at scale, and so on. In every one of these cases, the team was left scratching their collective head, unable to figure out how their excellent plan had failed.

All of these problems would have been prevented if they had converted small groups one at a time rather than doing a flash-cut. The migration process is complex, but can be improved after each batch of users is processed. The process can be paused to fix scaling and performance issues. Training the first group teaches you how to better train future groups.

While flash-cuts seem like a way to reduce complexity, the truth is that they increase the risk of failure. Flash-cuts should be avoided at all costs, even if this means going back to the drawing board and reengineering the design to let it

permit gradual conversions.

Physical Network Conversion

When a midsize company converted its network wiring from thin Ethernet to 10Base-T, it divided the problem into two main preparatory components and had a different group attack each part of the project planning. The first group had to get the new physical wiring layer installed in the wiring closets and cubicles. The second group had to make sure that every machine in the building was capable of supporting 10Base-T, by adding a card or adapter.

The first group ran all the wires through the ceiling and terminated them in the wiring closets. Next, the group went through the building and pulled the wires down from the ceiling, terminated them in the cubicles and offices, and tested them, visiting each cubicle or office only once.

When both groups had finished their preparatory work, they gradually went through the building, moving people to offices with new wiring but leaving the old cabling in place so that they could switch back if there were problems.

This conversion was done well from the point of view of avoiding a flash-cut and converting people over gradually. However, the customers found it too intrusive because they were interrupted three times: once for wiring to their work areas, once for the new network hardware in their machines, and finally for the actual conversion. Although it would have been very difficult to coordinate, and would have required extensive planning, the teams could have visited each cubicle together and performed all the work at once. Realistically, though, this would have complicated and delayed the project too much.

Having better communication initially would have made things easier on the customers, letting them know all the benefits of the new wiring, apologizing in advance for the need to disturb them three times (one of which would require a reboot), and scheduling the disturbances for more convenient times. Customers find interruptions less of an annoyance if they understand what is going on, have some control over the scheduling, and know what they are going to get out of it ultimately.

Sometimes, a conversion or a part of a conversion must be performed simultaneously for everyone. For example, if you are converting from one

corporate-wide calendar server to another, where the two systems cannot communicate and exchange information, you may need to convert everyone at once; otherwise, people on the old system will not be able to schedule meetings with people on the new system, and vice versa.

Performing a successful flash-cut requires a lot of careful planning and some comprehensive testing, including load testing. Persuade a few key users of that system to test the new system with their daily tasks before making the switch. If you get the people who use the system the most heavily to test the new one, you are more likely to find any problems with it before it goes live, and the people who rely on it the most will have become comfortable with it before they have to start using it in earnest. People use the same tools in different ways, so more testers will gain you better feature-test coverage.

For a flash-cut, two-way communication is particularly critical. Make sure that all your customers know what is happening and when, and that you know and have addressed their concerns in advance of the cutover.

If a flash-cut is truly unavoidable, find ways to do testing without a launch. As described in [Chapter 20](#), “[Service Launch: DevOps](#),” set up a lab environment and practice, practice, practice.

Phone Number Conversion

In 2000, British Telecom converted the city of London from two area codes to one and lengthened the phone numbers from seven digits to eight, in one large number change. Numbers that were of the form (171) xxx-xxxx became (20) 7xxx-xxxx, and numbers that were of the form (181) xxx-xxxx became (20) 8xxx-xxxx. More than six months before the designated cutover date, the company started advertising the change; also, the new area code and new phone number combination started working. For a few months after the designated cutover date, the old area codes in combination with the old phone numbers continued to work, as is usual with telephone number changes.

However, local calls to London numbers beginning with a 7 or an 8 went from seven to eight digits overnight. Because this sudden change was certain to cause confusion, British Telecom telephoned every single customer who would be affected by the change to explain, person to person, what the change meant and to answer any questions that their customers might have. Now that’s customer service!

21.8 Backout Plan

When rolling out a conversion, it is critical to have a backout plan. A *conversion*, by definition, means removing one service and replacing it with another. If the new service does not work correctly, the customer has been deprived of one of the tools that he or she uses to perform work, which may seriously affect the person's productivity.

If a conversion fails, you need to be able to restore the customer's service quickly to the state it was in before you made any changes and then go away, figure out why it failed, and fix it. In practical terms, this means that you should leave both services running simultaneously, if possible, and have a simple, automated way of switching someone between the two services.

Bear in mind that the failure may not be instantaneous or may not be discovered for a while. It could be a result of reliability problems in the software, it could be caused by capacity limitations, or it may be a feature that the customer uses infrequently or only at certain times of the year or month. So you should leave your backout mechanism in place for a while, until you are certain that the conversion has been completed successfully. How long? For critical services, one significant reckoning period is a good idea, such as a fiscal quarter for a company, or a semester for a university.

21.8.1 Instant Roll-Back

When performing a conversion, it is nice if your backout plan enables you to instantly roll everything back to a known working state if a problem is discovered. That way, any customer disruption resulting from a problem with the new system can be minimized.

How you provide instant roll-back depends on the conversion that you are performing. One component of providing instant roll-back might be to leave the old systems in place. If you are simply pointing customers' clients to a new server, you can switch back and forth by changing a single DNS record. To make DNS updates happen more rapidly, set the time to live (TTL) field to a lower value—5 minutes, perhaps—well in advance of making the switch. Then, when things are stable, set the TTL back to whatever value is usually in place.

Another approach that achieves instant roll-back is to perform the conversion by stopping one service and starting another. In some cases, you may have two client applications on the customers' machines—one that uses the old system and another that uses the new one. This approach works especially well when the new service has been running for tests on a different port than the existing

service.

For services that are behind a load balancer, you may be able to bring up one or more new instances that are running the new service, and direct a particular group of customers to those instances. This approach enables a quick roll-back while also providing a slow staged roll-out, minimizing the impact of any problems.

Sometimes, the change being made is an upgrade of a software package to a newer release. If the old software is retained in a dormant state on the server while the new software is in use, you can instantly perform a roll-back by switching to the old software. Vendors can do a lot to make this difficult, but some are very good about making it easy. For example, if versions 1.2 and 1.3 of a server get installed in *optexample-1.2* and *optexample-1.3*, respectively, but a symbolic link *optexample* points to the version that is in use, you can roll back the change by simply repointing the single symbolic link.

21.8.2 Decision Point

A major difficulty with backout plans is deciding when to execute them. When a conversion goes wrong, the technicians tend to promise that things will work with “one more change,” but management tends to push toward starting the backout plan. It is essential to have decided in advance the point at which the backout plan will be put into use. For example, one might decide ahead of time that if the conversion isn’t completed within 2 hours of the start of the next business day, then the backout plan must be executed. Obviously, if in the first minutes of the conversion one meets insurmountable problems, it can be better to back out of what’s been done so far and reschedule the conversion. However, getting a second opinion can be useful. What is insurmountable to you may be an easy task for someone else on your team.

When an upgrade has failed, there is a big temptation to keep trying more and more things to fix it. We know we have a backout plan, and we know we promised to start reverting if the upgrade wasn’t complete by a certain time, but we keep on saying, “I just want to try one more thing.” It is natural to want to keep trying. It’s a good thing, actually. Most likely, we got where we are today by not giving up in the face of insurmountable problems. However, when a maintenance window is ending and we need to revert, we need to revert. Often, our egos won’t let us, which is why it can be useful to designate someone outside the process, such as our manager, to watch the clock and make us stop when we said we would stop.

Revert. There will be more time to try again later.

21.9 Summary

A successful conversion project requires lots of advance planning and a solid infrastructure. Customers will judge the conversion by how organized you were, how well you were able to minimize adverse impact to them, and whether the new system “just works.”

A conversion requires many individual steps. With the layers approach, you perform one step for all the customers before moving on to the next step. With the pillars approach, you perform all the required tasks for a particular customer before moving on to the next customer. Ideally a hybrid approach can be used where all the intrusive steps are grouped together (pillars) so customers receive one big intrusive service interruption, and the remaining steps are layered.

The principles for roll-outs of any kind, updates, new services, or conversions are the same. Test changes before they are deployed. Deploy slowly to small groups, then to larger groups. Use vendor support when needed. Have a backout plan, and decide ahead of time the criteria by which the backout plan will be activated. Avoid flash-cuts at all costs.

Exercises

1. Explain the pillars versus layers metaphor and how you can use it to minimize the adverse impacts of service conversions on customers.
2. Which conversions can you foresee in your network’s future? Choose one, and build a plan for performing it with minimum customer impact.
3. Try to add an instant roll-back option to the plan you developed in Exercise 2.
4. If you had to split your network, which services would you need to replicate, and how would you convert people from one network and set of services to the other? Consider each service in detail.
5. Have you made any conversions that could have been avoided? How could you have avoided them?
6. Think about a service conversion that really needs to be done in your environment. Would you do a phased roll-out or a flash-cut? Why?
7. Suppose your IT group is converting everyone from using an office phone system to voice over IP (VoIP). Create an outline of the process using the pillars method. Now create one with the layers method.

8. In Exercise 7, would a hybrid approach be more useful than a strict layers or pillars model? If so, how?

Chapter 22. Disaster Recovery and Data Integrity

A disaster is a catastrophic event that causes a massive outage affecting an entire building or site. A disaster can be anything from a natural disaster, such as an earthquake, to the more common problem of stray backhoes cutting your cables by accident. A disaster is anything that has a significant impact on your company's ability to do business.

A disaster-recovery (DR) plan lists which disasters could hit the company and includes a plan for responding to those disasters. Disaster-recovery planning includes implementing ways to mitigate potential disasters and making preparations to enable quick restoration of key services. The plan identifies what those key services are and specifies how quickly they need to be restored.

All sites need to do some level of DR planning. DR planners must consider what happens if something catastrophic occurs at any one of their organization's sites and how they can recover from it. This chapter concentrates on the electronic data aspects of DR. However, this part of the plan should be built as part of a larger program to meet the company's legal and financial obligations. A number of books are dedicated to disaster-recovery planning, including these resources:

- *The Business Continuity Management Desk Reference: Guide to Business Continuity Planning, Crisis Management & IT Disaster Recovery* by Watters ([2010](#))
- *Disaster Recovery Planning: Getting to Business-Savvy Business Continuity*, 4th ed., by Toigo ([2003](#))
- *A Practical Guide to Risk Management* by Coleman ([2011](#))

Building a disaster-recovery plan involves understanding both the risks that your site faces and your company's legal and fiduciary responsibilities. From this basis, you can begin your preparations. This chapter describes what is involved in building a plan for your site.

As with any project, building a disaster-recovery plan starts with understanding the requirements: determining which disasters could afflict your site, the likelihood that those disasters will strike, the cost to your company if they do strike, and how quickly the various parts of your business need to be revived. Once you and your management understand what can happen, you can get a budget allocated for the project and start looking at how to meet and, preferably, beat those requirements.

22.1 Risk Analysis

The first step in building a disaster-recovery plan is to perform a risk analysis. A risk analysis involves determining which disasters the company is at risk of experiencing and what the chances are of those disasters occurring. The analyst also determines the likely cost to the company for each disaster should it occur. The company then uses this information to decide approximately how much money it is reasonable to spend on trying to mitigate the effects of each type of disaster.

Since risk management involves specialized skills that most employees don't use daily, it can be wise to hire external consultants to perform the risk analysis. A large company may hire specialists to produce a risk report, while having an in-house person be responsible for risk management.

Lack of Planning Can Cause Risk Taking

A computer equipment manufacturer had a fire in its facility in the west of Ireland. Knowing that the building's fire protection system was inadequate, several staff members went to the datacenter and started throwing equipment out the window because the equipment had a better chance of surviving the fall than the fire. Other staff members then carried the equipment up the hill to their neighboring building. The staff members in the burning building left when they judged that the fire hazard was too great.

All the equipment that they had thrown out the window actually survived, and the facility was operational again in record time. However, the lack of a disaster-recovery plan and adequate protection systems resulted in staff members' risking their lives. Fortunately, no one was badly injured in this incident. Nevertheless, their actions were in breach of fire safety codes and extremely risky because no one there was qualified to judge when the fire had become too hazardous.

The approximate budget for risk mitigation is determined by the formula $(D - M) \times R$, where D is the probable cost of disaster, M is the probable cost after mitigation, and R is the risk of disaster.

For example, if a company's premises has 1 chance in 1 million of being affected by flooding, and if a flood would cost the company \$10 million, the budget for mitigating the effects of the flood would be in the range of \$10. In other words, it's not even worth stocking up on sandbags in preparation for a

flood. In contrast, if a company has 1 chance in 3,000 of being within 10 miles of the epicenter of an earthquake measuring 5.0 on the Richter scale, which would cause a loss of \$60 million, the budget for reducing or preventing that damage would be in the \$20,000 range.

A simpler, smaller-scale example is a large site that has a single point of failure, in that all LANs are tied together by one large router. If it died, it would take one day to repair, and there is a 70 percent chance that failure will occur once every 24 months. The outage would cause 1,000 people to be unable to work for a day. The company estimates the loss of productivity to be \$68,000. The SAs are given a budget of \$23,800 to build router redundancy. The SAs also need to investigate the cost of reducing the outage time to 4 hours—for example, by expanding contracts with vendor support. If that costs a reasonable amount, it further reduces the amount the company would lose and, therefore, the amount it should spend on full redundancy.

This example is somewhat simplified. Each disaster can occur to different degrees with different likelihoods and have a wide range of cost implications. Damage prevention for one level of a particular disaster will probably have mitigating effects on the amount of damage sustained at a higher level of the same disaster. All this complexity is taken into account by a professional risk analyst when he or she recommends a budget for the various types of disaster preparedness.

22.2 Legal Obligations

Beyond the basic cost to the company, additional considerations need to be taken into account as part of the DR planning process. Commercial companies have legal obligations to their vendors, customers, and shareholders in terms of meeting contract demands. Public companies have to abide by the laws of the stock markets on which they are traded. Some industries, such as banking, are governed by local laws relating to business continuity. Universities have contractual obligations to their students. Building codes and work-safety regulations also must be followed.

The legal department should be able to elaborate on these obligations. Typically, they are of the form “The company must be able to resume shipping product within one week” or “The company can delay reporting quarterly results by at most three days under these circumstances.” Those obligations translate into requirements for the DR plan. They define how quickly various pieces of the physical and electronic infrastructure must be restored to working order. Restoring individual parts of the company to working order before the entire

infrastructure is operational requires an in-depth understanding of which pieces of infrastructure those parts rely on and a detailed plan of how to get them working. Meeting the time commitments also requires an understanding of how long restoring those components will take. We look at that issue further later in this chapter.

22.3 Damage Limitation

Damage limitation is about reducing the cost of the disasters. Most damage-limitation efforts involve additional cost to the company and are subject to the cost/benefit analysis that is performed by the risk analysts. Some damage limitation can come at little or no cost to the company through advance planning and good processes.

For example, in an area prone to minor flooding, placing critical services above ground level may not significantly increase construction and move-in costs, but might prevent flood disasters in the future. Choosing rack-mountable equipment and reasonably sturdy racks to bolt it into, rather than putting equipment on shelves, can significantly reduce the impact of a minor earthquake for little or no extra cost. Putting lightning rods on a building in an area that is prone to lightning storms is also a cheap way of limiting damage. These solutions are particularly economical because they fix the problem once, rather than requiring a recurring cost.

Limiting the damage caused by a major disaster is more costly and always should be subject to a cost/benefit analysis. For example, a datacenter could be built in an underground military-style bunker to protect against tornados and bombs. In an earthquake zone, expensive mechanisms allow racks to move independently in a constrained manner to reduce the risk of computer backplanes' shearing—the major issue with rigidly fixed racks during a strong earthquake. These mechanisms for limiting damage can be so costly that only the largest companies are likely to be able to justify implementing them.

Most damage-limitation mechanisms fall somewhere between almost free and outlandishly expensive. Fire-prevention systems typically fall into the latter category. It is wise to consider implementing a fire-protection system that is designed to limit damage to equipment in the datacenter when activated. Local laws and human safety concerns limit what is possible in this area, but popular systems currently include gas-based systems that remove heat or oxygen and water-based systems with early-warning mechanisms that permit an operator to detect and resolve a problem before there is a fire, such as a disk or power supply overheating. Systems for detecting moisture under raised datacenter

floors or in rarely visited UPS or generator rooms are also moderately priced damage-limitation mechanisms.

Another area that often merits attention is loss of power. Short power outages, spikes, or brownouts can be handled by a UPS; longer interruptions will require a generator. The more equipment on protected power and the longer you need to be able to run that equipment, the more it will cost. Typically the most essential equipment should be protected against power loss—for example, freezers at a biotech company and call centers at a customer service company.

22.4 Preparation

Even with a reasonable amount of damage control in place, your organization may still experience a disaster. Part of your disaster planning must include preparation for this eventuality. Being prepared for a disaster means being able to restore the essential systems to working order in a timely manner, as defined by your legal, ethical, and fiduciary obligations.

Restoring services after a disaster can require rebuilding the necessary data and services on new equipment if the old equipment is not operational. Thus, you need to arrange a source of replacement hardware in advance from companies that provide this service. You also need to have another site to which this equipment can be sent if the primary site cannot be used because of safety reasons, lack of power, or lack of connectivity. Make sure that the company providing the standby equipment knows where to send it in an emergency. Make sure that you get turnaround time commitments from the provider and that you know which hardware the company will be able to provide on short notice. Don't forget to take the turnaround time on this equipment into account when calculating how long the entire process will take. If a disaster is large enough to require the company's services, chances are that company will have other customers that also are affected. Find out how the company plans to handle the situation in which both you and your neighbor have the right to the only 50 KW generator the vendor has in your metro area.

Once you have the machines, you need to recreate your system. Typically, you first rebuild the system and then restore the data. This requires that you have backups of the data stored off-site—usually at a commercial storage-and-retrieval service. You also need to be able to easily identify which tapes are required for restoring the essential services. This basic part of the preparation plan relies on infrastructure that your site should have already put in place. An ongoing part of disaster preparation involves retrieving tapes from the off-site storage company on a regular basis to see how long it takes. This time is

subtracted from the total amount of time available to completely restore the relevant systems to working order. If it takes too long to get the tapes, it may be impossible to complete the rebuild on time. For more on these issues, see [Chapter 44](#), “[Backup and Restore](#),” particularly [Section 44.7.2](#).

A site usually will need to have important documents archived at a document repository for safekeeping. Such repositories specialize in DR scenarios. If your company uses a repository, you may want to consider using it to house the data tapes as well.

Remember that you may need power, telephone, and network connectivity as part of restoring services. Work with the facilities group on these aspects of DR planning. It may be advisable to arrange for an emergency office location to house critical functions as part of the disaster plan.

Good Preparation for an Emergency Facility

A company in California provided call center services for its clients, which were predominantly large financial institutions. The company had a well-rehearsed procedure to execute in case of a disaster that affected the call center building. The company had external outlets for providing power and call center phone services, spare cables and equipment were maintained in reserve, and items such as tents, and folding tables and chairs, were kept in case they were needed.

When a strong earthquake struck in 1991, the call center was rapidly relocated outside and became operational again within minutes. Not long after it was set up, it received lots of calls from the company’s customers in New York, which wanted to make sure that services were still available if they required them. The call center staff calmly reassured customers that all services were operating normally. Customers had no idea that they were talking to someone who was sitting on a chair in the grass outside the building. The call center had to remain outside for several days until the building was certified safe—but it remained operational the entire time.

The plan to relocate the call center outside in the case of emergency worked well because in California the most likely emergency was an earthquake and the weather was likely to be dry, at least long enough for the tents to be put up. The company had prepared well for its most likely disaster scenario.

22.5 Data Integrity

Data integrity means ensuring that data is not altered by external sources. Data can be corrupted maliciously by viruses or individuals. It can also be corrupted inadvertently by individuals, bugs in programs, and undetected hardware malfunctions. For important data, consider ways to ensure integrity as part of day-to-day operations or the backup or archival process. For example, data that should not change can be checked against a read-only checksum of the data. Databases that should experience small changes or have only data added should be checked for unexpectedly large changes or deletions. Examples include source code control systems and databases of gene sequences. Exploit your knowledge of the data on your systems to automate integrity checking.

Disaster planning also involves ensuring that a complete and correct copy of the corporate data can be produced and restored to the systems. For disaster-recovery purposes, a company will need a recent, coherent copy of the data with all databases in sync. Data integrity meshes well with disaster recovery.

Industrial espionage and theft of intellectual property are not uncommon, and a company may find itself needing to fight for its intellectual property rights in a court of law. The ability to accurately restore data as it existed on a certain date can also be used to prove ownership of intellectual property. To be used as evidence, the date of the information retrieved must be accurately known, and the data must be in a consistent state. For both disaster-recovery purposes and use of the data as evidence in a court, the SAs need to ensure that the data has not been tampered with.

It is important to make sure that the implementers put in place the data-integrity mechanisms that the system designers recommend. It is inadvisable to wait for corruption to occur before recognizing the value of these systems.

22.6 Redundant Sites

The ultimate preparation for a disaster is to have fully redundant versions of everything that can take over when the primary version fails. In other words, an organization may have a redundant site with redundant systems at a location that will not be affected by the same disaster.

Although this sounds expensive, in large companies, especially banks, having a redundant site is a minimum standard. In fact, large companies have stopped using the term *disaster recovery* and instead use the term *contingency planning* or *business continuity planning*.

There are a few different design patterns for this sort of redundancy.

Moreover, different services may follow different design patterns, as described in [Chapter 18](#), “[Service Resiliency and Performance Patterns](#).”

To reduce expense, some companies may decide to use a second site as an alternative location for rebuilding services, rather than maintaining live redundant equipment at site. If the company has a contract for an emergency supply of equipment, that equipment could be sent to the alternative datacenter site. If the site that was affected by the disaster is badly damaged, this may be the fastest way to have the services up and running. Another option is to designate some services as less critical and to use the equipment from those services to rebuild the critical services. Sometimes, you are lucky enough to have a design that compartmentalizes various pieces, making it easy to design a redundant site.

22.7 Security Disasters

A growing concern is security disasters. Someone breaks into the corporate web site and changes the logo to something obscene. Someone steals the database of credit card numbers from an e-commerce site. A virus deletes all the files it can access. Unlike with natural disasters, no physical harm occurs with a security disaster, and the attack may not be from a physically local phenomenon.

A similar risk analysis can be performed to determine the kinds of measures required to protect data. Architecture decisions, for example, always have a risk component. One can manage this risk in many ways—for example, by building barriers around the system or by monitoring the system so that it can be shut down quickly in the event of an attack.

Sites often purchase large, canned systems without asking for an explanation of the security risks of such systems. Although no system is perfectly secure, a vendor should be able to explain the product’s security structure, the risk factors, and how recovery would occur in the event of data loss.

22.8 Media Relations

When a disaster occurs, the media will probably want to know what happened, what effect it is having on the company, and when services will be restored. Sadly, the answer to all three questions is usually “We aren’t sure.” This can be the worst answer you can give a reporter. Handling the media badly during a disaster can cause bigger problems than the original disaster.

We have two simple recommendations on this topic. First, have a public relations (PR) firm on retainer so that you aren’t trying to hire one as the disaster

is unfolding. Some PR firms specialize in disaster management, and some are proficient at handling security-related disasters. Second, plan ahead of time how you will deal with the media. This plan should include who is allowed to talk to the media, which kinds of things will and will not be said, and what the chain of command is if the designated decision makers aren't available. Anyone who talks to the media should receive training from the PR firm.

Note that these recommendations all have one thing in common: They require planning ahead of time. Never be in a disaster without a media plan. Don't try to write one during a disaster.

22.9 Summary

The most important aspect of disaster planning is understanding which services are the most critical to the business and what the time constraints are for restoring those services. You must know which disasters are likely to happen and how costly they would be before you can complete a risk analysis and determine the company's budget for limiting the damage.

A disaster plan should be built with consideration of those criteria. It should account for the time to get new equipment, retrieve the off-site backups, and rebuild the critical systems from scratch. These steps require advance planning for getting the correct equipment and being able to quickly determine which backup tapes are needed for rebuilding the critical systems.

Look for simple ways to limit damage, as well as more complex and expensive ways. Preparations that are automatic and become part of the infrastructure are most effective. Fire containment, water detection, earthquake bracing, and proper rack-mount equipment fall into this category. You also must prepare a plan for a team of people to execute in case of emergency. Simple plans are often the most effective. The team members must be familiar with their individual roles and should practice disaster recovery a few times a year.

Full redundancy, including a redundant site, is beyond the budget of most companies. If a company has a second datacenter site, however, there are ways to incorporate it into the disaster plan at reasonable expense.

Exercises

1. What is disaster-recovery planning?
2. What are the key elements of a disaster-recovery plan?
3. How much should be spent to mitigate a potential disaster?

- 4.** Which business units in your company would need to be up and running first after a disaster, and how quickly would they need to be operational?
- 5.** Which commitments does your company have to its customers, and how do those commitments influence your disaster planning?
- 6.** Which disasters are most likely to hit each of your sites? How big an area might each disaster affect, and how many of your company's buildings could be affected?
- 7.** What would the cost be to your company if a moderate disaster hit one of its locations?
- 8.** Which forms of disaster limitation do you have in place now?
- 9.** Which forms of disaster limitation would you like to implement? How much would each of them cost?
- 10.** If you lost the use of a datacenter facility because of a disaster, how would you restore service?
- 11.** What are your plans for dealing with the media in the event of a disaster? What is the name of the PR firm you have retained to help you?

Part V: Infrastructure

Chapter 23. Network Architecture

This chapter is about the design of office networks (wired and wireless), datacenter networks, and the wide area networks (WANs) that connect them. The next chapter focuses on operating these networks.

A site's network is the foundation of its infrastructure. A poorly built network affects all other components of the system. A network cannot be considered in isolation. Decisions made as part of the network design and implementation process influence how infrastructure services are implemented. Therefore, the people who are responsible for designing those services should be consulted as part of the network design process.

23.1 Physical Versus Logical

A network can be depicted in two different ways: as the physical network and as the logical network. The physical network consists of the physical wires and devices that make up the network. The logical network describes the software-based partitions, segments, and connections that we overlay on the physical network.

Hardware is inflexible. In the old days the physical and logical networks were basically the same thing. If you wanted two LANs, one for sales and one for engineering, it required two sets of network switches. Each PC would be plugged into a network jack that was connected to one switch or the other. Now we build a single physical infrastructure and carve out the individual LAN segments as VLANs. That is, one switch can segregate traffic based on which port it arrived on.

VLANs make networks easier to manage. In the past, when an employee moved offices, someone had to physically go to the wiring closet and move that employee's connection from one switch to another. Now we can make the equivalent change by setting the VLAN assignment of the employee's port.

VLANs Versus Subnets

Sometimes the terms VLAN and subnet are used interchangeably. In truth, they are different, but related, concepts.

A subnet is a network segment that can be routed, and is not further subdivided, such as 192.168.128.0/23 (IPv4) or fd00::1/64 (IPv6). It contains a set of IP addresses that can be assigned to network interface

cards (NICs). Two devices in the same subnet communicate directly; the packets do not need to be routed through a layer 3 device such as a router.

A VLAN is a construct that is implemented in switch software to group switch ports into different LAN segments, or broadcast domains. In theory, multiple subnets can be assigned to a VLAN, though usually only one is. Therefore the VLAN assigned to each switch port usually determines which subnet a device connected to that port is in.

Each VLAN is also a broadcast domain. Most traffic is unicast: Packets are sent from one machine to another. Some traffic is broadcast: Every device on the VLAN receives and processes the packet. This type of traffic is used sparingly since bandwidth is limited. A network can become overloaded if a large amount of broadcast traffic is being sent. Note that the VLAN is the broadcast domain, not the subnet.

23.2 The OSI Model

The Open Systems Interconnection (OSI) reference model for networks has gained widespread acceptance and is used throughout this chapter. The model looks at the network as logical layers and is briefly described in [Table 23.1](#).

Layer	Name	Description
1	Physical	The physical connection between devices: copper, fiber, radio, laser
2	Data link	Interface (or MAC) addressing, flow control, low-level error notification
3	Network	Logical addressing (e.g., IP addresses) and routing (e.g., RIP, OSPF, IGRP)
4	Transport	Data transport, error checking and recovery, virtual circuits (e.g., TCP sessions)
5	Session	Communication-session management (e.g., AppleTalk name binding, or PPTP)
6	Presentation	Data formats, character encoding, compression, encryption (e.g., ASCII, Unicode, HTML, MP3, MPEG)
7	Application	Application protocols (e.g., SMTP for email, HTTP for web, and FTP for file transfer)

Table 23.1: The OSI Network Model

Network devices decide the path that data travels along the physical network, which consists of cables, wireless links, and network devices (layer 1). A network device that makes those decisions based on the hardware or MAC address of the source or destination host is referred to as a layer 2 device. A device that makes decisions based on the IP (or AppleTalk or DECnet) address of the source or destination host is known as a layer 3 device. One that uses transport information, such as TCP port numbers, is a layer 4 device.

Engineers more familiar with TCP/IP networking often simplify this as follows: layer 1 is the physical cable; layer 2 is devices that deal with a particular LAN; layer 3 is the routers and gateways that route packets between LANs; layer 4 is the protocol being used.

Layer 5, the session layer, is a layer that doesn't map well into the world of TCP/IP, except for point-to-point tunnel protocols (PPTP), such as generic routing encapsulation (GRE) tunnels. Layer 6 is the data format: ASCII, HTML, MP3, or MPEG. Encryption and compression are usually handled here as well.

Layer 7 is the application protocol itself: HTTP for web serving, SMTP for email transmission, IMAP4 for email access, FTP for file transfer, and so on.

The OSI model is a useful guideline for understanding the way networks are intended to work, but many layering violations occur in the real world. For example, a VPN connection made through an HTTPS proxy sends layers 3 and 4 traffic over a layer 7 application protocol.

Layers 8, 9, and 10

A common joke is that the OSI model has three additional layers:

- Layer 8: User
- Layer 9: Financial
- Layer 10: Political

Many corporate network architectures focus on solving problems at layer 10 but are limited by layer 9 in what they can achieve.

23.3 Wired Office Networks

Office networks tend to have a mixture of wired and wireless (WiFi) access. Wired connections are more reliable because they are a physical connection, with no radio interference as with WiFi. They are also faster. The fastest speed available in a wired interface is usually three to five years ahead of that available with WiFi technology.

The speed and lack of interference are particularly noticeable for people using high-bandwidth or low-latency applications, such as soft phones, voice over IP (VoIP), streaming video, and multimedia conferencing systems.

23.3.1 Physical Infrastructure

For a wired network, a person plugs a device into a network jack at his or her desk, in a conference room, in a lab, or wherever the person happens to be. That network jack, or drop, is at the end of a physical cable that runs through walls and ceilings until it terminates in another jack in a wiring closet (also called an intermediate distribution frame [IDF]).

Replacing this infrastructure wiring is expensive and disruptive. Therefore anytime you are involved in commissioning a new building, or renovating an old one, make sure the highest available specification wiring is run, with more cables than you think you will ever need.

In the IDF, each jack is patched to a switch port using a short network cable, also called a patch cable. When switches were expensive (multiple hundreds of dollars per port), it made economic sense to connect only the actively used ports to switches. If a previously unused jack needed to be activated, someone had to visit the office and make a connection in the IDF room to enable that port. If there were no more ports left, a new switch had to be installed. Today switches are inexpensive. It is significantly less expensive to pre-wire each jack to a switch port. The cost of the additional switches is more than covered by the increase in productivity for the end users with a plug-and-play environment, as compared to an environment where changes require a series of tasks, including an on-site visit, to be completed before a jack is brought live. Also, IDFs are more secure and more reliable if visits are extremely rare.

In a small company with a single office area, the IDF will be inside the computer room, where the office network switches are connected into the core network. Usually it will be a rack in the computer room.

In large buildings, there will be several IDFs—typically one or more per floor. Usually the IDFs are in the same approximate location on each floor. Sometimes there will be some inter-IDF wiring. Always, the IDFs will be connected back to a main distribution frame (MDF). The connections from the IDFs to the MDF need to be higher bandwidth than the connections from desk to IDF, because the traffic from many jacks is aggregated into a single uplink. The MDFs link back to the datacenter, or to a primary MDF, which connects back to one or more datacenters. For midsize companies, the campus MDF may be a row of racks at

the back or front of the datacenter.

23.3.2 Logical Design

On top of the physical infrastructure is the logical design. The office network is divided into individual network segments. One of the factors to consider when deciding on the size of the office subnets is the amount of broadcast traffic there will be on the network. This is a combination of the number of devices and the amount of broadcast traffic generated by the device OS and applications it runs.

While it varies depending on usage pattern, a general rule of thumb is that approximately 400 well-behaved (or 50 ill-behaved) hosts can reasonably share a network segment. However, a stronger limit on the number of hosts per VLAN is uplink capacity. The uplink is the connection between this VLAN and the other VLANs. If 100 hosts are on a VLAN and all generate 10 Mbps of traffic destined to other VLANs, then the uplink must be at least 1000 Mbps.

For simplicity and ease of support, each VLAN should contain a single subnet. The primary strategies for dividing an office LAN into subnets are based on floor plan, security level, or device type:

- **One big subnet:** There is a single subnet. This design works well for a small office. A single subnet can have hundreds of machines. The primary limit is broadcast traffic, as described previously.
- **Floor plan–centric:** One subnet is allocated to each hallway or area of a building. Each area is planned to be the right size such that the number of devices will not overload the network and the uplink capacity. If all users and devices are at the same security level, this is the simplest way to scale a network.
- **Security level:** Privileged users are on a different subnet. This model is used when filtering and access are based on IP address. For example, there may be a firewall that prevents anyone but support engineers from accessing production servers. The support engineers are put on a special VLAN and that IP subnet is then permitted, by the firewall, to access the production servers.
- **Change control level:** Subnet assignment is based on stability needs. When there is a lot of change control, it can be useful to have separate subnets, and often separate network hardware, for more dynamic environments, such as engineering labs, and less for more staid environments such as general office use. That way changes can be made in those environments without the need to wait for change windows. In

addition, the production environment can be protected by firewalls from the lab environment.

- **Device function:** Particular devices are put on separate subnets. A subnet per device function is commonly used to separate out untrusted devices. For example, one company had strict rules about security requirements for machines on corporate networks. However, the door lock system was based on machines that could not be audited. Therefore, those devices were isolated to a separate network. With the Internet of Things (IoT), there are a lot more network-enabled devices than there used to be. For example, with smart lighting, every lightbulb is network enabled. Often these Things do not have very good security, and they may not receive regular patches. You may want to bundle them into separate subnets, where you can protect against a rogue lightbulb sending spam or being used as part of a botnet to launch a DDoS attack after it has been hacked.

VLAN Myths

Many people think that VLANs provide security between devices that are on separate VLANs on the same switch. However, VLANs are not as secure as you might think.

Switches have an internal table that maps a MAC address to a switch port, usually called a CAM table. This table can usually contain only a fixed number of entries. When the CAM table size is exceeded, switches often stop acting like a switch and start acting like a bridge, sending all traffic to all ports. This behavior opens the door to direct communication between devices in different VLANs. If these devices are in different security zones, this communication will bypass any network-based controls, such as firewalls, that you have put in place.

Another myth is that VLANs add bandwidth. Some people think that separating devices into more VLANs on a congested switch will improve network performance. However, switch backplanes and uplinks have fixed bandwidth.

Switches send traffic that is directed to a specific host, but only to the specific port to which the host is connected. A port can run at maximum bandwidth without affecting traffic on other ports, regardless of how many VLANs there are. However, the backplane of the switch needs to handle traffic for all the ports that are connected to it. Typically switch backplanes are not designed to have enough capacity such that all of the

switch ports can run at full capacity simultaneously. Dividing the ports into more VLANs will not change the amount of traffic that needs to cross the backplane or the capacity of the backplane. Equally, traffic to and from devices that are on other switches needs to cross an uplink to the backbone, or another switch, and this uplink also has fixed capacity. If the uplink is congested, changing the VLAN configuration of the switch will not make any difference to the amount of traffic that needs to cross the uplink, or to the capacity of the uplink.

23.3.3 Network Access Control

Network access control (NAC) determines whether a device that connects to your network is permitted to join, and which VLAN it should be in. NAC is relatively rare today but should be a requirement for corporate networks. Without NAC, anyone can find a free jack and plug right into the corporate network. This is a security nightmare.

With NAC, a device that connects to a jack in an office is first authenticated. Once that step is complete, the NAC system reconfigures the switch port to be in the appropriate VLAN. NAC options include the following:

- **MAC based:** NAC involves checking the MAC address of the machine against a database of authorized MACs. It is the weakest form of NAC, because the MAC address of an unauthorized machine can easily be changed to an authorized MAC.
- **Authentication based:** NAC involves the end user supplying credentials to connect to the network. These may be a simple username and password or two-factor authentication, such as a token or smartcard. Depending on the integration with the OS and the company's authentication database, it may be that when the person has authenticated to his or her machine, that person has also authenticated to the network. However, if the authentication is explicit each time the person connects to the network, it will give a poor end-user experience.
- **Certificate based:** NAC involves a one-time verification that the person is who he or she claims to be, and then installs a certificate on the person's machine. Once the certificate is installed, the end user has a seamless experience thereafter.

There are plenty of NAC solution vendors. Look for one that will give a good end-user experience, centralized management, and integration into your existing authentication and authorization systems. Also make sure that the same solution

will support both your wired and wireless environments.

23.3.4 Location for Emergency Services

Another consideration for your logical network design is how you will pass location information to emergency services. If someone dials emergency services from the soft phone on his or her workstation to report a life-threatening situation, how will emergency services be able to find that person? As a network administrator, you need to be able to reliably turn an IP address into a physical location. With the strategy of using a VLAN per hallway or area, you can automatically provide emergency services with a location, no matter which building of your global enterprise the person happens to be in at the time. Other VLAN strategies require a different approach, such as identifying the switch port being used by that IP address, and mapping that to a location. However, sometimes simply providing the building address may be sufficient.

23.4 Wireless Office Networks

Wireless networks (WiFi) have gone from being an expensive niche technology to the primary way for most people to access a network. Without WiFi, mobile devices would not be very usable.

Wireless networks are more convenient than wired networks. There is no need to search for a free jack and a long-enough cable. Your connection-based applications are not dropped when you move from your desk to the meeting room and back. Also, upgrades to the speed of the network do not require ripping open the walls to replace wiring. Doing so will require opening the ceilings to replace the wireless access points, and perhaps drilling into the ceiling to securely mount the new brackets. Even so, these steps are still less disruptive and cheaper than replacing the physical cable plant.

23.4.1 Physical Infrastructure

The most important aspects of a wireless network are that it has good coverage and sufficient bandwidth. Don't install a home WiFi base station and expect to have a working office network. An office is larger and has more users, more sources of interference, and higher bandwidth demands.

Work with a vendor of enterprise-class wireless access points to make sure that your wireless coverage is solid and you choose the appropriate devices. You will need to know how many wireless devices and which protocols you will need to support. Work with the vendor to set up a proof of concept (PoC) in your

office before you agree on a final configuration and bill of materials (BoM).

23.4.2 Logical Design

As with wired networks, you should have some network access control. NAC will put devices into different VLANs, based on the type of device and device authentication. For example, you may put company-owned devices into one VLAN, employee-owned devices into another, and guest devices into a third. You may have yet other VLANs to isolate various types of IoT devices. Your wired and wireless network access control systems should be unified.

Wireless access should be unified across the company. When an executive travels from her usual workplace in Johannesburg to offices in London, Seoul, New York, Beijing, and St. Petersburg her devices should automatically join the local wireless networks. This means using the same wireless service set identifiers (SSIDs) in all locations, and having unified authentication and access control.

Another aspect of your VLAN strategy for wireless networks is to consider the fact that wireless devices are usually mobile. If you have a large building, what happens when users walk through the building with their phone and tablet? Should they move from one VLAN to another as they move through the building? What impact does this have on VoIP calls, or other applications that they are using while on the move? Typically, for the best customer experience, you want to keep devices in the same VLAN for the entire time that users are in range of your WiFi service. This is achieved by using enterprise-class features often called IP or WiFi “roaming.” On a related note, providing location information to emergency services is also a consideration for wireless devices. Being able to associate an IP address with the WiFi access point through which the device is connecting to the network should give sufficient locality.

While you might not permit guests on your wired network, it has become accepted practice to have a guest wireless network. It should provide an environment that is open to the Internet but has no direct access to the corporate network. Visitors and business partners will often want to demonstrate their products over the Internet, using their own devices. Also, if employees have friends or partners visiting the office, and they are waiting around while the employee gets ready to leave, you would rather that the employee sets them up with guest WiFi access than with access to the corporate network.

Guest WiFi Restrictions

If your corporate network censors access to particular web sites, consider whether your guest network should as well. If it does not, people can evade your filtering policy by using the guest network.

However, this can cause great complications for visitors. For example, a high school filtered YouTube except for specifically permitted videos. Guest speakers who wished to present video material were often stymied when they discovered that their video was blocked and requesting it to be unblocked would take longer than their visit.

Some authentication should still be required for the guest WiFi so that people can't just sit outside your office and use your company's infrastructure to launch attacks against other sites. Some companies provide a self-service portal for employees to apply for limited-time guest WiFi access, with a temporary, unique username and password. Others have a fixed username and password combination that is readily made available to anyone inside the building, and is changed periodically.

We believe that guest WiFi should be like tea. If your guests want tea, you graciously give it to them. You don't ask them to sign a contract, enter a complex password, or install a special tea-enabling software package. You just give them a nice cup of tea. Likewise, guest WiFi should be simple. Have a simple password that is readily available. If there is a portal, minimize the clicks required. Don't require the user to scroll to the bottom of a lengthy acceptable use policy (AUP), then check multiple check boxes, then click a submit button. Instead, have a single button at the top of the page that reads, "I agree to the policy below."

23.5 Datacenter Networks

The LAN in your datacenter has different requirements than the LAN in your office. The datacenter has a higher density of machines and higher bandwidth demands. Rather than jacks connected to IDFs, we have racks of equipment.

As with office networks, a datacenter network design can be separated into physical and logical aspects. The physical design covers the network devices and the cable plant. The logical design describes the subnet strategy for the datacenter. Both the logical and physical designs should enable you to provision any datacenter subnet anywhere within the datacenter, at least for your primary (largest) security zone. That way, your network design does not limit how you can utilize the datacenter space, or cause delays and extra work in the installation

process.

Case Study: Design Limitations Complicate Operations

One multinational company had large datacenters that had grown significantly and organically over several decades. Many of the datacenters consisted of several rooms with only limited conduit space for inter-room wiring. The datacenter network had also grown organically, and could not accommodate provisioning all VLANs in every room of a given datacenter.

Server installation could turn into a multi-week process. Because it was a large company, several different teams were involved. The datacenter team needed to identify a rack location where there was sufficient power and cooling. The network team needed to identify networks in that room that met the server team's requirements for the primary, backup, and any other interfaces, and allocate switch ports. The DNS team needed to allocate IPv4 addresses on those networks. A problem with any of these steps would result in having to start from scratch. For example, if there were no free IPv4 addresses on a selected network, the network team would have to find another candidate network; but they might find that this network was not available in the selected room, so the datacenter team would need to go looking for rack space in one of the rooms where that network was available. If there was insufficient power or rack space in the rooms where all the necessary networks were provisioned, the process would start again.

A network design that enables every datacenter network to be provisioned anywhere within the datacenter greatly simplifies datacenter operations, and reduces operational overhead costs. This could be achieved by a physical and logical design that permits any VLAN in any rack.

23.5.1 Physical Infrastructure

There are a number of different strategies for wiring a datacenter. The three most popular are having a central switch, individual switches in each rack, and datacenter fabric systems.

Central Switch with Patch Panels

This is the most basic design. The datacenter has a single core switch, and each

machine connects to it. To make this manageable, each machine in a rack connects to a patch panel in the same rack. A patch panel is pictured in [Figure 23.1](#).

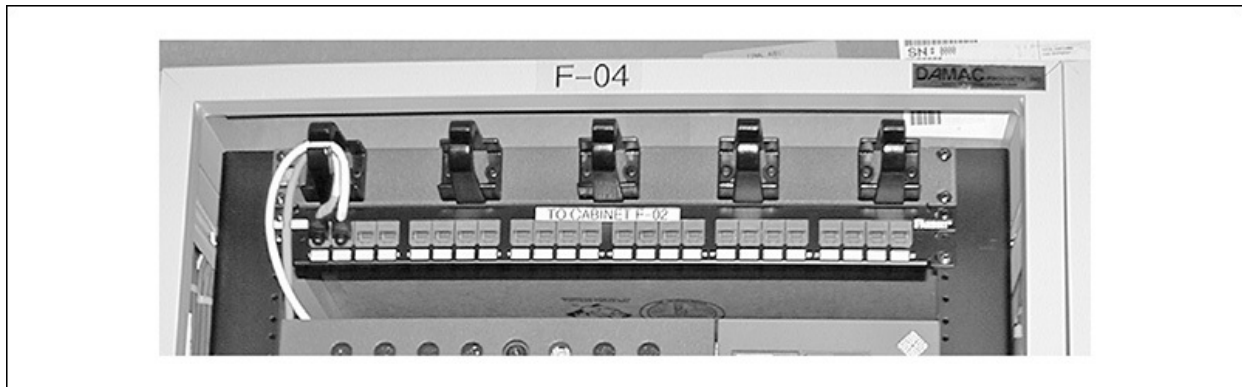


Figure 23.1: Patch panel at the top of a rack

The patch panel brings connections to the network rack, and a patch cable connects the machine to a switch port. A network rack is a rack reserved for interconnections and network equipment. One is pictured in [Figure 23.2](#).

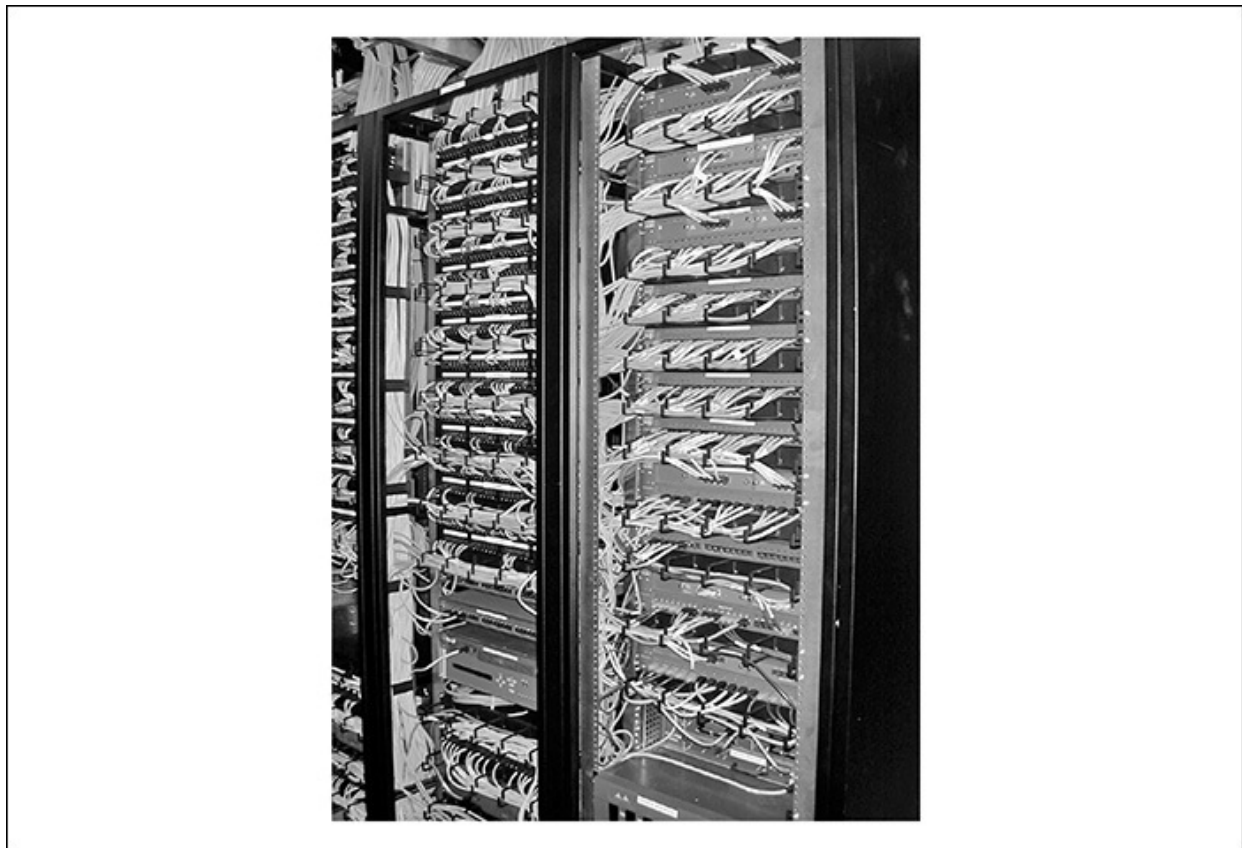


Figure 23.2: Patch panels in a network rack

In a larger datacenter there are multiple core switches that are interconnected

to act as one large network. Some datacenters have a network rack at the end of each row. To improve resiliency each machine may have multiple connections, each to a different switch.

The downsides of this design are that patch panels take up a lot of valuable datacenter space, and that this approach involves a lot of cabling. If there are 40 machines in each rack, with two connections each, 80 cables must be run to the network rack. This is difficult to manage and nearly impossible to upgrade.

One TOR per Rack Connected to a Core

This is a more modern configuration. Each rack has its own switch called a TOR (top of rack) switch, even if it isn't at the top of the rack. Machines in a rack connect to the rack's TOR. The TORs all have high-speed connections to a central switch or switches.

To improve resiliency each rack has two TOR switches and each machine has one connection to each TOR. The TORs have redundant connections to the core switches. With this design, any one switch can be taken down for maintenance without loss of service. Upgrading to faster technologies is easier since fewer cables are run between racks.

The downsides of this design are that the bandwidth out of the rack is limited by the switch uplink, and that this approach involves a lot of switches, each of which must be maintained. Imagine 40 racks with 2 switches each. Monitoring and upgrading 80 switches is a lot of work!

TORs Connected as a Fabric

This uses more sophisticated ways to connect the TORs that provide better management, bandwidth, and resiliency. In a unified fabric product, each TOR is an extension of the core switch. All configuration and operational procedures are done on the core switch rather than having to log into each individual TOR. It is as if you have one big switch for the entire datacenter. This greatly simplifies management and upgrades.

Clos fabrics use what's known as a Clos topology to permit groups, or pods, of racks to have almost infinitely scalable bandwidth to the rest of the network by providing multiple paths between any two points. Traditional network designs route all traffic from one subnet to another along a single "best" path. In a Clos fabric network, the path is randomly chosen from the set of equal paths, distributing the load evenly across them all.

Hosts are connected to TORs, which are connected to two or more fabric

switches, as shown in [Figure 23.3](#). The TORs and fabric switches are grouped so that all TORs in one group, or pod, are each connected to all the fabric switches in that server pod. For example, in [Figure 23.3](#), each of the m TORs ($T1$ to Tm) is connected to all n fabric switches ($F1$ to Fn)—this group of TORs and fabric switches is a single server pod. To increase the network capacity from each rack, you can increase the number of fabric switches in the server pod. It is possible to arrange for enough bandwidth to support all devices transmitting at full speed simultaneously.

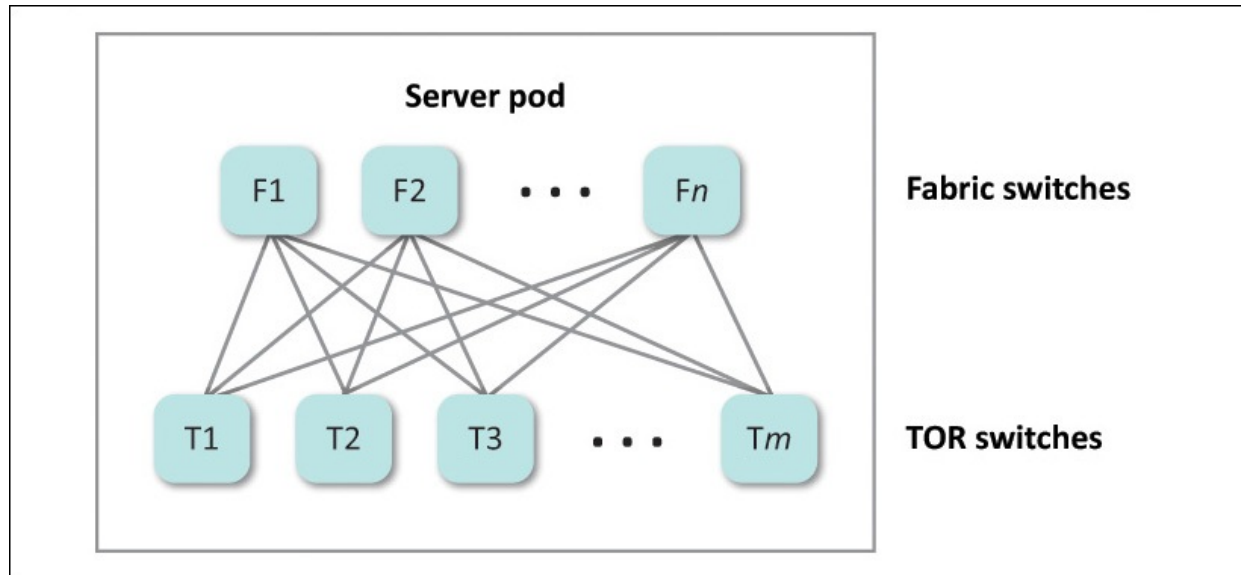


Figure 23.3: A fabric pod

The pods themselves can be connected in a Clos topology. Thus, the system is nearly infinitely scalable in terms of both internal bandwidth and total size. Some excellent descriptions of Clos fabrics have been published. Google’s implementation is described in “Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google’s Datacenter Network” by Singh *et al.* ([2015](#)), and Facebook’s topology is described in “Introducing Data Center Fabric, the Next-Generation Facebook Data Center Network” by Andreyev ([2014](#)).

23.5.2 Logical Design

Servers and services within a datacenter should never rely on any machines outside the datacenter, as discussed in [Section 14.2.4](#). In support of that policy, datacenter subnets should be entirely separate from non-datacenter subnets. Moreover, a datacenter subnet should never be configured outside the datacenter. Having separate subnets simplifies automated configuration checking to ensure

that this standard is followed.

Datacenter subnets are usually allocated based on a combination of security zone and the functional usage of the interfaces on that subnet. For example, a site may have a set of subnets for out-of-band (OOB) management interfaces of internal (green zone) servers, and a separate subnet or two for the OOB interfaces of Internet facing demilitarized zone (DMZ)—amber zone—servers. Other functional groupings include different subnets for high-speed data backup interfaces, for network device management interfaces, and for the primary interfaces over which the services are provided, again with a split between different security zones.

Most companies use different subnets to separate devices that are in different security zones. Subnets for different security zones should be on separate physical devices, rather than on different VLANs on the same device. With separate physical devices, you can be sure that traffic passes between the networks using only the access controlled paths that you have specifically defined and can monitor. In contrast, when the same physical hardware is used, a bug in the switch software could be used by an attacker to circumvent those controls. Different security zones using separate physical devices may be the exception to the “any VLAN anywhere” rule, unless a security zone is large enough to merit a separate TOR, or pair of TORs, in each rack.

Case Study: Stack Overflow’s Datacenter VLANs

Stack Overflow’s datacenter VLANs are standardized. There is a primary network, a secure network, and a DMZ network. Each VLAN’s subnet is the same in each datacenter except for the second octet that indicates the location. For example, 10.1.0.0/16 is one datacenter and 10.2.0.0/16 is another.

The VLANs are provided by a pair of redundant unified fabric switches. All servers have one connection to each unified fabric. As a result, each fabric can be taken down for maintenance without service disruption.

In the event that both fabrics are down, an additional network that uses different equipment exists. This network uses simple, older technologies that are less fancy. It does not have any external dependencies. The fabric devices and other network elements connect their management NICs to this network. In the event the main fabric switches go down, SAs can use the management network to connect to network devices to perform

diagnosis and repairs.

23.6 WAN Strategies

Offices and datacenters are connected via long-distance network technologies. This interconnection between locations is called a wide area network (WAN). There are two primary aspects to this strategy: topology and technology. The topology is the network diagram—if you were to draw the WAN on a piece of paper, with each location represented as a box, what would the network look like? The technology is the equipment and protocols used to make those connections.

In addition, you should consider the level of redundancy—how many direct links to other sites that each location has. Redundancy and topology are closely related. The technology used can also influence the topology.

If a site has a single way to connect to other sites, it has no redundancy—that connection is a single point of failure. To be properly redundant, any additional connections should be completely independent of each other. They should use separate network equipment, separate paths into the building, and separate vendors. For point-to-point connections, ideally they should terminate at different sites or datacenters, but at a minimum they should enter the other location using separate paths, and terminate on separate network equipment.

Demarcation Points

A demarcation point is the boundary between your organization and a utility company, such as a telephone company or network provider. The demarcation point can be a fiber cabinet, a set of punch-down blocks, a board in a rack, a piece of network hardware, or a small plastic box (often called a brick or biscuit) on the wall, with a jack or socket for plugging in a cable. The telephone company is responsible only for the wiring up to its demarcation point (demarc). If you have a fault with a line, you need to be able to point out where the correct demarc is so that the service engineer doesn't end up trying to test and fix another operational line. You also need to be able to test your cabling from the demarc all the way back to the network equipment. The main thing to know about your demarcation points is where they are. Make sure that they are properly labeled.

23.6.1 Topology

Common topologies include star, multi-star, ring, and cloud.

Star Topology

In the star topology, there is one central location, and every other site connects directly to that location, as shown in [Figure 23.4](#). The other sites may be dual-connected to that location. For companies with a single datacenter or computer room, the datacenter will be the center of the star. If the datacenter fails, the company's whole network infrastructure and services will go down.

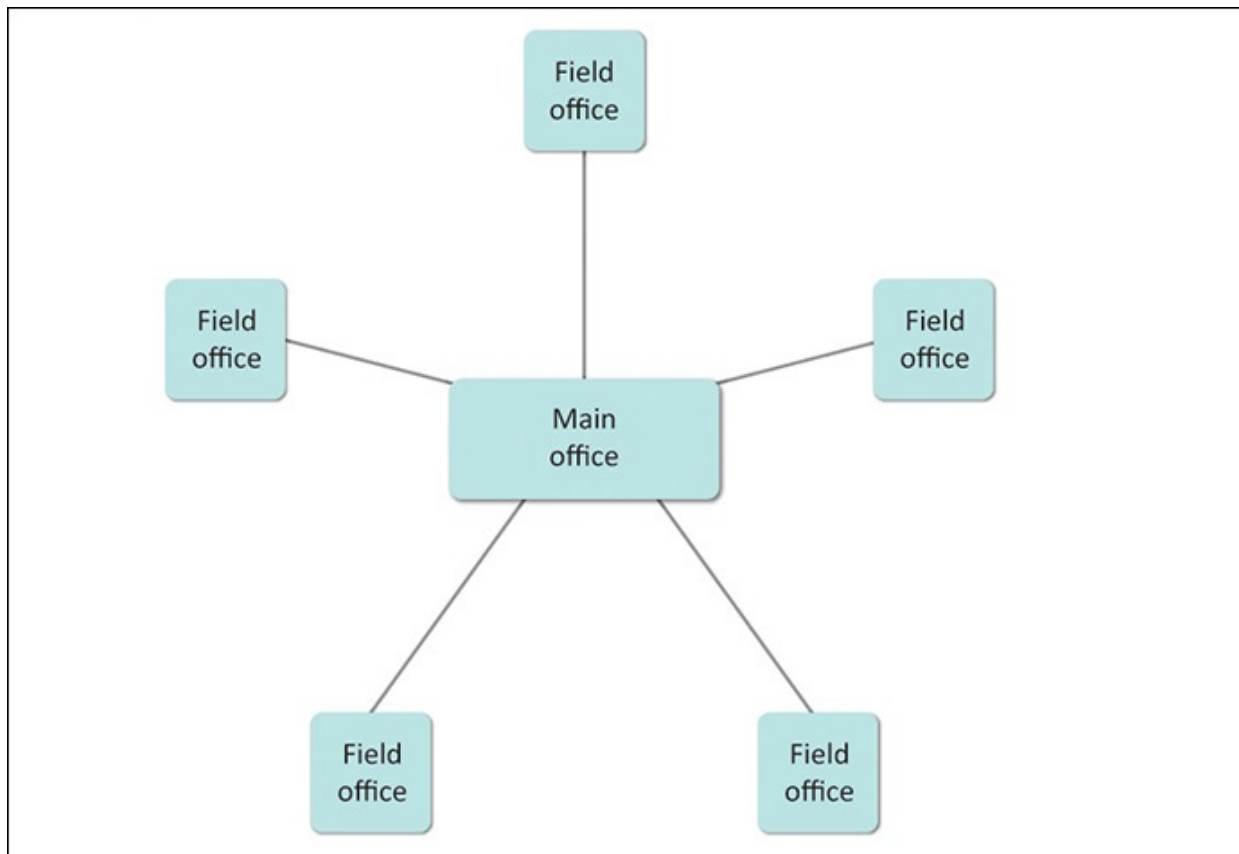


Figure 23.4: A wide area network with a star topology

A more resilient variation on the star topology is the **dual star topology**, where the company has an additional datacenter or disaster recovery location. The primary and secondary locations are dual-connected to each other, and each site is connected to both the primary and secondary locations. No single failure can take the company down, or lead to any of the sites being isolated from the corporate network.

Multi-star Topology

Large companies may choose a multi-star or hub topology. In this design, each region has a star, or dual-star, topology, with each site connecting to the regional hub(s). The hubs are then connected in a star or other topology. Hubs are selected based on regional groupings of sites, often one or two per country or continent. It is common to have each datacenter also serve as a hub, as facilities and personnel can be shared. [Figure 23.5](#) shows a hub arrangement with each office connected to two hubs for redundancy. (Note that “datacenter” is often abbreviated to “DC,” as it is in this figure.)

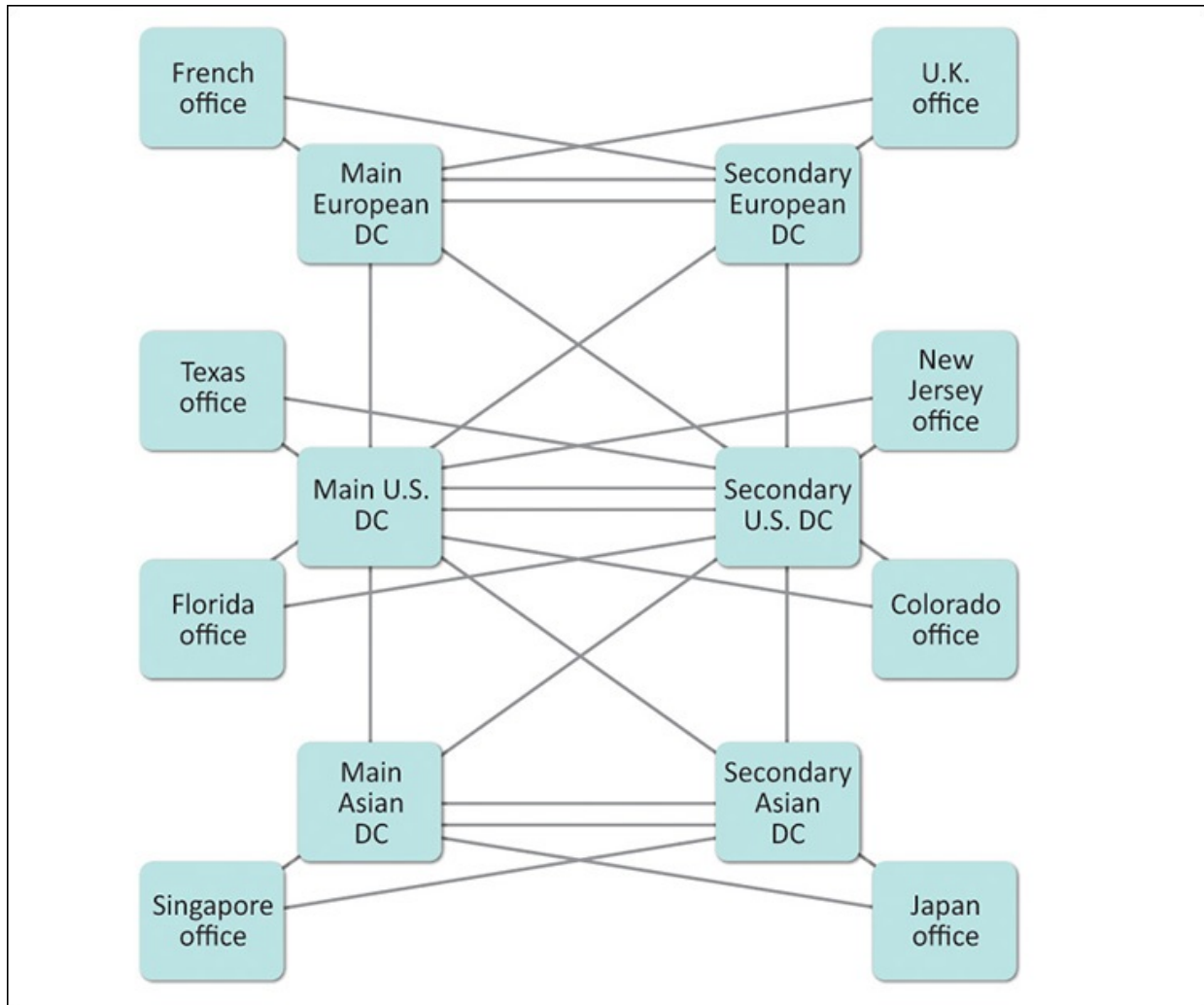


Figure 23.5: A redundant multi-star topology for a WAN

Ring Topology

Companies with several locations that are not too geographically distant may elect to use a ring topology, rather than a star or dual-star design. In a ring topology, each site connects to two others, as depicted in [Figure 23.6](#). The main disadvantage of a ring topology is that adding another site while maintaining a

ring topology is quite disruptive.

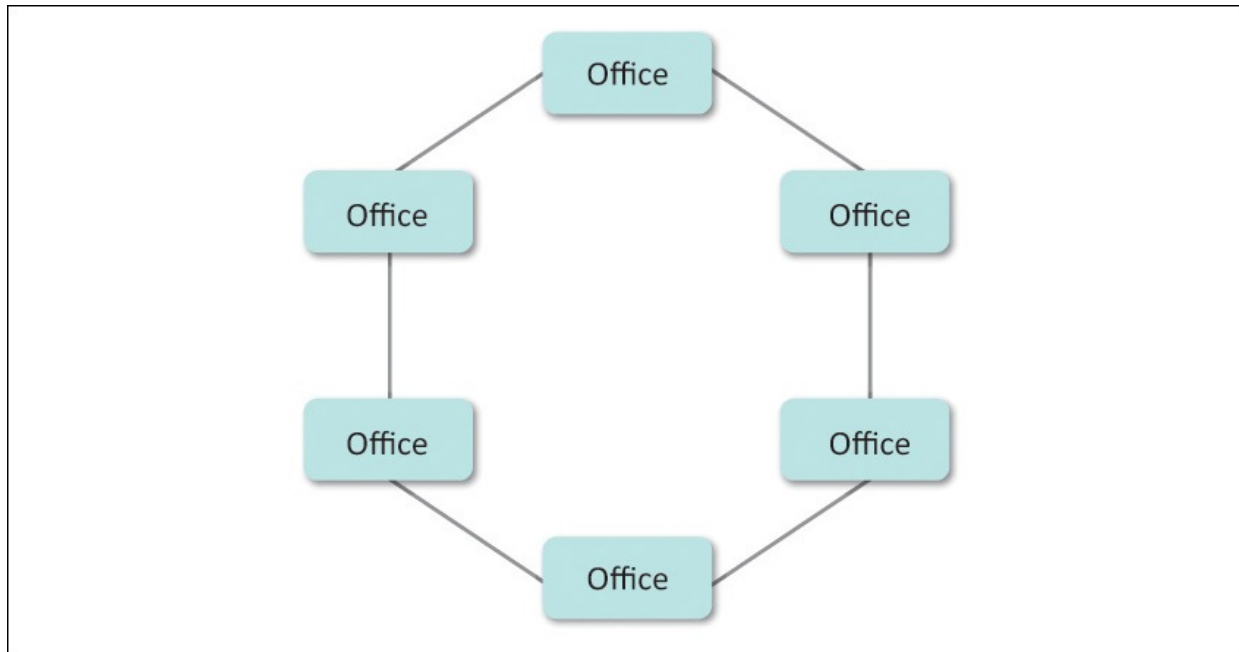


Figure 23.6: A ring topology, with each site connected to two others

Cloud Topology

Another common choice for companies with several locations that are not too geographically diverse is a cloud topology. In this topology each site connects to a provider's network. You don't know what the provider's topology is, but packets you send in come out the desired location. While the details are cloudy (that's literally where the name comes from), you are actually one of many customers who are sharing a large network infrastructure maintained by the service provider. What you see is a set of technology-specific virtual circuits, as if you've paid to have a private WAN, but it is really an overlay on top of the vendor's network.

The benefit of this topology is that someone else takes care of the details. It is also cost-effective because costs are amortized over many customers. However, if all customers want to use their full network bandwidth at the same time, traffic can be congested.

23.6.2 Technology

The technology one uses to implement a topology generally falls into one of two categories: dedicated lines or virtual private networks (VPNs) over the Internet.

With a dedicated line, you lease a network connection between two points. In the old days the telecom provider provided a circuit—an actual wire—between

the two points. Today these circuits are emulated over their larger infrastructure. A VPN sends packets over the Internet in a way that looks like a point-to-point link to your equipment. Packets are encrypted so that others on the Internet cannot see the contents of your traffic.

Dedicated lines provide guaranteed bandwidth and latency, which are stipulated in the contract with the network vendor. Bandwidth and latency can both be measured to ensure that the contractual obligations are met. Dedicated lines are also reasonably immune to denial of service (DoS) attacks, unless there is a significant attack against the vendor's entire network infrastructure. Companies can work with the network vendor to understand the physical path that the line takes, to ensure that redundant connections take separate paths. Separate paths are required so that a construction worker with a backhoe cannot accidentally dig up both lines with one swipe. The cost of dedicated lines typically increases with the distance between the two sites. This option can be quite expensive, but it is easy to understand and debug, and the bandwidth and latency guarantees are attractive to many companies. A minor variation on this approach is to increase security and privacy by encrypting all traffic that traverses these lines.

VPNs across the Internet are typically a more cost-effective approach. However, bandwidth and latency are not guaranteed and can be highly variable. For a site with a high-speed Internet connection, a VPN may provide more bandwidth than the company could reasonably afford with a leased line. Nevertheless, a company's Internet infrastructure is also subject to DoS attacks, which can consume all available bandwidth, effectively cutting off the VPN connection to the rest of the company and isolating the site.

VPN problems can be hard to find and diagnose, and next to impossible to resolve, as they may be caused by problems with Internet infrastructure at any of the ISPs between the two endpoints. For example, a large fiber cut at an intermediate ISP can cause significant congestion and packet loss on other intermediate links, but because your company is not a customer of that ISP, you will not be informed. To provide assistance in such cases, some services specialize in monitoring Internet-facing infrastructure from many points around the Internet. They can help to identify these problems, and get your traffic rerouted around the congested links. If you choose to use the VPN approach, we highly recommend that you subscribe to such a service.

An additional consideration is that the VPN approach requires that every site has a connection to the Internet, which should be appropriately secured and monitored. Therefore this approach may incur additional overhead for the

security team. Also, true redundancy in the VPN approach requires each site to have two separate Internet connections to two separate ISPs, with automatic failover between them. This requirement adds additional routing complexity at each site, and usually requires high-end network gear, which can reduce or eliminate the cost savings relative to the dedicated line approach.

A hybrid approach is also possible, where each site has one dedicated line to another site, and uses a VPN connection for redundancy. This approach still requires Internet connectivity and security in each site, but the routing is simpler. Any approach using VPNs needs to be considered in conjunction with the company's Internet strategy, which is discussed further in [Section 23.8](#).

Case Study: Dual-Connected Offices

A large international company with sales offices around the world established a standard for WAN connections as part of a network upgrade project. Previously the network had grown organically, such that each office had different network hardware and technologies. Some sites had network connections that were more reliable than others, some of which were more manageable than others.

Under the new standard, each sales office would have two dedicated lines, one to each of the two nearest regional hubs that were already in place. Each connection would use a different telecom provider, to reduce the single points of failure. This also permitted the company to negotiate with two global providers rather than hundreds, as had been the case before.

In the event that two providers did not service the location, one of the network connections would be a ISP connection and a VPN would be used as the second link.

A network equipment vendor was identified that had appropriate products. The same device could be used everywhere, simply by changing an interface and connector depending on the local telecom standards. The vendor also offered on-site parts replacement, which meant that the network team would not have to travel as much, once the new equipment was deployed. One router was used for each connection, meaning that when one died, the other could be used to connect to and configure the replacement hardware after it was installed by the vendor's local people. As with the case study in [Section 17.2](#), the configurations were prototyped in a lab before the standard was set.

Each office was upgraded as part of a massive deployment project. Project managers coordinated the network providers delivering new circuits, the equipment vendors delivering the hardware, and flight schedules of the network team as they made what was hoped to be their last visit to each office.

As the new equipment was deployed, the network became more stable, reliable, and less expensive to manage.

23.7 Routing

To get traffic from one subnet to another, it needs to be routed, which is layer 3 in the OSI model. Routing is a fundamental component of a network. There are many books on routing and the various routing protocols that can be used. This section does not attempt to cover such a broad topic, beyond giving a high-level overview and a brief introduction to some of the most common routing protocols.

Routing can be complex, particularly for large sites. Your aim should be to keep the routing as simple and easy to understand as possible. One of the primary ways to do so is to develop an IP addressing scheme that is easy to understand and supports you in simplifying routing and enabling route summarization.

There are three primary strategies used to route traffic: static configurations, routing protocols, and exterior gateway protocols.

23.7.1 Static Routing

Static routing takes advantage of the fact that a router always knows how to talk directly to a subnet that is directly connected to it. In other words, if you have only one router, there is no need for a fancy routing protocol since it knows how to route packets between its subnets. If there are two routers, with some subnets connected to router A and others connected to router B, one can manually enter static routes on router A for each of router B's subnets, and vice versa.

While this is very simple, it is not very flexible. As the number of routers increases, the manual configuration becomes untenable. In the case of an outage, static routes have to be manually changed to route around the problem until the outage is resolved. Therefore this kind of configuration is useful only for very small sites.

23.7.2 Interior Routing Protocol

Another strategy is to use a routing protocol, such as OSPF and EIGRP, that enables each router to communicate with its neighboring routers to advertise which subnets are connected to it. Each router then uses this information to update its routing table. If a subnet is down or disconnected, the routing protocol informs all other routers that it is no longer available.

In addition, each router tells its neighbors about other subnets it has learned about from other routers. It adds a weight or distance metric to the routes that it receives from other routers, so that its neighbors can figure out the best path to get to those networks. These protocols can scale to many hundreds of routers.

23.7.3 Exterior Gateway Protocol

This routing protocol communicates between autonomous systems. Imagine your company is divided into two divisions, each with its own network. Each uses OSPF to communicate route information among its routers. Each of those is an autonomous system. To route between them, an exterior gateway protocol is used. The most common EGP is Exterior Border Gateway Protocol (EBGP).

An EGP such as EBGP can be used when different parts of a company have their own network teams. Each uses an interior routing protocol for its part of the network but they all use EBGP where their networks meet. This permits multiple administrative domains to work together. EBGP is also what is used to route between ISPs on the Internet.

23.8 Internet Access

Part of your network architecture involves defining how and where it will connect to the Internet. At a high level, there are a few different approaches you can take. The first is to have a single connection to the Internet for the whole company. The second is to have a limited set of connections, perhaps one or two per geographic region, typically out of the regional datacenters. The third approach is to have an Internet connection at every site. As discussed in [Section 23.6.2](#), the Internet strategy also influences the available options for WAN connectivity.

There are two aspects to Internet connectivity, and some companies also separate these and have different approaches for each. The first aspect is outbound Internet access: how people in the company access services on the Internet. The second aspect is inbound access: how the company makes services available on the Internet. Some companies choose to have outbound access at

many locations, but inbound access at only one or a few sites. Inbound and outbound Internet access are two quite separate things, with completely different use cases, so it is perfectly reasonable to choose different approaches for each of them.

23.8.1 Outbound Connectivity

The Internet is a source of ever-changing threats and attacks. Not surprisingly, then, many companies choose to limit their exposure to a single site. This approach makes it easier to closely monitor those threats and attacks, and to put mitigation measures in place. Companies can focus their security efforts on a small set of machines, and rely on everything else being protected by strong firewalls that prevent direct access to and from the Internet. However, this one location is a single point of failure, and a problem there can cut the whole company off from the Internet. In addition, for a global company, Internet traffic for end users in other regions has to cross the internal WAN, which will have high latency due to the speed of light, and may be congested due to other internal traffic. In addition, if attackers launch a denial of service attack against that single connection to the Internet, it is likely that they can take the company offline. With multiple connections to the Internet, it is harder for an attacker to successfully do so, particularly if inbound and outbound access are completely separated.

The end-user experience can be improved by having multiple Internet connections, usually one per regional hub. Security is still manageable, because the number of Internet access locations is still strictly limited. It is not too hard to make sure that the machines in these few locations are kept up-to-date with security patches, malware prevention, and so on. It is also easy to regularly scan for vulnerabilities, and to ensure firewall configurations are secure. In addition, for companies that are concerned about data leakage, there are still only a few places to monitor and perform scans on outbound content. Also, a company is more protected against denial of service attacks, as an attacker now needs to attack more locations to take the company offline. However, WAN connectivity for remote sites via VPN is not possible, because the remote sites do not have Internet connectivity.

With an Internet connection at every site, it is possible to connect all sites using VPNs across the Internet. However, maintaining a high level of security at all these locations will be very challenging, unless configuration management and patching are automated and driven by templates. Even so, if a typical employee at the company accesses most of the applications that he or she needs

over the Internet, because they are SaaS solutions provided by other companies, then this approach makes the most sense.

23.8.2 Inbound Connectivity

Inbound access to the company's own services offers the greatest attack surface, particularly for distributed denial of service (DDoS) attacks. One approach that can be used for inbound access is to not provide those services from the company's own premises. These services can be placed in a service provider's datacenter, with high-speed, highly redundant Internet access made available either by purchasing IaaS or PaaS services from a cloud provider or by renting space for the company's own equipment in a service provider's datacenters. The best defense against a DDoS attack is to have more bandwidth than your attacker.

The other approaches commonly used for inbound connectivity are to provide services from one central location, or from a few regional locations. For companies with a global customer base, it is best to provide services in-region for their customers, or they will lose customers because the site is too slow. Global traffic managers ensure that traffic is directed to the closest available servers.

It is unusual for a company to have inbound Internet access at all locations. Most regional offices will not have a datacenter, and services that are provided to customers over the Internet should be provisioned on servers that are in a proper datacenter. However, that does not mean that it does not make sense for these offices to have outbound Internet access, so long as it is properly managed and secured.

23.9 Corporate Standards

An essential part of the network architecture is to define and document standards for offices, datacenters, and other types of facilities. For example, your company may have office locations that can be categorized as small, medium, and large. It may have large regional hub and disaster recovery datacenters, medium-size datacenters for some countries, and perhaps small datacenters with minimal infrastructure in other countries. Depending on the type of business, there may be different sized retail stores or manufacturing facilities.

Standard designs promote cookie-cutter solutions that are identical at each site. This approach enables you to use templates to define network configurations, from logical and physical designs all the way down to the device

configurations. It enables cost savings through automation, and increases reliability and improves support through consistency. It enables you to treat network devices like cattle, rather than pets, as described in [Chapter 3, “Pets and Cattle.”](#)

The first step is to define the categories for which you need to define the design. For example, you might have large and small datacenters, plus small, medium, and large office locations. For each type of location, you need to define logical and physical designs. The fewer of these categories there are, the easier it is to learn, understand, and support the site.

Many companies have grown organically, without a particular set of standard designs. Even in these environments, the network team should define what each site should look like, and look for opportunities to make the sites standard. For example, when the company builds new sites, it deploys the new standard. When the company rents an extra floor in an existing building, the build-out follows the new standard. When the company renovates an office area, it should redo the network at the same time. The network team should collaborate with the facilities team so that the network component is properly included in the plan.

For datacenters, look for opportunities when new zones need to be deployed, additional capacity needs to be added, or the network gear needs a hardware refresh. Deploy new servers onto networks that you have created according to the new standard, and as servers are retired and refreshed, you will slowly converge on the new standard.

23.9.1 Logical Design

The logical design defines the high-level design from the point of view of the end users, the application teams, and the support team. Using a logical design diagram, it is possible to trace the path that traffic takes from an end user’s workstation to an application server in a corporate datacenter, or an extranet partner’s site, or a site on the Internet, in terms of which zones it will pass through.

The logical design does not specify locations, quantities or types of hardware, or the number of physical connections or subnets. It does not define IP address ranges. It is generic, and any number of sites can, and should, be built using the same logical design. [Figures 23.4](#) and [23.5](#) are examples of logical designs for a WAN.

The logical design for a site defines the network zones that will be implemented in that size and type of site. For example, the logical design for a

large office location might specify that the site has an employee zone, a customer zone, a BYOD employee zone, and a guest zone.

The large datacenter logical design may specify a number of different functional zones, as described in [Section 23.5](#). It may specify that the site has inbound and outbound Internet connectivity zones, and separate extranet inbound, extranet outbound, and extranet collaboration zones for direct connections to third parties. It may also specify various security zones that will be implemented in that datacenter. For example, in addition to partner connectivity zones and Internet connectivity zones, there may be high-security data protection zones for highly confidential customer data. The small datacenter design may have the same set of functional zones, but fewer security zones, and perhaps no Internet and no extranet connectivity zones.

Start with the largest size for a given type of location, and create a logical design that is easy to understand. The logical design for smaller sites of that type should normally consist of subtracting specific elements from the the largest one.

The logical design gets updated when new zones are introduced. For example, when a company that previously had no dedicated extranet connectivity signs a deal that results in the first extranet connection, the company needs to design the extranet connectivity, and add extranet zones to the relevant logical design diagrams. Such changes should be quite rare.

Each logical design should have a version number, and each version of a logical design should be readily available for as long as that logical design is in use somewhere in the company. The company should also define a strategy for dealing with a change in logical design. Should the new logical design be propagated out to all existing sites over some period of time? Or should a site be updated only when a requirement for the new zone arises, or when there is a hardware refresh?

Also, there may be additions to the set of logical designs when the company introduces a new site type or size. For example, if a company that previously had only small and large datacenters introduces a medium-size datacenter, which perhaps has Internet connectivity zones but no extranet, then the company would need to add a medium-size datacenter to its list of logical designs.

23.9.2 Physical Design

The physical design maps the logical design onto topology, device types, and physical connectivity, including the number of connections between devices. It includes routers, switches, and firewalls, as well as connections from the site to

other locations, such as datacenters, the Internet, and partner sites. The physical design is still generic, and shows where parts of the design can be extended or scaled by adding devices and connections, thereby replicating a portion of the infrastructure. The physical design is cloned for all sites of the same size and type.

The physical design does not specify equipment vendors and part numbers, but rather specifies generic components, such as “large fabric router,” “TOR,” “small firewall,” “large web proxy,” and so on. Vendors update their product lines on a regular basis, and companies should reevaluate their choice of vendor and products periodically. The physical design should not need to be updated just because the vendor has refreshed its device offerings, or because the company has switched vendor or product lines.

The physical design is updated when there is a change in topology, such as a change from using core routers on a high-speed backbone in the datacenter to using a switching fabric. It also needs to be updated when there is a change in the logical design. As with the logical design, each physical design should have a version number. It should also specify on which logical design it is based.

Equipment List

The company should maintain a standard equipment list that maps the generic component names in the physical design to a list of part numbers and corresponding vendors. It should specify the default configuration that should be purchased for each, as well as the scaling options. The combination of these two results in an easy-to-produce bill of materials for any new deployment.

Update the list whenever model numbers or vendors change. The list should also have a version number and a date. While it should be relatively stable, an equipment list that hasn’t been updated in a while gives an indication that it may be time to do another round of hardware evaluations, as the technology will have moved on.

Implementation Design

Each individual site has an implementation design that shows the specific hardware used for each device in the physical design, along with its name. An important component of the implementation design is a diagram of the physical network. Such a diagram should show the connections between devices, including the speed. For wide area connections, latency is included on the network diagram. The implementation design also includes the IP addressing scheme for that site.

The implementation design is updated whenever there is a change in the equipment, devices or connections are added or retired, and so on. It is a living document, and should be easy to update. It should be dated, so that it is clear when the last update was made, and should also have a version number. The implementation design for each site also should specify the version numbers of the logical designs, physical designs, and equipment list that were used to create that site.

23.10 Software-Defined Networks

Software-defined networks (SDN) take a different approach than traditional routing to control the flow of traffic between devices on the network. An SDN makes routing as programmable as any other application. It enables applications to specify their needs (such as low latency or a certain amount of bandwidth) to a network controller. The network controller has an overview of the network. It can dynamically configure the network devices to create a specific path through the network for that application's traffic. An SDN also enables network administrators to specify how particular types of network traffic flows should be handled within their network.

The SDN operates at a macro level, treating the network as a unit, and optimizing traffic paths to make best use of the network, based on the network performance characteristics that the applications need. An SDN also enables network administrators to make the best use of their network, redirecting traffic away from heavily used WAN links to more lightly used links. Doing so realizes cost savings for the company, delaying the need to upgrade an expensive WAN link by months or even years. It can also improve the end users' experience by directing traffic flows based on the needs of the applications.

With traditional routing, if client A and client B are on the same office network and are communicating with server A and server B, respectively, and these servers are on the same datacenter network, then their traffic will follow the same route. Say that client A is using app A, which is sensitive to latency, and client B is performing a large data transfer using app B. With traditional routing, the data transfer may cause queueing and packet loss for app A, resulting in a poor end-user experience. With SDN, if both applications are SDN-enabled, the SDN controller can decide to send the data transfer traffic over lightly loaded links, even if it needs to go halfway around the world, so as to ensure that app A can use the more direct low-latency links without contention.

An overview of the potential new applications possible with SDN can be

found in the article “OpenFlow: A Radical New Idea in Networking” ([Limoncelli 2012](#)).

23.11 IPv6

Vint Cerf, one of the original designers of the Internet, is fond of saying we need to stop running the experimental version of the Internet (IPv4) and move to the production version of the Internet, IPv6.

The number of devices that are IP capable is increasing rapidly. Toilets, light bulbs, televisions, refrigerators, cars, sensors, and all sorts of other everyday devices are now IP capable. Even before the Internet of Things (IoT) boom, IPv4 address space was in extremely short supply. We are now working around the fact that there are more devices than IPv4 addresses with layers of network address translation (NAT). But, as our company’s subject-matter experts on IT, SAs also need to make sure that our companies start to prepare for deploying IPv6.

23.11.1 The Need for IPv6

IPv4 has only about 3.7 billion usable IP addresses. This may sound like a lot, but the Internet ran out of IP addresses to allocate in 2015. Actually, it’s not surprising if you think about it. The number 3.7 billion means there is an average of one IPv4 address for every 2 people living on this planet. Christina’s family currently averages just over 5 IP-enabled devices per person at home, compared to 2 per person 10 years ago. And naturally we have many more IP-enabled devices at work. On top of that, the Internet is now reaching far more people than it used to. When you look at it that way, it is amazing that we have managed to stretch those paltry 3.7 billion IPv4 addresses this far. Fortunately, since 1993 the IETF has been working on IPv6, which has 3.4×10^{38} (340 undecillion) addresses.

Also, IPv4 does not scale very well. Broadcasts limit the size of a LAN, and there are other problems. IPv6 replaces broadcasts with multicast, and fixes other problems as well. Sadly, IPv6 is not upward compatible with IPv4. The upgrade path involves running both IPv4 and IPv6 in parallel for decades until IPv4 can be removed entirely.

23.11.2 Deploying IPv6

IPv6 is not difficult to understand or deploy. At sites that have deployed IPv6, users don’t even notice. They type hostnames, not IP addresses. For SAs, there is

a learning curve initially but it isn't that different. The biggest change is often that logs and databases that store IPv4 addresses now need to store IPv6 addresses.

When approaching management about rolling out IPv6, promote its benefits to the business. People are publishing books and articles on their experiences and advice for deploying IPv6. Read up on this subject, and become familiar with some successful strategies that others have used. Pay particular attention to the business benefits that others have successfully touted to get the project funded, and be ready with hard data and solid high-level explanations as to why this change is good for the company. One useful resource is the article “Successful Strategies for IPv6 Rollouts: Really!” by Limoncelli & Cerf ([2011](#)).

As when rolling out any new service or upgrading an existing one, start slowly. Do not try to do everything at once. There are a number of strategies that you can use for a gradual roll-out of IPv6.

Start with the Outside and Move Inward

A good first step is to start with your Internet-facing infrastructure, and move inward. The path to getting your IPv6 project approved is to show its business benefits, and to have a small, contained project that doesn't try to do everything at once—and this approach fits the bill.

The key business benefit is that you offer better service to the growing number of people who are using devices that are IPv6 only, particularly mobile devices. If your company does not offer native IPv6 services, those customers need to go through their ISP's 6to4 gateway. Traffic from a lot of different devices in diverse locations is funneled through these gateways to get to IPv4-only services. They are often slow, which means that your service seems slow. In addition, all these devices appear to your applications as if they are in a few central locations—the locations of the 6to4 gateways. Since you no longer have the true geographic location of the client device, that throws off geo-targeted advertising, which could lose your company money.

Choose an application, such as a web site, that is behind a load balancer, and implement IPv6 from your ISP to the load balancer. The load balancer should be capable of communicating with clients using IPv6 on the frontend and communicating with the web servers using IPv4 on the backend. This seemingly simple project touches the network devices, the connection to your ISP, any firewalls between the ISP and the load balancer, the load balancers, and DNS records. It also involves updating the test environment, so that you can properly perform QA tests of the changes, run all the standard (and ideally automated)

tests against the new setup, and add in some new ones to explicitly repeat the tests using IPv6. While this is a relatively small, self-contained, and low-risk project, it is a good way to demonstrate a successful IPv6 deployment. It also paves the way for more IPv6 deployments, as it deploys IPv6 to key parts of the infrastructure. Make sure that you measure what percentage of the traffic you receive is IPv6, and graph it. As a next project, you can perform a slow roll-out of IPv6 to the web servers themselves, so that the load balancer can communicate to them over IPv6 as well.

Start with Less Critical Networks

Another obvious candidate for an early IPv6 deployment is your guest WiFi network. The mobile devices that will use it likely use IPv6 on a regular basis on their mobile providers' networks. The guest WiFi network is almost certainly not mission critical, and it will give you the opportunity to gain additional experience with very little risk. This deployment can then expand out to cover BYOD WiFi, guest and BYOD wired, corporate WiFi, and eventually corporate wired office networks.

Use Test Labs and Slow Roll-Out Strategies

Deploying IPv6 isn't hard, but like any change, it should be tested and rolled out in a controlled manner. Build IPv6 into all your plans wherever you see an opportunity that doesn't require a massive investment. Convert an area of your lab to have both IPv4 and IPv6. Make it available to all engineers, architects, and developers at your company for testing. As you update your automated test suites, include tests that run in the IPv6 lab environment and tests that explicitly use IPv6. Be ready to deploy IPv6 in other labs as people request it; deploy it to the SAs' networks, then perhaps to the developers' networks, and then to the rest of the company. When applications are ready to offer services over IPv6, they just need to add the relevant DNS entries, and they are good to go.

23.12 Summary

The OSI model is useful for describing the different aspects of a network. As a rough generalization, the network team is concerned with layers 1 to 3, SAs are concerned with layers 3 to 7, and application teams are concerned with layers 6 and 7. The physical layout of a network is layer 1, which can be completely different from the logical layout, which is layer 3. For most people the logical layout is the most relevant. It describes the layer 3 devices, such as routers and firewalls, that network traffic passes through on its way from point A to point B.

Networking is a vast topic, with many books dedicated to it. This chapter gave you an overview of the physical infrastructure and logical design strategies for wired and wireless office networks as well as datacenter networks. We also briefly described WAN design, routing, and strategies for Internet access.

For all aspects of IT, well-defined and documented standards need to be the foundation of everything you do—and network standards are no exception. Use standard physical and logical designs, and standard equipment. Doing so makes the network substantially easier to understand and support.

Finally, we encourage you to look to the future. You need to be planning for IPv6 now. It's not as hard as you may think, and starting with IPv6 projects that are small and limited in scope is a good way to get management buy-in and to build confidence—your own and your management's. We also included a brief introduction to SDN. As it is a potential cost-saving technology, particularly for global companies, you need to be aware of the benefits. Make sure, however, that you have sufficient visibility into your network so that it won't become incomprehensible and impossible to debug with SDN.

Exercises

1. Draw a physical network diagram for your organization, or its largest office.
2. Draw a logical network diagram for your organization, or its largest office.
3. How much do the physical and logical diagrams differ, and why?
4. If you were responsible for the network design in a new office building, what would you do? Describe the physical infrastructure and logical designs that you would use.
5. Which topology does your WAN use? Draw physical and logical maps for the WAN. How much do they differ, and why?
6. How much time does your team spend visiting office wiring closets each month? Could you decrease that? If so, how, and what would it cost? Explain why it is, or is not, worth investing in reducing wiring closet visits. If you take into account the productivity of other employees, how does that change the equation?
7. Does your office have guest WiFi? If so, describe what a guest must do to get onto the network. Do you think that access method is reasonable for guest WiFi, and if not how would you improve it? If you do not have guest WiFi, design a guest WiFi solution.

- 8.** If you were to travel to another company office, how easy or hard would it be for you to get your various devices onto the corporate networks there, and why? Could you make it easier, and if so how? Justify your answer.
- 9.** What are the approaches that one can use for datacenter networks? Which approach is used in your datacenter?
- 10.** How long does it take to get a new machine onto the datacenter network, on average? What are the difficult corner cases? What could be improved and how?
- 11.** Where are your demarcation points? How are they documented and labeled?
- 12.** Which protocols do you use on your network, and what are the corresponding RFC numbers for those protocols? Are any of them proprietary protocols? If so, how could you avoid using those protocols?
- 13.** Devise a plan for deploying IPv6 at your company, including how you will get management buy-in for it.
- 14.** Why haven't you approached management yet about deploying IPv6? If you have but didn't succeed, why do you think that is, and how could you do better next time?

Chapter 24. Network Operations

All IT services rely on the network. If the network fails, so does everything that is connected to it, and everything that relies on those devices, and so on. A solid network design helps you to build a reliable network, but it gets you only so far. Once the network has been built, it needs to be operated, maintained, and run in a way that ensures it is highly reliable and available. This chapter describes best practices for monitoring, managing, and supporting an enterprise network, including recommendations for tools and organizational structures to facilitate supporting the network, particularly in large enterprises.

Increasing numbers and types of devices are becoming IP-enabled. Corporate telephones, building access, and security systems used to have separate wiring and non-IP protocols. Now no company would consider buying a solution that does not connect to its IP network. Other systems such as lighting, which were not previously part of any network except the building power grid, are going in the same direction. Smart lighting is convenient, saves energy, and can even be used to enhance sales in stores and productivity in offices. It will rapidly become the norm. Other things that we never previously contemplated connecting to the network will follow suit soon, and as they do, network reliability will become even more critical than it is today.

People often use the term “stability” when talking about highly reliable and available services. However, stability implies a lack of change, with the implication that if we don’t make changes to the network, or make changes only occasionally, the network will be more reliable. However, we advocate that you also increase the reliability of your network by using the small batches principle ([Chapter 2](#)), by undertaking rigorous testing, and by treating your network devices as cattle, rather than pets ([Chapters 3](#) and [4](#)).

24.1 Monitoring

Network operations starts with monitoring. Monitoring provides visibility that facilitates all other operational tasks.

If you are not monitoring your network, you aren’t providing a service. You won’t know that you have a problem until someone reports it, and you certainly won’t be able to prevent the problem from happening. Monitoring is a basic requirement of any service. These are the minimal items that should be monitored for a network:

- Network devices (routers and switches, not endpoints such as PCs):
 - Health (up/down status)
 - Internal components (blades, slaves, extensions)
 - Resource utilization (memory, CPU, and so on)
- For each WAN link, LAN trunk plus the individual links that make up a bonded set:
 - Health (up/down status)
 - Utilization (how much capacity is in use)
 - Error counts

Some of what you monitor is used to detect and identify service-impacting problems. However, to provide a highly reliable and available network, you must also detect and identify problems that are not yet service-impacting, and use the information to prevent them from becoming service-impacting, or to detect anomalies that may be indicative of another problem. In other words, you need to detect when a system is sick, so it can be fixed before it dies and negatively impacts users.

For example, having one out of a number of redundant links down does not cause a service impact, but detecting it and resolving it in a timely manner can prevent a service impact. Similarly, network utilization should be used to predict when more bandwidth will be required, so that it can be provisioned before lack of bandwidth causes service impact. This type of monitoring can also be used to detect sudden, unexpected increases or decreases in traffic, which may indicate a problem elsewhere.

24.2 Management

Network management is often conflated with monitoring, but they are really two entirely separate things. Monitoring observes the current state of the network and helps you to detect, predict, and prevent problems. Management means controlling device configuration and firmware versions.

Network management tools have lagged behind the corresponding tools that are readily available for managing workstations and servers, largely because network devices often run proprietary hardware and software that is not conducive to developing open source solutions. However, just as [Chapter 3](#), “[Pets and Cattle](#),” recommends treating servers as cattle, not pets, the same philosophy should be applied to how network devices are managed. In addition, we should strive to manage the network configuration as version-controlled

code.

24.2.1 Access and Audit Trail

Any changes that are made to network devices need to generate an audit trail that enables one to trace the change back to a person. Audit trails are a regulatory requirement for many businesses, and extremely useful for all businesses. The audit logs should be sent off the device to a centralized logging service, for event correlation and subsequent review as and when they are required.

As with any SA tasks, changing the configuration of network devices is not something that just anyone should be able to do. A mistake in the network configuration can have widespread impacts on the company. Write access to network devices should be restricted to a small team of people who work closely together, and who understand the site architecture and configuration standards. All access to network devices should be through individual accounts, rather than through role accounts, so that such access is logged to a particular user. Access provisioning and authentication should use centralized authorization and authentication databases. On-device authentication databases should be used for recovery credentials only.

24.2.2 Life Cycle

The life cycle of a device is the series of stages it goes through between being purchased and ultimately disposed of. The life cycle of network devices is considerably different than the one described for workstations in [Chapter 7](#), “[Workstation Software Life Cycle](#).”

The life-cycle phases for network devices can be described as in stock, assigned, installed, deploying, operational, servicing, decommissioning, and disposed. The life-cycle state should be tracked in the inventory system.

Depending on the type of device and the life-cycle phase that it is in, different levels of change control are appropriate. The guiding principle is that change management is needed when the change affects the production network. For example, new network equipment that is being deployed needs to be treated as production equipment as soon as it is participating in routing with production systems, or passing traffic across the production network.

Network equipment differs from servers in how it becomes live. A server can be fully deployed, with all the appropriate software installed, configured, and running, but it is not truly in production until production traffic is directed at it. Depending on the service, it may be brought into production by making a change

in a load balancer, or in DNS, so as to direct traffic to the server. With network equipment, as soon as it is fully configured and connected to the network, it is an active network component. This difference makes it harder to fully test network equipment in its configured state before it is part of the live network.

Your aim is to bring the device into production as efficiently as possible without causing an issue in the production environment. Therefore you should leave the change that brings the device into the production network until the end. Perform all the other changes in advance, so that when the device joins the network you can take your hands off it and know that it is set up exactly as you want. Understanding the life cycle of network devices helps you know how to achieve this goal. Let's look at the life-cycle phases in more detail:

- 1. In stock:** A device is in stock when it is in storage and not allocated to a project or a site.
- 2. Assigned:** A device moves to assigned state when it has been earmarked for a particular use.
- 3. Installed:** A device moves to installed state when it has been unboxed and racked. Up until this point, no change control has been necessary, except for any datacenter change control process that restricts when physical installations can take place.
- 4. Deploying:** In the deploying phase, the device is powered up, and connected to the management network and a console server. The correct version of software is installed, along with some initial configuration. As long as the network device is sufficiently isolated so that it cannot impact the production network—by injecting routes into the routing table, for example—then no change control is necessary during this phase. As the deploying state can involve a considerable amount of work, it is useful to impose the rule that a device in this state cannot be connected to the production network in a way that could cause an impact. With that rule and appropriate controls in place, even the strictest change review board will stipulate that changes to a device in the deploying state do not require formal change control.
- 5. Operational:** Moving a device from the deploying state into the operational state usually involves change control. Once it is in operational state, it is subject to standard change-management processes.
- 6. Servicing:** If a device fails in some way, it may be taken temporarily out of service and put into the servicing state. From here, it may return to the operational state, if it is put directly back into the service, or it may return

to stock, if it was replaced by another device.

7. Decommissioning: When a device moves into the decommissioning state, no new connections should be made to this device, and existing ones should start to migrate off. However, while it is still connected to the production network, it is subject to change control. Some sites have up to three decommissioning states, which are usually numbered.

(a) **Decomm 1:** The first decommissioning state indicates that connections and services are being migrated off the device.

(b) **Decomm 2:** The second state indicates that all connections and services have been migrated off.

(c) **Decomm 3:** The third state indicates that the device has been powered down. From there the device can move back to stock, or it can be disposed of.

8. Disposed: After a device has been decommissioned and disposed of, it is marked as disposed in inventory. Usually devices in the disposed state are visible in inventory for a fixed period of time, after which information about that device is archived.

Change-control processes are explained in [Chapter 32](#), “[Change Management](#).”

Life Cycle of a Building’s Network

A similar life-cycle strategy can be applied to the networks of entire buildings. For example, when opening a new office location, the entire network is in the deploying phase. There is no change control: There are no users and the people doing the installation need carte blanche to get their work done. Once the network is certified, it is ready for use. Users can move into the building and change control is required.

Decommissioning a building follows a similar, but reverse process.

24.2.3 Configuration Management

Configuration management for network devices lags behind what is being done with workstations and servers. Treating infrastructure as code (IaC), the topic of [Chapter 4](#), is becoming more popular, and we look forward to the possibilities that it brings to network management. Network devices that can’t be managed via IaC methods can be managed more efficiently by using templates and tracking changes.

Configurations should be based on standardized templates. As discussed in

[Section 23.9](#), you should have a limited number of standard network designs that are cloned to different locations. Each of those designs should correspond to a template that is used to build and maintain network devices in the corresponding locations, with the individual device customizations—such as specific IP addresses and networks—being applied at the end. Templates enable you to treat network devices more like cattle. Some significant advantages of network fabrics are that they treat all devices in the fabric as a unit, typically support the use of templates, and have a centralized configuration. However, version control of configuration files is often not built in.

Track changes by taking backups, or snapshots, of device configurations. Typically network devices have plaintext configuration files, whether or not they are part of a fabric. To gain some of the benefits of IaC, start by automatically retrieving these on a regular basis and storing them in a VCS. By doing so, you can see which changes were applied between retrievals, even if you cannot see who applied the change and why. You can also compare the configuration with the configuration that would be generated using the current version of the templates, to detect discrepancies.

Both commercial and open source products are available that collect and store network configurations for historical purposes. RANCID (Really Awesome New Cisco config Differ) has evolved to handle a wide variety of vendors and models. An interesting study of how such data can be used can be found in “An Analysis of Network Configuration Artifacts” ([Plonka & Tack 2009](#)).

Audit logs from network devices should be collected centrally for later use and analysis. Network devices should be configured to log configuration changes to an external log server. This server will log who made the change and provide a timestamp. Integrate this into the change-management systems to track which modifications are related to specific change requests.

24.2.4 Software Versions

Aside from configuration management, the other important aspect of network management is patching and upgrading the software, or firmware, that is running on the network devices. Traditionally, network devices are managed as pets. They are all upgraded separately, and they may all be running different versions of code. The proliferation of different versions of code makes it difficult to automate, and difficult to keep up-to-date with vulnerability patching and bug tracking. However, switched fabric networks are typically more like cattle. The fabric is upgraded as a unit, patched as a unit, and configured as a unit.

For non-fabric networks, it is important to keep the number of software versions to a minimum. Ideally, you should have a maximum of two versions for any given vendor. One version should be the strategic version, and the other should be the version that you are phasing out. New devices should be deployed with the strategic version, rather than with whatever version the vendor shipped on the device.

24.2.5 Deployment Process

Firmware, or software, releases for network devices should be treated in the same way as any other software. The new release should be subjected to an intensive testing process in the lab. The deployment to the lab equipment and the subsequent testing should both be as automated as possible. Start with a checklist of what to test and how, and add to it as you find new failure scenarios, perhaps due to problems in production.

Automate the checks one at a time until you have done as much as you can. Use virtualization to build a test network and to simulate device and link failures. The more automated your testing process is, the less time it will take to certify new releases, the more rigorous the testing can be, and the more confidence you can have in releases that pass the tests.

As with other software, start slowly when deploying new versions into production, and start with less critical devices that your least risk-averse customers rely on. As you build confidence in the new version, you can increase the rate at which you deploy it. Make sure that you have operational processes in place that ensure only tested and certified releases can be deployed in production. This may be a tool that controls the production software repository and permits only certified releases to be made available there. Alternatively, it may be a part of the change-management process that specifies the testing documentation and software deployment schedule have to be reviewed and signed off within the change tool as part of the change approval process.

The monitoring system should also track firmware versions on devices, so that you know how many devices are running each version, and which devices those are. This information can be linked to a list of approved versions, which also tags the versions as legacy, strategic, or unsupported. Doing so makes it simple to detect deviations from the standards, and to measure deployment rates.

24.3 Documentation

Network documentation has two main purposes. The first is to assist people in

troubleshooting problems, so that they can understand and resolve the problems as quickly as possible. The second is to explain what the network looks like, how it has been put together, and why.

There are four main types of documentation. The first is network design and implementation documentation, which describes in detail how the environment is built. The other three are DNS, the CMDB, and labels. These items are living documentation, meaning that they need to be updated on a daily basis as people perform their work. They are invaluable when troubleshooting a problem, but only insofar as they are up-to-date.

24.3.1 Network Design and Implementation

This form of documentation is used to bring new team members up to speed. It is also used when designing new services or applications so that the service or application owners do not make bad assumptions about the network, thereby creating performance problems for themselves or the infrastructure. Network design and implementation documentation consists of logical and physical network diagrams, with overlays showing bandwidth, typical latency, vendor, contact and support information for WAN links, services provided, IP addresses, BGP ASNs, OSPF areas, and so on for routers.

This type of documentation also includes more detailed solution design and implementation documents, templates, and configuration standards. These documents need to be actively kept up-to-date; otherwise, they will become useless. Updated documentation should be an explicit deliverable, with review and sign-off, in every project, however small.

Network Drawings with Overlays

Good network drawing tools allow the use of overlays. That is, you start with a basic network diagram, and then create overlays with additional information. For example, for a WAN diagram, you would start with the logical diagram showing how the sites are connected. Then create an overlay that shows the speed and latency of each connection. Another overlay would have the physical devices. Another would have device model numbers. Another would have the fully qualified domain names (FQDNs) and IP addresses. Another would have the routing information, such as BGP ASNs or OSPF areas. Another would have the link vendor names, support numbers, and circuit IDs. Additional overlays could be used for showing key services, such as DNS, DHCP, and AD domain controllers.

Overlays enable you to choose which information to export from the tool into an image that you incorporate into a document or a wiki. Each document has an image with just the information the particular audience needs, without extraneous details cluttering the image and making it confusing. However, all the information is stored in a single diagram (for example, in Microsoft Visio format), so that generating these custom images for each document does not require duplicating and then editing the network diagram. Duplication leads to network diagrams getting out of sync.

24.3.2 DNS

Troubleshooting documentation comes in many forms. The most basic is DNS. Make sure that every interface of every device has matching forward and reverse DNS entries, so that when you are debugging a problem and run a traceroute diagnostic, you can easily understand the path that the traffic is taking, and identify the devices along the way.

The naming standard should specify how to include the device name and interface name in the DNS entries, so that both can be seen and understood at a glance. The DNS naming scheme should also contain location information, so that people can easily see when the traffic crosses a WAN link, for example, or determine that traffic is stuck in a loop between Hong Kong and Singapore. For WAN links, the DNS name of the interface should include the name of the service provider for that link. It should also be easy for someone who knows the naming scheme to identify which devices are firewalls, which might have rules that are blocking traffic for the application you are trying to debug.

A good naming scheme often eliminates the need to refer to the documentation, because your standard troubleshooting tools automatically give you all the information you need. Not needing to find and refer to the documentation is a significant time saver, and when debugging a serious incident, every second counts. For example, seeing a hostname such as `nyc4-ge1.example.net` is a good indicator that the hop is through the first gigabit Ethernet interface on the fourth router in New York City.

24.3.3 CMDB

Another essential piece of documentation for troubleshooting and resolving incidents is an accurate inventory system, or CMDB, which tells you the make, model, serial number, and optional components of the device that is causing

problems, as well as its exact location in the datacenter. This information enables you to put together replacement hardware, and quickly find and swap out the problem device with a single trip to the datacenter. It also makes it easy to open a support case for the device, because the serial number is at your fingertips, even if the device is down or inaccessible.

24.3.4 Labeling

Another very important form of documentation is labeling. Label printers are inexpensive; buy at least one for every location where it might be needed. Physical devices should be labeled on their front and back with their name and asset ID. Power cords should be labeled at the plug end to indicate which device is using that cord. Network cables and patch panels should have labels that allow you to easily trace connections without tugging on wires. WAN or Internet connections usually involve customer premises equipment (CPE), which is supplied by the link vendor. These should be labeled with the name and number of the vendor, along with the contract number, or whatever else is needed to open a support case.

24.4 Support

In small companies, there are a couple of SAs who do everything. In larger companies, support is tiered. A helpdesk that responds to alerts and end-user queries and problems, performs triage, and spends a limited amount of time on each issue is level 1 (L1) support. Specialists who take on longer-running or more challenging problems in their area of expertise are level 2 (L2) support. And subject-matter experts (SMEs) who usually work on projects, but are also the final point of escalation within the company, are level 3 (L3) support. We will look at organizational structures for network teams in more detail later in this section.

A network can be managed only if it is observable. Regardless of how the team is organized, the network is best supported by providing tools that gather network information from multiple sources, and then present it in a coherent way. Making these tools available to all support tiers, and training them on using the tools, significantly speeds up debugging and problem resolution. Faster resolution times mean cost savings for the company through increased productivity, along with happier customers. It also means more job satisfaction for the L1 support team, who should be able to resolve more issues than before in the allotted time. It can also be useful to provide end users with access to these tools so that they can help themselves in some situations, and can provide

more complete problem reports when they are unable to fix the problems themselves.

For example, you might create a self-service portal that enables people to query the status of their network ports. This portal could show up/down status, VLAN ID, error rates, and any other settings for a particular network port. Every technical person can be empowered to diagnose her own network issues. A self-service portal could also permit the helpdesk to make basic changes, such as setting the VLAN ID. Doing so shifts that kind of work away from higher-tier support engineers, whose time is more valuable. Such a portal can restrict the changes that can be made to ensure that they are done correctly, safely, and only where appropriate.

24.4.1 Tools

Getting visibility into your network is immensely powerful, and providing that visibility to other teams empowers your whole company to perform better and more efficiently. The network is the center of your company's connected universe. It should not be a black box, but rather a fish bowl, where everyone can see what is going on.

Some of the tools that the network team requires to achieve this overlap with other groups' needs, such as log processing, monitoring, alerting, and event correlation. These tools may be provided by a central team and shared between all the groups that need them, with each team being able to create their own dashboards with the information that they are interested in.

Some other tools are the responsibility of the network team, although they can be leveraged to provide value across other teams, too—particularly those that support end users, servers, services, or applications. Network tooling is constantly evolving, and it is really worthwhile researching what is currently available. At a minimum, you should provide a tool that displays detailed diagnostics for office networks; smart network sniffers; and, for large networks, a network path visualization tool.

The currently available commercial tools are able to do everything described in the following subsections, and in most cases, less powerful free versions are also available. In some cases, you may need to provide an interface that stitches together information from a few different tools to present it well, and there are even tools available to help you to do that.

Diagnostics for Office Networks

Some of the most powerful tools the network team can supply are ones that gather information—for example, everything about a MAC address or everything about a jack. With the right tools the network team can also make available everything we know about an end user’s workstation. Provide a tool that gives a history of workstations that an end user has logged into, along with the OS version and MAC addresses of those workstations. Link that to a tool that gives a history of where those MAC addresses have been seen on the network—which VLANs, switch ports, IP addresses, and DHCP leases they were using at which times. Combine that with a tool that gives a history of the switches and switch ports—port up/down, port errors, port traffic volume, switch up/down, switch load and memory, switch backplane traffic volume, and switch uplink volume. Finally, further enrich the data with extracts from the logs pertaining to that user and those workstations, MAC addresses, IP addresses, and switches.

Without the right tools, much of this historical information is not possible to gather after a problem report comes in. Even when it is all available, it needs to be retrieved from multiple different sources, all requiring some level of access. The right tools can gather and store this information, which can later be searched, retrieved, and combined by other tools. This history can be utilized by the helpdesk and end-user support personnel to quickly narrow down the problem to a particular workstation, a faulty jack, or an overloaded switch, for example. Network discovery tools do most of this work, including device fingerprinting, and can provide richer data when linked with DHCP servers and Microsoft’s ActiveDirectory or other authentication and directory services. The other data described here can be retrieved from your historical monitoring service and your log servers.

Smart Network Sniffers

Another incredibly powerful tool is an array of smart network sniffers connected all across the network. Smart network sniffers capture and analyze network traffic on the fly, storing statistical data about top talkers, top applications, and so on. In addition, they provide flow reports with latency, packet loss, and TCP statistics such as out-of-order packets, retransmission rates, and fragmentation. A flow is an application session between two devices. Smart network sniffers also detect traffic that is indicative of a problem, and start storing a full traffic capture for later analysis. They come with some useful built-in rules, but should also be configurable so that you can specify your own sets of rules to trigger a traffic capture. Full traffic captures are retained for only a short period, since they are large. By comparison, the statistical data is summarized and retained for much

longer.

The statistical data can give you a much better understanding of your network and the way it is used, allowing network, server, and application teams to optimize the way that the network is used so as to maximize performance and minimize costs. For intermittent problems, having a smart sniffer that detects the problem and starts recording when it is identified is invaluable. One of the greatest challenges with intermittent problems is gathering enough data about a transient issue to be able to properly analyze and understand the problem.

Smart Network Sniffer Proves Useful

At one company, the smart network sniffer showed that a couple of machines were doing a lot of DNS queries. These machines were running the monitoring software. The team realized that each time it polled a device, each of these machines was performing a DNS lookup on the device name because the monitoring software was not doing any DNS caching. They enabled DNS caching on the problematic machines. Doing so improved performance on the monitoring server, because it was querying a local DNS cache, rather than sending all queries off-box. It also reduced the load on the DNS servers, which improved DNS performance for all.

Network Path Visualization

Another tool that is particularly useful in a large global enterprise is one that provides network path visualization. Each node (or hop) shown on the path is a layer 3 network device, which routes the traffic to the next layer 3 device in the path.

A network path visualization tool can be used to show the path that packets take from A to B, and from B back to A, which can expose routing asymmetries. It can be used to show geolocation data and the path from multiple locations to a particular service, illuminating how the global load balancing is working for that application. In all cases it should show hop-by-hop performance (latency, jitter, and packet loss), enabling you to identify trouble spots.

This information can be invaluable when service owners receive reports of performance problems with their application from some groups of users, but the application and servers it runs on all look healthy. For example, it can highlight that there is some congestion and packet loss along a particular path, which would otherwise be very difficult to identify. Once the problem is identified, a

solution can be found.

Such tools are usually agent based, with data collectors running on various machines around the network. They are also available as SaaS services on the Internet, and can enable companies to identify and work around problems with intermediate ISP networks, where they have no direct relationship with that ISP.

24.4.2 Organizational Structure

The size and structure of your network team can be as important as the network itself. The size of the company and its network are the key factors in determining your network team's organizational structure.

Small companies may subscribe to a managed network service, which is often provided by their ISP, although there may be other options available. Such a service provides monitoring, a support portal with status information, a support organization the company can call when there are problems, and a portal where designated individuals can make changes or subscribe to additional services. Everything is provisioned and monitored by remote automation. The company will typically receive better overall service than would be possible with a local SA, who typically has a limited budget for tools and must also support any local workstations, servers, phones, printers, and other devices.

Midsize companies typically have in-house staff, some of whom are dedicated network specialists. These people usually perform a combination of project work and L2 support. The helpdesk escalates the more challenging network problems to them, and they work with the vendor, when required.

Large companies typically have a tiered support model, with at least three tiers, as described earlier. The L1 team is often called the network operations center (NOC). In really large companies, sometimes the NOC and the helpdesk will be separate teams. The helpdesk may be called the user support center (USC), and it performs triage, handles alerts for non-network devices, and provides initial support for non-network problems. The NOC handles alerts for network devices only, performs triage on network issues, and spends a limited amount of time trying to resolve each network problem before escalating to L2. When the NOC and helpdesk are separate entities, the helpdesk is referred to as level 0 (L0) for the network team, with the NOC being L1. Sometimes, in companies with an extensive WAN, there is even a separate NOC for the WAN, which handles all communication and escalation with the WAN vendors globally, informs the other teams about WAN problems, and routes around those problems. The WAN NOC typically combines L1 and L2 functions for the

WAN. (The L1, L2, and L3 level designations are not to be confused with the layers of the OSI model.)

Typically the NOC and the helpdesk are centralized teams, even if they are located in geographically diverse locations to do follow-the-sun support. Since problems can impact people in multiple locations, it makes sense to centralize this function.

In contrast, the L2 team may be centralized or may be composed of a set of L2 teams, one for each business unit, or region, country, metropolitan area, or building. When L2 support is regionalized, the L2 support teams are sometimes referred to as regional operations centers (ROC). For large companies, the global L2 network support team is typically sufficiently large that it is not feasible to have all those people reporting to a single manager; thus, some sort of division is required. However, dividing the team across regional or divisional boundaries can result in diverging implementations and standards. For this reason, it is important that there is a unified L3 team, or network architecture team, that defines standards and ensures that they are followed.

An approach that works well is to start by dividing the team along functional boundaries. The L2 team that supports wired office networks could be separate from the one that supports wireless office networks. The datacenter networks are supported by a different team. VoIP is with yet another group, and there are additional groups for the WAN, security, network services, and network tools.

The L2 teams should be as centralized as possible, rather than having someone in every building. A company's network should not be so fragile that it requires dedicated network support in every building. Moreover, it should be built in such a way that it can be supported from one's desk, and from home, for everything but a hardware failure. Members of the L2 team will benefit from working together in a group on a daily basis, being able to discuss problems, ask for advice, and learn from others. The colocation makes it easier for them to cover for each other during vacations, illnesses, conferences, and trainings. It also encourages people to follow the standards, which is best for the overall health and manageability of the network.

Sometimes, however, it is necessary to have someone physically on-site to replace broken hardware. Should someone from a nearby office drive over with the replacement part? Some types of companies, such as retail stores, banks, postal services, and so on, have many branches in diverse locations, perhaps spanning thousands of miles and many countries. Does someone need to fly to the location with replacement hardware? That would be very expensive and slow

support. Large companies with many office locations usually outsource this service. Often the facilities company that cleans the offices and provides security services will also provide trained network technicians who can replace a piece of hardware and reconnect all of the cables. These companies may keep spare parts on hand, or each office may be equipped with spares. The company will handle receiving the return merchandise authorization (RMA) replacement parts, and returning the broken ones. Depending on costs, this model may also be attractive to smaller companies where an L2 support engineer would otherwise lose a few hours of productivity driving to and from the other site to do the replacement.

24.4.3 Network Services

The network team is often responsible for other network services, in addition to OSI layers 1, 2, and 3. For example, sometimes network security services such as firewalls, proxies, intrusion detection systems (IDS), and remote access are also the responsibility of the network team. Other network services that often get assigned to the network team include voice over IP (VoIP), video conferencing, and multimedia services. IP services, such as network time (NTP) and DDI (DNS, DHCP, and IPAM), are sometimes the responsibility of the Windows or Unix server teams, and sometimes the responsibility of the network team. Often this choice is made based on the platform providing the service—Windows and Unix server teams support this service if it is on their platform, whereas the network team supports it if it is provisioned on dedicated appliances. Network tools, as discussed in [Section 24.4.1](#), will usually be the responsibility of the network team as well.

VoIP can be a large and complex topic in its own right, especially for large companies with complex needs. For example, at banks VoIP may be further separated out into standard desk and conference phones, call center, trader voice, and voice recording services. In really big companies with complex VoIP needs, voice and multimedia will usually be a separate group, which may need to work closely with the desktop teams when soft phones are used. The VoIP team then becomes a customer of the network team and the IP services team. The network team needs to provide quality of service (QoS) support for voice and multimedia traffic, and multicast support for streaming video services. The network team may also need to detect the type of device and dynamically assign it to an appropriate VLAN. The IP services team needs to configure particular DHCP options to support these services.

24.5 Summary

The network needs to be highly reliable, because everything else relies on it. However, that does not mean that you cannot or should not make changes to it. On the contrary, you should build a network that enables you to frequently and nondisruptively roll out changes, and create strict but well-considered operational processes that foster agility without compromising reliability. Automation and tooling are key to reaching these goals.

The foundation of network operations is monitoring. Monitoring enables you to detect issues that may become user-affecting incidents, and to resolve them before they reach this stage. Other network tools are often underrated, but also can be incredibly powerful, gathering and combining information in enlightening ways. When you can provide your helpdesk with easy access to all the information about an end-user workstation's network access, it facilitates faster problem resolution and reduces the daily support burden on the network team. Other tools can give surprising and useful insights into your network that can help you debug complex problems and improve performance.

As with workstations, servers, and applications, when you can centrally manage network devices, controlling configuration and software versions of many devices from a single point, it is much easier to treat your network devices like cattle rather than pets. Datacenter fabrics promote this model, and Clos fabric networks can be extended into offices, too. Wired office networks should be made as remotely configurable as possible, ideally with automatic VLAN configuration using NAC, so that they are plug-and-play systems. Wireless office networks should provide excellent coverage and plenty of bandwidth for the growing number of mobile network devices. They should work exactly the same in all offices, and you should always provide guest WiFi access.

Larger organizations usually need tiered network support to respond to issues in a timely manner and to distribute the load in a reasonable way. It is important to give senior network engineers the time and space to work on projects, or the network will stagnate and deteriorate. We described some strategies for tiered support models, and for environments where the network team also supports other network services like voice and DDI.

Exercises

1. Why is monitoring important to operations?
2. Which elements of a network should be monitored?
3. What do you monitor on your network? What would you like to monitor, and how would you go about it?

4. Who can make changes to network equipment? How do they get access, and how is access revoked? How do you know who made changes and what they were?
5. If a change to a network device caused a problem, how would the on-call person roll back the change? To what extent could other changes be lost in the process? How easy would it be to know exactly which change caused the problem, and why? How long would the roll-back take? Which aspects of roll-backs would you like to improve at your company, why, and how would you do it?
6. What are the stages in the life cycle of a network device? How is this life cycle tracked at your company? Which change control rules apply at each phase, and why?
7. Which vendors' equipment do you use in your network? How could you reduce the number of vendors that you use? What would be the advantages and disadvantages of doing so?
8. How many versions of code are you running? What is the smallest possible number of versions you could run, and why? Draw up a plan to reach that goal.
9. Describe the process for rolling out new firmware at your site from the moment that the vendor releases the new version. Do you ever reach 100% coverage? If so, how long does it take? If not, what percentage is normal, why, and how long does it take? How could you improve that?
10. Which network documentation do you have? Who can access it? Who knows where to find it? How often is it needed during troubleshooting?
11. How do people get support for network issues? How much problem diagnosis can your helpdesk or end users do on their own? Which tools are available? Propose a way to make your customers or helpdesk more self-sufficient.
12. How would you organize a three-tier network support organization for a global company? Describe the division of labor, roles and responsibilities, and organizational structure, and indicate how the teams should communicate and hand off work.
13. Which additional network services does the network team support? What is the support model for those services?

Chapter 25. Datacenters Overview

Datacenters are where servers and network gear are housed. Even a company with only a single server needs to put it somewhere. Because many people rely on these machines, servers should be housed in a location that promotes reliability by keeping the servers physically safe and out of the way of accidental bumps, and by providing sufficient clean, protected power and sufficient cooling. There are many types of locations:

- **Datacenter:** A large facility purpose-built for housing computing equipment. Usually a dedicated building but sometimes a floor of a building.
- **Colocation (“colo”) space:** A datacenter run by one organization, which then rents space to other companies.
- **Computer room:** A single room in an office building.
- **Computer closet:** A small closet, usually only large enough for one rack.
- **Telecom closet or patch room:** A small closet with cross-connecting wires; no active equipment or cooling. Sometimes called the demarc room.

We often use the term “datacenter” as a generic term that can have any of these meanings.

This chapter provides an overview of the available options for server location, and some guidelines on choosing the right approach, or combination of approaches, for your company. [Chapter 26, “Running a Datacenter,”](#) discusses datacenter operations.

A Datacenter Is Not an Office

Whichever datacenter solution you choose, remember that this space is intended for computers, not people. People shouldn’t be seated in the machine room. It isn’t healthy for someone to work in a cold, noisy environment. Also, the cost per square meter for datacenter real estate is much higher than the cost for regular office space. It makes much more financial sense to have people sit in less expensive office space.

Another consideration is that human residents aren’t good for the machines. People and the paper that they tend to have in their work areas both generate dust, which gets sucked into the machines and reduces their cooling ability. People also generate additional heat, about 600 BTU per

person, for the HVAC system to extract. The additional heat extraction costs money.

It is better for the people and better for the company to have people sit in office space, not in datacenters.

25.1 Build, Rent, or Outsource

The primary datacenter decision facing any company is whether to build out its own datacenters, to rent datacenter space, or to outsource server management completely to a cloud provider. Usually a hybrid approach is used.

25.1.1 Building

Building and managing a reliable datacenter is very expensive and requires staff with a diverse set of skills. Reliability and high availability do not come without a deep understanding of each technology and its potential failure modes. For example, a datacenter requires experts in power systems, HVAC (heating, ventilation, and air conditioning), datacenter design, networking, facility management, service architecture, and datacenter operations. Employing sufficient staff with such a diverse skill set is an expensive proposition for most companies.

Therefore, designing and building a datacenter is generally done only by very large companies. At large scale, it is cheaper to own than to rent. By controlling all aspects of design and operations, one can drive down costs and find other efficiencies. This practice is called **vertical integration**. In these situations, typically the company will contract the datacenter design and construction process to a third party with expertise in that area, and then have its own staff manage the facility after it is built. Only the largest of companies, such as Google, Apple, and Facebook, can justify hiring full-time staff with such skills.

25.1.2 Renting

For companies with a small or moderate amount of server real estate, it is cheaper to rent space at a colocation facility. Typically this space will be a cage of a certain size with a specified number of racks. The cage provides security from other datacenter tenants. Within the cage, each company is responsible for managing its own space allocation and operations. The colo facility provides reliable, conditioned power, HVAC, fire-suppression capabilities, and racks. Some facilities even provide Internet connectivity.

25.1.3 Outsourcing

Companies with a small or moderate amount of server real estate often outsource the hardware management by using an infrastructure as a service (IaaS) offering. Some IaaS offerings provide bare hardware, with the customers then managing their own OS, applications, and configuration. This enables companies to implement their own change control processes, as appropriate for their business, while removing the burden of hardware support and datacenter operations. Some companies opt to have the OS, and even the applications, installed, patched, and upgraded by the service provider, further reducing the need for in-house expertise.

25.1.4 No Datacenter

Some companies avoid the need to think about any datacenter space at all by relying on web-based application providers, known as software as a service (SaaS). For example, Google Apps for Work, and Salesforce, among many others, are SaaS offerings. The company that develops the application maintains the servers that run the application and sells compartmentalized access to this service to its customers. Customers of this service do not need any in-house support for such applications. For small companies that need only a limited set of applications, it may be possible and economical to purchase everything they need as a SaaS offering, leaving only the local network and Internet connection in the “datacenter” space.

Most Internet service providers (ISPs) offer small business packages, where they support the network equipment at the customer site, including LAN and wireless. With this combination, a small company needs just workstation support, which can often be purchased on a contract basis from a local IT company. Alternatively, the company could purchase VDIs from the cloud provider, and just have thin clients in the office.

25.1.5 Hybrid

It is common to see a mixture of these strategies. A large global company may build large regional hub datacenters in a few strategic locations around the world, but include a small computer room in each office that holds network equipment and servers that provide local services. A medium-size company may rent colo space or buy IaaS services, plus use SaaS-based applications. Small companies may rely entirely on SaaS-based applications and, in turn, need only a small computer closet for their network equipment.

25.2 Requirements

As with any decision, the process of choosing a company's datacenter strategy should start by defining requirements. Requirements are a mixture of business requirements and technical requirements. As SAs, we often end up too narrowly focused on the technical requirements because we are not as familiar with the needs of the business. However, it is the business need that should drive the technical requirements. Once you understand what the business needs, you can look into how that translates into a technological solution.

25.2.1 Business Requirements

Small companies may just have a few business-related requirements. For example, they need to run payroll and perform basic accounting for corporate bookkeeping; they need applications for electronic marketing, email, printing, and scanning; and they need to run a couple of applications that are specific to their core business. It may be that these applications need to be available only during working hours on weekdays. A small company will want to minimize its IT costs and will not have a budget for dedicated IT staff. Larger companies typically rely on more applications and have higher availability requirements. Different departments may have different requirements.

Depending on the business, companies may have basic privacy requirements for their data, or they may work in a regulated environment where the ability to read or modify the data needs to be strictly controlled. When considering the options, you can use the items described in the following subsections as a starting point for the conversation with the business.

Availability

How reliable does the service need to be? Which time zones are the people accessing the service going to be in? Will the service experience near-constant load, or will there be peak usage times? What kind of service level agreement (SLA) is required? Does that differ for peak usage times? Cloud providers are typically better able to specify, monitor, and maintain an SLA than a small in-house team.

Business Continuity

Which requirements does the company have to continue operating, or return to operational status, in case of an outage? If hosting with a third party, what are the arrangements to get data back and service restored at another company in the event of a bankruptcy, hacking event, equipment seizure, and so on? A company

has more control over business-continuity testing and validation when everything is in-house, but SaaS providers are usually a lot better at handling large infrastructure outages.

Budget

How much money can the company spend on IT? What is the budget for operational expenditures (OpEx) versus capital expenditures (CapEx)? Rented datacenter space falls under OpEx, but owned datacenter space is CapEx. Public cloud infrastructure costs are OpEx, whereas the cost of buying the company's own servers is CapEx. Many companies prefer to have more OpEx and less CapEx for financial reasons. Cloud-based solutions allow for rapidly scaling both service and costs up and down as required, which is not the case with a company's own infrastructure.

Change Control

To what extent does the company need to control when IT changes happen, and how much change can happen at once? Does the company need to be able to define **freeze periods**, preventing changes from happening so as to promote stability during business-critical time periods? A company can enforce whatever change control policies it wants on its own infrastructure, but cloud providers are selling a service with an SLA. Typically the customer will have no control over the provider's change cycle, and as long as the provider remains within the terms of the SLA, the customer will have no further recourse, even if there is an outage during a particularly critical period.

Regulatory Constraints

Which regulations apply to the company that impact IT decisions? Can a given cloud provider guarantee compliance and that its solution will pass any audit? Given the severity of the consequences, is that guarantee sufficient, or should that data remain in-house? Which safeguards need to be in place to detect and prevent data leakage? Some corporate data may be subject to data retention or privacy laws. Some data may need to stay within a specific country or region, or must be stored on company-owned computers in company-owned facilities. Other data may require a strict audit trail for access and modification. Failing to meet these regulatory constraints can sometimes put a company out of business.

Data Privacy

Which additional data privacy requirements does the company have? Even when

local laws do not apply, companies may have internal policies with respect to exposure of certain data to third parties. For example, at one company support packages from customers, which would contain the customer's intellectual property, were the most carefully protected data in the company—more so than the company's own source code. This data was not subject to any regulatory protection, but the company's own internal data privacy requirements would require that this data not be uploaded to a public cloud. Note that while public cloud offerings may include an encryption option, it is often the case that the cloud provider also keeps the encryption keys, which negates the benefit of encrypting the data.

Flexibility

How static are the company's computing needs? Does the company need the ability to rapidly scale up and down again in response to particular events? For example, a news media site may be flooded with traffic after a significant event, such as a natural disaster. For a company with its own infrastructure, having sufficient capacity to meet these rare peaks is expensive, but being unavailable because the servers were overloaded may also prove expensive. Cloud-based solutions allow for rapid scaling up and down as required, with companies paying for only what they actually use. The inability to scale can lead to a significant opportunity cost, with a company missing out on a significant business opportunity because it was unable to react quickly enough. For example, in 2010 Netflix was able to support Apple video format with just a few weeks' notice, because its cloud provider (AWS) enabled Netflix to rapidly scale up to meet the demand of re-rendering all the videos in a short period of time. If Netflix had not subscribed to a cloud provider's service, it would not have been able to put together sufficient computing power to meet this demand in the short term.

25.2.2 Technical Requirements

Technical requirements support the business requirements but may be invisible to the executive-level managers who do not have an IT background.

Services and Applications

Which services and applications does the company need to run, and which facilities does it require? Most companies will need basic services such as authentication, DNS, DHCP, email, printing, phones, calendaring, meeting software, file sharing, backups, and so on. In addition to these standard

infrastructure and business tools, there will be some other tools that are more specific to that company's particular business. Evaluate each application to determine if it must run locally relative to its users, it can be run remotely, or there is a viable SaaS product available that meets the business users' requirements at a reasonable cost.

Update Frequency

How up-to-date do the applications and OS installations need to be? How quickly do new features need to be available to the end users? PaaS and SaaS solutions will often be more up-to-date than the options that an in-house team can offer because of the level of automation at those companies. Also, SaaS products are typically updated frequently—sometimes even several times a day. Usually an in-house solution will be built on shrink-wrapped software, with updates available every month or few months. Updates to the production systems will happen even more rarely, because of the effort involved in certifying and rolling out the new release.

Security

Which security requirements does the company have? At a minimum, all companies need to patch security flaws and implement anti-malware measures. All devices need patching. Vulnerabilities need to be tracked and assessed, and then patched in a timely manner. Antivirus and anti-malware measures need to be implemented at various layers, such as on the client workstations and on all communications to and from the Internet. Security and update frequency are related. The faster that the fix for a security flaw is deployed, the less chance there is that it will be exploited. It can be prohibitively expensive for a small company to have the technical expertise required to track, evaluate, and patch security vulnerabilities as they arise.

Remote Access

Do business users need to be able to access any, or all, of the services remotely? For example, some people may need access while traveling on business. A sales team may spend most of their time working from remote customer locations, and may need remote access to key sales applications. Some people may work from home on a regular basis. The company may have small remote offices—they need a way to access the services. Applications or services can fall into three general categories: applications that *must* be remotely accessible, applications that *may* be remotely accessed, and applications that *must not* be accessed from

outside the company network. Those that must or may be accessed remotely are good candidates for a SaaS solution. SaaS providers may also offer geographically diverse solutions, so that users connect to the instance closest to them, which reduces latency, usually yielding a better overall experience. This is an attractive option for a small or midsize company that has geographically diverse employees, but few (or no) datacenters of its own.

Customer Access

Which services need to be available to customers? For example, if a corporate web site is available to customers, does it act as an interface to some other applications? Are there other online services? Where are the customers? Similar to employee remote access, if the customers can be anywhere in the world, a cloud provider is likely to provide a better user experience than the company can. A cloud provider can distribute the application to its global datacenters and route customers to the closest available instance.

Responsiveness

How responsive does each application need to be? Responsiveness is measured by the time elapsed between when the end user does something and when he or she gets feedback from the system. The physical distance between a client and the application affects speed and latency. For some applications, such as document editing, end users expect instantaneous feedback—no lag between when they type something and when it appears on their screens. Similarly, when voice and video communications suffer from delays, it is very hard to have a proper conversation. However, when users click “send” for an email, they expect the email client to quickly indicate that the mail has been sent, but they expect it to take a little longer to show up in the recipient’s mailbox. Understanding the end users’ expectations regarding the responsiveness of each application allows us to properly assess SaaS and in-house solutions in relation to this requirement.

Capacity

How many simultaneous users, or transactions per second, does each application need to be able to handle? How is that expected to change over time? What does that translate into in terms of equipment, network, and datacenter capacity?

Cloud providers are usually better at scaling a service up and down rapidly, and as required. Companies with their own infrastructure are less nimble. However, buying a lot of capacity from an IaaS or PaaS provider may be a more expensive option for a company with relatively static needs. If the company decides to

build its own datacenters, it needs to understand how large they will need to be.

25.3 Summary

Datacenter strategies vary from full-blown in-house solutions to fully outsourced SaaS solutions. Large companies may employ a mix of strategies, running their own datacenters in corporate hub locations, and using a SaaS solution when that provides a better product and service for the cost than an in-house service would.

Building and running one's own datacenters requires a large team with a diverse skill set. It may be cost-effective for large companies, but smaller companies will often be better off with some form of outsourced solution. When choosing the right approach for your company, you need to look at both the business and technical requirements, and assess how each option meets those requirements.

Exercises

1. What are the high-level options for an enterprise datacenter strategy? Explain the differences between the options.
2. A friend of yours is setting up a small company that will develop web sites, and asks you to help with the IT setup. Which approach will you recommend and why?
3. If your friend is setting up a physician's practice, does your recommendation change? Explain.
4. Choose a common business application and investigate SaaS solutions. Which one would you recommend to a small, local business, and why? Which one would you recommend to a large global corporation, and why?
5. Which high-level datacenter strategy do you think is right for your company? List as many services and applications as you can that are in use at your company, and specify which approach you would use for each, and why. Are there any locations where you would use a different approach? If so, what and why?
6. What are the business continuity requirements for each application at your company? How are they met today? What could be improved, and how would you approach it?

Chapter 26. Running a Datacenter

This chapter is about the fundamentals of running a datacenter. Most of us rarely get to build a datacenter, but more often inherit one that must be managed. The way that we manage it has a significant impact on how well the datacenter serves the needs of its users and the staff who operate the datacenter itself.

At the macro level, running a datacenter is about capacity management: making sure there is enough space, power, and cooling based on the ebb and flow of demand for the datacenter. Generally this means growing the capacity as datacenter needs tend to grow over time.

At the micro level, running a datacenter is about life-cycle management: managing the process of adding and removing machines plus coordinating repairs and maintenance windows. To achieve these things, there are best practices for daily operational tasks such as patching cables, labeling, providing communication to and from the datacenter, and providing remote management support.

How each of those tasks is handled affects the overall reliability of the datacenter and the services it provides, as well as the effectiveness of the people who need to work in there. Other smaller items, such as having proper staging areas, workbenches, and tools, also contribute to the effectiveness of your datacenter operations.

What ties all these processes together is the inventory system. When everything is properly tracked through the inventory system, it becomes much easier to manage the datacenter effectively.

26.1 Capacity Management

A datacenter has a certain amount of space, number of racks, and capacity for power and cooling. One of the most fundamental components of managing a datacenter is tracking the usage of each of these components. It is not quick, cheap, or easy to increase capacity for any of these components, so you need to start preparing to budget for, and provision, additional capacity long in advance of hitting the limit of any of these resources.

Computer equipment tends to get smaller over time, meaning that more machines will fit into an existing space. However, that typically also means that the same amount of datacenter space will have greater power and cooling requirements over time. The amount and type of network cabling will also

change over time.

Tracking these resources requires an inventory system. The inventory system should track items such as machine types, along with their size and power requirements, chassis utilization, network connections, console connections, and KVM usage.

Wherever possible, the inventory system should gather data automatically. If it must be gathered manually, the inventory will always be out-of-date. For example, some network tools track switch port usage by querying devices with Simple Network Management Protocol (SNMP). They also track history, and can gather MAC and IP addresses that are associated with switch ports.

Another way to assure the inventory is kept up-to-date is to drive processes off the inventory database. This gives people an incentive to keep the inventory current. For example, if backups, monitoring, and other centralized functions are based off the inventory, people will update the inventory. If the inventory database has an owner field, and that owner is billed for space, people will have an incentive to power off and remove unused machines.

Small Companies Need Inventory Management, Too

A small company didn't believe it needed an inventory system, given that it simply had nine racks in a rented colo. Nine racks was small enough that one of the SAs could basically keep the inventory in his head. Churn was slow enough that when new machines were installed, one could simply schedule a visit to the colo and verify the planned rack location was available.

The company eventually installed an inventory system called Device42, with plans to use its ability only to track IP addresses. However, since the system could do automated discovery, the company enabled that feature just to see how well it worked.

Soon the full inventory was being maintained in the database to great benefit. All the SAs had visibility to which space was available, rather than relying on one person's memory. They discovered machines that weren't being used or monitored or maintained. Fixing these situations led to reduced power usage and improved reliability.

26.1.1 Rack Space

How much free rack space does your datacenter have? If you needed to rack five

devices that are each three rack-units (3U) high, do you know where you would put them, or would you need to walk around the datacenter looking for space? If space in your datacenter is divided based on cost center, security zone, or some other parameter, do you know how much space is available in each area?

Tracking rack space is one of the many tasks for which having standard hardware types and an up-to-date inventory system make your work much easier. Associate each asset in the inventory database with a standard hardware type, which has certain dimensions. Also put each rack into the inventory database, associated with its standard type, which has certain dimensions. Give each rack a location, and optionally a “zone” or other metadata that constrains its usage.

When you install a machine in a datacenter, assign it a rack and a position in the rack. For devices that are greater than 1U high, local standards can define whether the rack position that is specified in the inventory is the lowest or highest rack position that the device occupies. Since the device types, and therefore their dimensions, are tracked in the inventory system, the inventory system can track which rack locations are occupied by each device. At this point, you can use the inventory system to generate reports on how much rack space is available, broken down by height, depth, “zone,” and any other parameter that is important to you. You can also use it to allocate and reserve rack space for equipment as it is ordered, so that you know where to place it when it arrives, and can pre-wire the space as necessary.

Once you can track rack space utilization, you can generate graphs and reports based on occupied, reserved, and free space. This enables you to understand how fast datacenter utilization is changing, and to predict when you will need to start planning for more (or less) capacity. Historical graphs are an excellent tool for presenting the information to management.

For sites that do not yet track any of this data in inventory, it may seem like a lot of work to get started. Perhaps it is, but it can save you from finding yourself in the uncomfortable situation of running out of datacenter capacity before you are able to bring more capacity online. Tracking this data in inventory also yields time-savings later. New computer placement can be done from your desk, rather than requiring an on-site visit. For companies that rent datacenter space, often in remote locations, this consideration is particularly important.

Running Out of Rack Space

A company that did not track rack space utilization in inventory realized too late that it needed more datacenter space. As a result, it ran out of

space in its datacenter while an additional datacenter was still under construction. In the meantime, SAs still had to install machines. With no good options available, they came up with a creative—albeit far from ideal—solution.

The SAs realized that many of the older, large, free-standing machines were themselves rack-like cabinets with unused space inside. The SAs started installing smaller machines inside the still-running older machines, diligently labeling the main machine with its own name and listing the machines that were inside. It was an unusual practice and made machines more difficult to find if SAs didn't remember to look at the larger machines as additional racks. However, the biggest problem was that they were consuming more power per square foot than the UPS could manage and risked running out of power capacity. Ideally, the new datacenter should have been commissioned before they reached this point.

26.1.2 Power

Power is another component of the datacenter that requires planning and significant time to scale up. It is also important to track closely, so as to be able to plan ahead.

Each component of the power system has a limit. Each UPS can supply only a certain amount of power, as can each generator. The power company can supply only so much power to the building. The ATS, wiring, circuit breakers, and any other components of the power distribution frame all have limits.

When a datacenter is built, the power system will be sized based on the typical power consumption per square meter for a datacenter. However, over time, this number changes. Machines get smaller, so more can fit in the same space, increasing the total power consumption. Machines get more efficient, consuming less power than previous generations, reducing the total power consumption. So, is your power consumption going up or down? Probably up, but if you don't track it, you won't know for certain.

A number of datacenter-related metrics should be tracked and graphed over time. Each machine vendor can supply you with minimum, average, and maximum power consumption values for its equipment. These values should be associated with the standard machine types and components in the inventory system. Since the inventory system is also used to track machine location, automated periodic extracts of the inventory data will enable you to generate historical graphs of that data over time.

The other metrics to track relate to actual usage. There are three important areas to track here: the amount of power that the datacenter is drawing from each UPS, the amount of power that the cooling system is drawing, and the amount of power that the UPS is drawing from the mains. The third metric is important for sizing the generators, which need to be able to produce enough power to keep the UPS and cooling systems running. It also lets you know when you may need to ask the power company to bring more capacity to the building.

Datacenters are usually built with at least two separate power subsystems. Each rack has power strips connected to at least two of the separate power subsystems. Machines with redundant power supplies have each power supply connected to a different power subsystem. If there are two power subsystems and all machines in the datacenter have redundant power supplies, then you should be able to take down either of those power subsystems for maintenance without causing an outage. However, for that strategy to work, you also need to make sure that either one of the power subsystems can take the full peak load of the datacenter. In other words, the peak load on subsystem A plus the peak load on subsystem B must not exceed the maximum capacity of the smaller of the two power subsystems.

With knowledge of the limits of each component, sufficient historical data, and knowledge of your company's plans for growth, it is possible to estimate when each component of the power system will need to be scaled up. Combine that information with lead times, budgetary cycles, and the approval process, and you will know when to begin.

Running Out of Power

One large company found that it was running low on power in one of its datacenters. The datacenter staff did track utilization, with both measurements and theoretical numbers from their inventory system. The company started the process to increase power capacity, but found that some of the lead times had increased, and the datacenter personnel realized that they would not be able to meet the demand if the utilization continued increasing at projected rates.

The company introduced some measures to keep the demand for power in check until the new capacity came online. It established a "one in, two out" policy for servers, meaning that to install a new server, SAs needed to decommission and remove two old servers. The aim of this policy was to encourage more aggressive decommissioning of old, power-hungry equipment. They made exceptions for critical network equipment, and for

one-to-one replacements of end-of-life (EOL) hardware that was being replaced with something that consumed less power.

Using these techniques, the datacenter personnel were able to keep power consumption within the limits of what the datacenter could deliver until they were able to bring additional capacity online.

Larger sites have multiple power sources, with separate UPS units and generators. Some sites will have multiple small UPS units behind one ATS and generator bank. Small in-rack UPS units are generally more power-efficient than large datacenter-scale UPS units. Multiple small UPS units are harder to maintain, but limit the potential impact of a UPS failure. Whatever power architecture your site uses, encode it into inventory; then make sure that the machines in the datacenter are associated with the correct power source, and that you are collecting data from all units. This makes it possible to conveniently track power capacity on all sources with minimal effort. It also makes it easy to identify which devices will be impacted by scheduled maintenance.

Reduce Utilization by Tracking Ownership

Sun Microsystems had dozens of datacenters and a casual policy about who could put machines in them. In the early 2000s, management realized these datacenters were costing the company a lot of money, especially in terms of power used. The company set a goal of reducing power consumption by a certain number of kilowatts and consolidating the datacenters.

With no inventory system in place, no one knew which department owned each machine. Before employees could start the power reduction project, they needed an inventory system that tracked machine ownership.

They began identifying owners based on machine name, email blasts, and other forensic techniques. As owners were identified, many asked that their machines be decommissioned. These machines hadn't been used in years and had been forgotten, though they had been using power all this time. Enough machines were decommissioned this way that the power goal was met before the official consolidation began.

26.1.3 Wiring

Most datacenters will have some amount of wiring infrastructure. For example,

each rack may have a patch panel at the top that is wired back to a central network row. Like power, the wiring infrastructure is a resource that should be actively managed. This wiring should also be tracked in inventory, with the type of wiring (e.g., copper or fiber) and the specification (Cat-6, Cat-7, OM3, OM4, . . .), with provision being made to take patches out of service when they are found to be bad, and put back into service when they are repaired.

When the availability of infrastructure wiring is also tracked in the inventory system, it can save SAs time when installing new hardware. They can look for a rack with sufficient space, power, and cabling of the appropriate type while sitting at their desks, and then reserve the resources they need.

26.1.4 Network and Console

Similarly, the usage of network and console ports should be tracked in inventory. It is frustrating to install a device, only to find out that there are no console ports available, or need to access your device's console, only to find that someone "borrowed" the connection to install their device. Equally important, when installing a device, you do not want to discover that all the switch ports are already in use and you need to order more network equipment before you can bring your already delivered server online.

26.2 Life-Cycle Management

Day-to-day datacenter operations revolve around the life cycle of the computer equipment within that datacenter. The life cycle of datacenter equipment starts with installation, which is followed by a series of moves, adds, and changes, and some maintenance work. Eventually the equipment is decommissioned. When these datacenter processes are all based around the inventory system, the datacenter is much easier to manage.

As discussed previously, capacity management is much easier with an inventory than without one, and can be graphed and even automated when such a system is used. A good inventory can be used to automatically build a dependency matrix for each device. If a service or a server is having a problem, which components does it rely on? This question can be easily answered from the inventory. If several components are having an issue, what do they have in common? The inventory system can be a powerful tool for SAs who did not build the service and perhaps are not so familiar with it.

Such a system also allows you to automatically build the failure domain for a device. For example, if you must do work on a particular UPS, you need to know

which machines rely on that UPS, and of those, which machines have a redundant power supply that is connected to a different UPS, and therefore should be unaffected. However, to reap these benefits, it is critical that everything is tracked through the inventory system.

26.2.1 Installation

When a system is installed in a datacenter, it takes up some rack space, uses some power, adds heat, and uses network and console resources. Update the inventory to show which device is racked in which location, with its type and additional components. This information automatically updates the rack space, power, and cooling figures. In addition, update the inventory system with information on which power rails, console, patch panels, and switch ports are being used. Many other reports and processes can be built upon an accurate inventory.

Also be aware of the airflow design of the datacenter. Most datacenters are designed with hot and cold aisles. Devices should be racked so that they draw in air from the cold aisle, and vent it out into the hot aisle. New cooled air is pumped into the cold aisles, and hot air is extracted from the hot aisles. If you rack a device the wrong way around, it will draw in hot air, and may be unable to cool itself sufficiently.

26.2.2 Moves, Adds, and Changes

A datacenter is a dynamic environment. Devices may have components and network connectivity added, changed, and removed. Having good rack locations in the inventory system makes it a lot easier for a datacenter technician to be able to quickly locate and modify a particular device.

To support changes to modular systems, the inventory system needs to be configured so that devices with expansion capabilities and swappable components are assemblies with which components can be associated. Each time the configuration of an installed device changes, the inventory must be updated accordingly. Each additional component uses some more power and cooling, and may use additional ports and cables.

Similarly, network or console equipment may be swapped out as part of a hardware refresh program or due to a failure. Likewise, a component failure in a network device may cause you to repatch everything on the failed board to another board or another device. Make sure that you track all of these changes.

Whenever possible, plan out the moves using the inventory system. It is more

comfortable and convenient to work from a desk than in the datacenter. Then execute on the plan, and mark it as done in inventory, which commits the changes. Using the inventory as a planning tool ensures that it gets updated. For sites with a dedicated datacenter operations team, the SAs should do their planning using the inventory, which then generates a service request ticket for datacenter operations with all the necessary details. When datacenter operations completes the task, they update the ticket with any variations they had to make to the original plan, and then close the ticket. Closing the ticket commits the changes to inventory.

26.2.3 Maintenance

It is a very rare piece of hardware that does not need maintenance during its time in the datacenter. Most hardware eventually suffers some form of failure. Disks, fans, power supplies, memory, whatever components your machines have—they are imperfect and can fail. Even if they do not fail, you may want to upgrade the hardware to add capacity. All of this falls under the aegis of maintenance.

Hardware maintenance requires access to the machine. Depending on the machine and how it is designed, it may be possible to work on some components while the machine is still in position in the rack. Some machines are designed so that you can slide them out on rails to perform more complex maintenance without having to remove the machine from the rack entirely. In other situations, you will need to remove the machine completely.

In a well-run datacenter, it should be possible to work on any machine without disrupting the nearby machines. However, in datacenters where cables are poorly managed, doing maintenance on a machine may require or risk disconnecting the power or network cables of another machine. Rigorous cable management that ensures all machines can slide out without snagging on something and disrupting another service pays off many times over in ease of maintenance and reduced outages.

Hardware Life-Cycle Automation

At Google, the hardware maintenance of equipment is automated as much as possible. When the monitoring system detects that a machine could be suffering from hardware problems, it moves the machine into maintenance state in inventory, and runs a full series of diagnostics on the machine. The diagnostics are analyzed to detect the likely problem. A trouble ticket is then created, with the results of the analysis as well as the full hardware diagnostics, and the ticket is assigned to the appropriate

team.

When the hardware support team removes the device from the datacenter, they flag it as out for repair. When they return it to the datacenter, they set its status to “repaired,” which kicks off a series of tests. Once it passes all the tests, the device is automatically reintegrated into the service.

26.2.4 Decommission

Eventually devices in the datacenter need to be decommissioned. Depending on how confident you and management are about your ability to remove a device without impacting another one, this may need to be an off-hours job (assuming your company has off-hours).

Some companies will dictate that the patch cables are to be removed only on designated days. Removing cables often results in some tugging on other cables, which may disrupt the network connectivity or cause a loss of power on other devices. As a result, this work may be considered high risk, and scheduled accordingly.

In any case, the inventory system needs to be updated. Disposing of a device in the inventory should result in all of the resources that it uses being freed up.

26.3 Patch Cables

Everything in the datacenter is interconnected with patch cables. Some sites choose to color-code their network cables. At the very least, cables of different qualities (Category 5, Category 6) and cables with different wiring (straight through, crossover) should be different colors. Some sites choose to have each subnet have a different color cable. Some sites reserve red for networks that are “live” on the Internet with no firewall protection. Other sites use red for crossover cables.

The short network cables that one uses to connect from a network outlet to a machine, or between two patch panels, or from a patch panel to a machine, are called **patch cables**, or simply patches. These cables are typically 1, 2, or 3 meters long. If you color-code by network type or copper category, you should use the same color-coding system for patches.

Tip: Buy Patch Cables Rather Than Making Them

Some people prefer to make their own patches, which can be done by

buying the right parts and a tool called a **crimper**. Patches are very inexpensive to make, which is an excellent justification for this practice. However, time and time again erratic network behavior and outages are traced to handmade cables. As networks get faster, tolerances get smaller. Making a Cat-5 cable that passes certification is very difficult. A Cat-6 cable can fail certification for even minor reasons; for example, each pair of wires needs to be twisted a specific number of times per meter, and each twist reduces crosstalk by a certain amount. To attach the modular RJ-45 connectors on each end one must untwist each pair, but if you untwist more than a few inches the crosstalk will be high enough that the cable will fail certification. It really is that demanding. How much time do you want to spend making and remaking cables until they pass certification?

When purchased in bulk, the price of a patch is quite reasonable. We don't recommend making them by hand.

All network and console wiring for servers in a rack should stay within that rack, other than the pre-wiring. Make sure that there is adequate cable-management space within the rack for the intra-rack cabling. Get cables in a variety of lengths so that you will always be able to find a cable that is the right length. Otherwise, you will have to deal with either a rat's nest of cables on the floor or a web of crisscrossing cables at the back of the rack. It always should be possible to find a cable that will run through the cable management with sufficient slack for sliding the machine forward a little and for seismic events. The cable should not have so much slack that it leaves a long trailing loop. If your hosts are placed on shelves that pull out, make sure that there is enough slack in the cables so the machines can keep functioning even when the shelves are completely extended. Cables should never run diagonally across the rack, where they will get in the way of someone working in the rack later.

Some datacenters may not have the capability of pre-wiring their racks. For example, a colocation center that will have customer equipment in the racks cannot know which kind of equipment will be in the racks and how many connections will be leaving a set of racks to be connected to other sets of racks or to the colocation center's own network equipment.

Tip: Why Do Patch Cables Come with Two Tie-Wraps?

Ever wonder why each individual patch cable you purchase has two tie-wraps? It isn't just so that the cables don't get tangled during transport. It

isn't to annoy you when you are trying to quickly unpack a large number of cables. It is so that you can make your installation neat and clean. When you are ready to use the patch, undo the tie-wraps and install the cable. Now recycle the tie-wraps by using them to latch the patch to the rack or other cable-management rail. Your cables will always be tidy. Alternatively, velcro cable ties are nicer to work with and can be purchased in bulk and made available for all cable management.

This technique does not work with fiber cables because the tie-wraps can pinch them, thereby damaging them.

Cable Bundling

In a computer room that isn't pre-wired, you will find yourself running a cable each time you set up a new machine. Consider making a bundle of 6 or 12 cables and running the entire bundle. It takes only a little longer than running one cable, and the next time a new machine is being installed, there's a good chance that there will be an unused cable that will work for that situation. It's useful to run a bundle from the network rack/row to a rack with a lot of empty space. To make a bundle, follow these steps:

1. Get 12 cables of the same type and length. Remove any packaging, but leave them tie-wrapped.
2. Label both ends of each cable. For example, label each end of the first cable A-1. Then label the ends of the second cable A-2. Continue until each end of every cable is labeled. (To make things easier, the next bundle can be B-1 through B-12.) It is important to label them now; trying to accurately label cables after they are run can take hours.
3. Find a long room or hallway without a lot of traffic.
4. Remove the tie-wrap from a cable, saving the tie. Run the cable down the hallway.
5. Repeat the process with the other cables.
6. Use the tie-wraps you've collected to bundle the cables. You should have enough wraps for one every couple of feet. Leave a meter or two free on each end.

Another trick for optimizing your cabling is to have vertical power

distribution units (PDUs), with lots of outlets, mounted at the sides of the racks. Buy a lot of really short power cords in a couple of lengths—for example, 1 foot and 2 feet—and plug each piece of equipment into the power socket next to it. As depicted in [Figure 26.1](#), short power cables are convenient and help to keep the wiring neat. This also avoids having long power cords trailing all over the rack next to the data cables and possibly even causing interference problems.

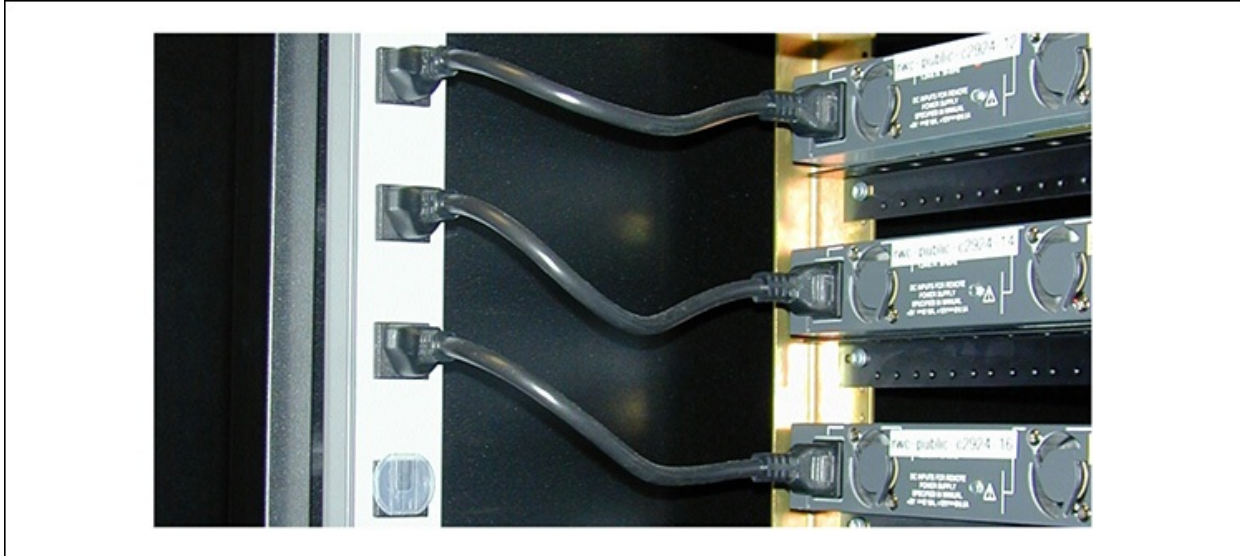


Figure 26.1: Vertical PDUs at GNAC, Inc.

Separation of Power and Data Cables

At a site where Christine performed consultant work, an SA received a report of a network problem. The customer who reported the problem found that data transfer between two hosts was very slow. The SA verified the problem and did further tests. She found that the network interface of one of the machines was recording a lot of errors. She went down to the datacenter to check the cabling. It all seemed solid, and replacing the cables made no difference.

While she was doing that, however, she noticed that the power cord of the machine that she had installed in a rush earlier in the day was crossing over the network cable that went into the interface that was having problems. All the other power cords were carefully kept away from network cables and neatly run through the cable management. She remembered Christine telling her about keeping network and power cables apart because of electromagnetic interference, so she took the extra minute or so to run the power cord through the cable-management system with the rest of the power cords. When she tested again, the network

problem had vanished.

26.4 Labeling

Good labeling is essential to a smoothly running datacenter. All equipment should be labeled on both the front and the back with its full name as it appears in the corporate namespace (see [Chapter 39](#), “[Namespaces](#)”) and in the console server system (see [Section 26.5](#)). Each rack should be labeled with its location in the datacenter, which is also tracked in the inventory system. Pre-wired patch panels should be labeled at both ends, to indicate where the corresponding panel is. Cables should be labeled so that you can positively identify both ends of the cable.

26.4.1 Labeling Rack Location

Racks should be labeled based on their row and position within the row. Put these labels high on the walls so that they can be seen from anywhere and the racks will be easy to locate. [Figures 26.2](#) and [26.3](#) show this form of rack-location labeling and how it is used on patch panels. Racks are clearly labeled at the top and have a patch panel that indicates the rack to which it is wired.



Figure 26.2: Rack numbering high on the walls at Synopsys

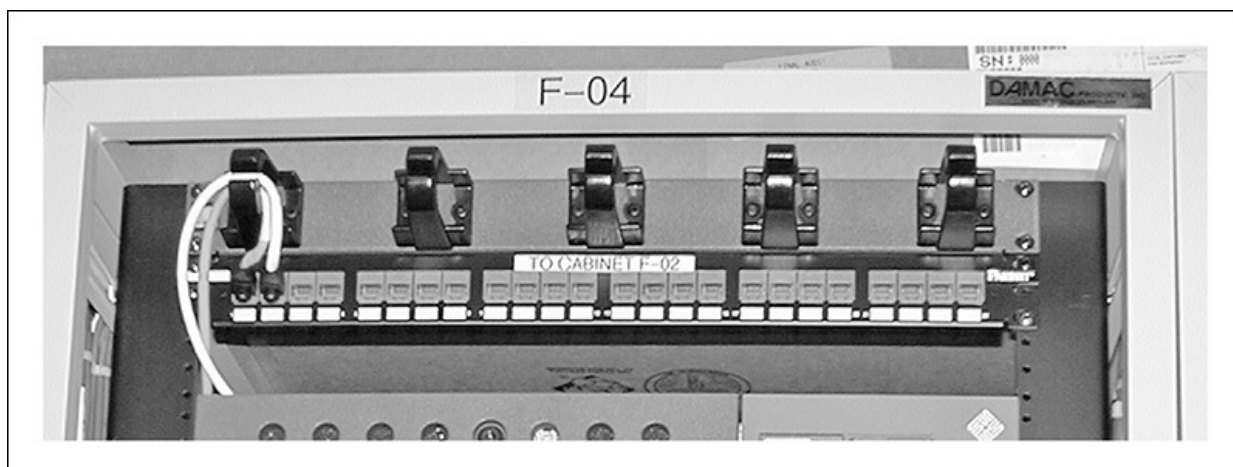


Figure 26.3: Rack labels at Synopsys

26.4.2 Labeling Patch Cables

If a machine has multiple connections of the same kind and it is not obvious from looking at the machine which one is used for which function, such as multiple network interfaces that belong on different networks, both the interfaces and the cables should be labeled. Color-coding the network cables can also help, with a different color perhaps being used for each security domain (it can be difficult to have a different color for every network at a large site).

For example, a firewall may have three network interfaces: one for the internal, protected network; one for the external, unprotected network; and one for a service network that is accessed from untrusted networks through the firewall. The interfaces should at least be labeled “int,” “ext,” and “serv,” and cables should have labels with corresponding tags attached. When you are debugging a problem, you will then be able to easily determine that the external network card has no link light. When you have to pull it out of the rack to work on a hardware fault, you will be able to put it back in and reconnect all the cables without having to think about it or trace cables.

Labeling both ends of every cable becomes tedious, especially when cables get reused and old labels must be removed and new ones attached. Cables are also notoriously difficult to label because not many labels stick well to their PVC shells over the long term. A useful alternative is to get pre-labeled cables that have their type and their length encoded into the label, along with a unique sequence number, and have the same label at each end. You then have an easier way of finding the other end of the cable—if you know approximately where it is already—rather than tracing it. Even if you have to trace it, you can confirm that you have the right cable before disconnecting it by checking the numbers.

Another alternative is to find cable ties with a flat tab at the end that normal labels will stick to. The cable ties can be permanently attached to either end of the cable, and labels on the tabs can be changed relatively easily.

If you are labeling the cables by hand, label them before you run the cables. This bears repeating: Label, then run. Otherwise, you will spend half a day playing guessing games until all the runs are labeled. We know this from experience.

Policy for Enforcing Labeling Standards

Eircom has a very strict labeling policy. Servers must be labeled front and back, and every power cord must be labeled at the far end with the name of the machine it is attached to. Network cables are color-coded rather than labeled. The policy is briefly and clearly described in a sign on the datacenter wall, as seen in [Figure 26.4](#).

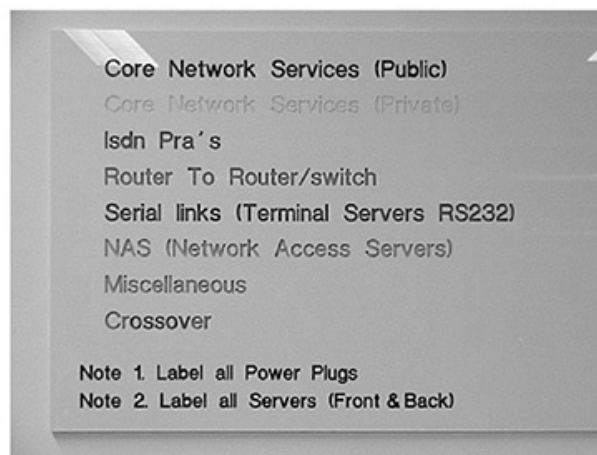


Figure 26.4: Cabling and labeling policy at Eircom

Periodic sweeps to check labels are made; any server or power cord that is not labeled is removed. This policy makes it very clear that any resulting problems are the fault of the person who installed the machine without labeling it or the power cord, rather than the fault of the person who disconnected the machine. Because these sweeps happen frequently, however, machines that do not comply with labeling standards are typically disconnected only before they have gone into production.

26.4.3 Labeling Network Equipment

For network equipment that connects to WANs, both the name of the other end

of the connection and the link vendor's identity number for the link should be on the label. This labeling should be on the piece of equipment that has the error lights for that link. For example, a CSU/DSU for a T1 would have a label that reads "T1 to San Diego office" or "512K link to WAN MPLS cloud," as appropriate. In addition, the T1 provider's circuit ID and telephone number should be included. Listing the phone number saves having to find it when there is an outage.

Network equipment typically also has facilities for labeling ports in software. The software-labeling facility should be used to its full potential, providing at least as much information as is available from the physical labels. As network equipment becomes smaller and more integrated, and as detailed physical labeling becomes more difficult, the software labels will become the most convenient way to store information that you need for debugging.

Using both physical labeling and software labeling leads to having multiple sources of information, which may differ. It is important to make sure that they are synchronized so they give the same information. Make someone responsible for ensuring that physical and software labels match, finding out the correct information, and fixing the labels when they do not match. Nothing is worse than having multiple sources of information that all disagree when you are trying to debug a problem.

It takes diligence, time, and effort to keep labeling up-to-date, but it saves lots of time during an outage, when it is important to be able to respond quickly. It can also prevent accidental outages from happening due to someone tracing a cable to the wrong spot.

26.5 Console Access

Certain tasks can be done only from the console of a computer. Console servers and keyboard, video, and mouse (KVM) switches make it possible to remotely access a computer's console. Modern servers include these features in their remote management subsystem, known as IPMI, iDRAC, iLOM, and so on. For an in-depth discussion, refer to [Section 14.3](#).

Console servers allow you to maintain console access to all the equipment in the datacenter, without the overhead of attaching a keyboard, video monitor, and mouse to every system. Having lots of monitors, or heads, in the datacenter is an inefficient way to use the valuable resource of datacenter floor space and the special power, air conditioning, and fire-suppression systems that are a part of it. Keyboards and monitors in datacenters also typically provide a very

unergonomic environment to work in; that discomfort can be an unhappy situation if you spend a lot of time on the console of a server attached to a head in a datacenter.

Console servers come in two primary flavors. In the first flavor, switch boxes allow you to attach the keyboard, video monitor, and mouse ports of many machines to the switch box, which enables you to access each of these machines as if you were sitting in front of it. This access is sometimes using a single KVM that is in the datacenter. Try to have as few such KVM heads in the datacenter as you can, and try to make the environment they are in an ergonomic one. A better version is IP-based access, often using a web browser, where you can access these consoles remotely. Vendors of KVM-over-IP switches often also offer a management system that provides security, authentication, authorization, and logging for this KVM console access.

The other flavor is a console server for machines that support serial consoles. The serial port of each of these machines is connected to a serial device, such as a terminal server. These terminal servers are on the network. Typically, some software on a central server controls them all and makes the consoles of the machines available by name, with authentication and some level of access control. The advantage of this system is that an SA who is properly authenticated can access the console of a system from anywhere: desk, home, and on the road and connected by remote access. Installing a console server improves productivity and convenience, cleans up the datacenter, and yields more space.

It can also be useful to have a few carts with dumb terminals or laptops that can be used as portable serial consoles. These carts can be conveniently wheeled up to any machine and used as a serial console if the main console server fails or an additional monitor and keyboard are needed. One such cart is shown in [Figure 26.5](#).

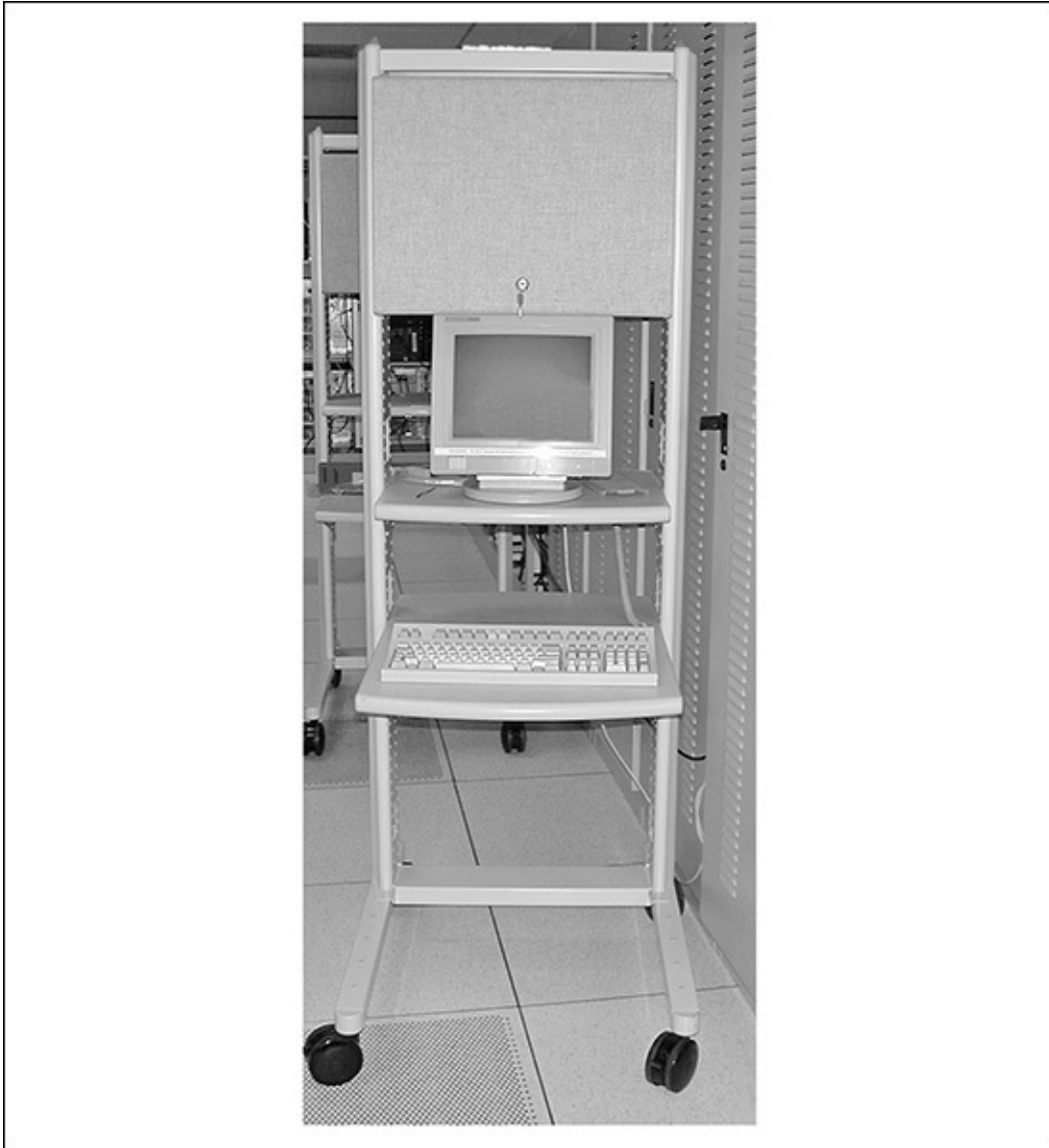


Figure 26.5: Wheeled carts at Synopsys

26.6 Workbench

Another key feature for a datacenter is easy access to a workbench with plenty of power sockets and an antistatic surface where SAs can work on machines—for example, adding memory, disks, or CPUs to new equipment before it goes into service or perhaps taking care of something that has a hardware fault. Ideally, the workbench should be near the datacenter but not part of it, so that it is not used as temporary rack space and so that it does not make the datacenter messy. These work spaces generate a lot of dust, especially if new hardware is unboxed there. Keeping this dust outside the datacenter is important.

If space is not available in which to perform this sort of work, SAs will end up

doing repairs on the datacenter floor and new installs at their desks, leading to unprofessional, messy offices or cubicles with boxes and pieces of equipment lying around. A professionally run SA group should look professional. This means having a properly equipped and sufficiently large work area that is designated for hardware work.

People Should Not Work in the Datacenter

Often SAs have offices located inside the datacenter, right next to all the machines. We strongly recommend against this for various reasons:

- Datacenters are not ergonomic. The datacenter has the perfect temperature and humidity for computers, not people. It is unhealthy to work long hours in such a cold room and dangerous to work around so much noise.
- It is bad for the systems. People generate heat. Each person in the datacenter requires an additional 600 BTU of cooling. That is 600 BTU of additional stress on the cooling system and the power that runs it.
- It is bad from a financial standpoint. The cost per square meter of space is considerably more expensive in a datacenter.
- SAs need to work surrounded by reference manuals, ergonomic desks, and so on—that is, in an environment that maximizes their productivity.

People should enter the datacenter only for work that can't be done any other way. Remote access systems, once rare, are now inexpensive and easy to implement.

26.7 Tools and Supplies

Your datacenter should be kept fully stocked with all the various cables, tools, and spares you need. This is easier to say than to do. With a large group of SAs, it takes continuous tracking of the spares and supplies and support from the SAs themselves to make sure that you don't run out, or at least run out only occasionally and not for too long. An SA who notices that the datacenter is running low on something or is about to use a significant quantity of anything should inform the person responsible for tracking the spares and supplies.

26.7.1 Tools

Ideally, tools should be kept in a cart with drawers, so that the set of tools can be wheeled to wherever it is needed. In a large machine room, you should have

multiple carts. The cart should be equipped with screwdrivers of various sizes, a couple of electric screwdrivers, Torx drivers, hex wrenches, chip pullers, needle-nose pliers, wire cutters, knives, static straps, a label maker or two, and anything else that you find yourself needing, even occasionally, to work on equipment in the datacenter. A cordless screwdriver is invaluable in a datacenter. Where to charge it, however, is not so simple. Often the charger will not like the power supplied by a PDU (often 208 V), or there may not be capacity in your PDU to do so. Instead, arrange for a 110 V, 20 A outlet near a workbench or other space.

It is useful to have many kinds of carts and trucks: two-wheel hand-trucks for moving crates, four-wheel flat carts for moving mixed equipment, carts with two or more shelves for tools, and so on. Mini-forklifts with a hand-cranked winch are excellent choices for putting heavy equipment into racks, enabling you to lift and position the piece of equipment at the preferred height in the rack. After the wheels are locked, the lift is stable, so the equipment can be mounted in the rack both safely and easily.

26.7.2 Spares and Supplies

Spares and supplies must be well organized so that they can be quickly picked up when needed and so that it is easy to do an inventory. Some people hang cables from wall hooks with labels above them; others use labeled bins of varying sizes that can be attached to the walls in rows. A couple of these arrangements are shown in [Figures 26.6](#) and [26.7](#). The bins provide a more compact arrangement but need to be planned for in advance of laying out the racks in the datacenter, because they will protrude significantly into the aisle. Small items, such as rack screws and terminators, should be kept in bins or small drawers.



Figure 26.6: Labeled blue bins at GNAC, Inc.



Figure 26.7: Blue bins and hanging cables at Eircom

Many sites prefer to keep spares in a different room with easy access from the datacenter. A workroom near the datacenter is ideal for this purpose. Keeping spares in another room may also protect them from the event that killed the

original. Large spares, such as spare machines, should always be kept in another room so that they don't use valuable datacenter floor space. Valuable spares, such as memory and CPUs, are usually kept in a locked cabinet.

If possible, you should keep spares for the components that you use or that fail most often. Your spares inventory might include standard disk drives of various sizes, power supplies, memory, CPUs, fans, or even entire machines if you have arrays of small, dedicated machines for particular functions.

26.7.3 Parking Spaces

A simple, cheap, effective way to improve life for the people who work in the datacenter is to have designated parking spaces for mobile items. Tools that are stored in a cart should have their designated place on the cart labeled. Carts should have labeled floor space where they are to be kept when unused. When someone is done using the floor-tile puller, there should be a labeled spot to return this device. The chargers for battery-operated tools should be stored in their own secure area. In all cases, the mobile items should be labeled with their return location.

Case Study: Parking Space for Tile Pullers

Two tile pullers were in the original Synopsys datacenter, which had a raised floor. However, because there was no designated place to leave the tile pullers, the SAs simply put them somewhere out of the way so that no one tripped over them. Whenever SAs wanted a tile puller, they had to walk up and down the rows until they found one.

One day, a couple of SAs got together and decided to designate a parking space for these tools. They picked a particular tile where no one would be in danger of tripping over them; labeled the tile to say, "The tile pullers live here. Return them after use"; and labeled each tile puller with "Return to tile at E5," using the existing row and column labeling on the walls of the datacenter.

The new practice was not particularly communicated to the group, but as soon as they saw the labels, the SAs immediately started following the practice. It made sense, and they wouldn't have to search the datacenter for tile pullers any more.

26.8 Summary

When working in an existing datacenter, where you don't have the option to completely build this facility from scratch, there are still some things that you can do to make working in the datacenter easier. In particular, when you use an inventory system to drive all datacenter processes, you can reap the benefits of being able to track and graph the capacity of all your datacenter assets. Increasing datacenter capacity needs a very long lead time, so it is important to have the right tools and tracking available to be able to predict your future needs far in advance.

Cabling is one of the major factors that determines how easy or difficult it is to work in the datacenter. It is also a significant factor in datacenter reliability. Neat, well-labeled cables reduce outages. Another way to make the datacenter easier to work in, and to ensure there are fewer problems, is to make sure that everything is properly labeled.

Set up your datacenter so that very few people need access to it. Make sure that all devices can be reached using properly secured remote console connections. Wherever possible, use properly secured out-of-band management to enable people to perform remote power-cycles and other low-level diagnostics. Do not locate an SA's office or desk within the datacenter. Make sure that there is adequate space near the datacenter for staging machines and performing maintenance on them.

For those who do need to work in the datacenter, make sure that there are adequate tools, spares, and power and network cables in a variety of lengths. Designate parking places for movable equipment, such as carts and tile pullers.

Seek ideas from the SAs; all of them will have features that they particularly like or dislike. Incorporate the good ones, and learn from the negative experiences of others.

Exercises

1. How much space is occupied by monitors in your datacenter? How many could you pull out with the use of serial console servers? How many could you pull out by deploying KVM switch boxes?
2. Where do you work on broken machines? Is there an area that could be turned into a workbench area?
3. Which tools would you want in a cart in your datacenter?
4. Which supplies do you think you would want in the datacenter, and how many of each? What should the high and low supply levels be for each item?

5. Which spares would you want, and how many of each?
6. Which equipment do you have that is always “walking off”? Can you think of good parking spaces for it?

Part VI: Helpdesks and Support

Chapter 27. Customer Support

This chapter is a macro view of customer support and helpdesks: what they are, how to organize them, how to manage them, and so on. The details of how to handle a call to the helpdesk are covered in the next chapter.

A helpdesk is a place, real or virtual, where people can get answers to their computing questions, report problems, and request new services. It may be a physical desk that people walk to, or it may be a virtual helpdesk that people access electronically. The helpdesk is the primary mechanism for providing customer support.

Nothing is more important than an IT organization's helpdesk. It is the face of your organization. The helpdesk staff make the first impression on your customers and maintain your relationship, good or bad, with them. The helpdesk staff fix the daily problems that are part of living with modern computers and are the heroes who are called when customers have an emergency. A good helpdesk reflects well on your organization. The typical customer sees only the customer support portion of your organization and often assumes that this is your entire organization. Customers have no idea which back-office operations and infrastructure duties are also performed. In short, a helpdesk is for helping the customers. Don't forget the *help* in helpdesk.

The basics of running a helpdesk are first to simply have one and to ensure that it has a friendly face. The customer support center should have enough staff to support the traffic, a defined scope of coverage, processes for staff, an escalation process for when things go badly, and call-tracking software.

27.1 Having a Helpdesk

Every organization has a helpdesk. It may be physical, such as a walk-up counter, or virtual, such as by phone or email. Sometimes, the helpdesk function is unofficial, being the portion of each day spent directly helping customers.

Small SA teams, with just one or two people, frequently have no official helpdesk, but that situation isn't sustainable. As the organization grows, small SA teams become big SA teams, and big SA teams become enterprise organizations. Organizations sometimes don't realize that they need to institute a formal helpdesk until it is too late.

Earlier is better when it comes to setting up a formal helpdesk. The best time to do this is nine months before you realize that you should have done this six

months ago. Organizations without access to time-travel devices need other techniques. Organizations grow through planning, and developing a formal helpdesk should be part of that planning. If growth is slow, you can simply look for warning signs. One warning sign is when SAs start to notice that their group has grown to the point that communication problems are occurring.

Alternatively, SAs might notice that they aren't able to get project work done because they are continually being interrupted by customer requests. Typically, the SAs might decide that it would be better if, for example, one SA could be interrupted in the morning to handle customer issues and focus on project work in the afternoon, and the other SA could do the opposite. If you are considering such a structure, you are in the formative stage of adopting a formal helpdesk.

The transition from ad hoc to formal helpdesk can be uncomfortable to customers. SAs should expect this push-back and do their best to ease the transition. Communicating the new helpdesk procedures clearly is important.

Tell People About Changes

When an organization is establishing a formal helpdesk, whether physical or virtual, people must be told that this is being done. When Lumeta had fewer than ten people, most of them did their own computer support, and Tom intervened for more difficult problems. Eventually, the company grew and had three SAs, including one who was dedicated to PC-related problems and other employees. For all intents and purposes, this person was the helpdesk.

The customers didn't understand that the various SAs had specializations. This lack of understanding frustrated both the SAs, who felt pestered with inappropriate questions, and the customers, who were confused because every SA wasn't able to help in every situation. The problem was fixed when an email went out explaining which SAs to contact for which kinds of problems; the message was repeated at weekly staff meetings for two meetings in a row. You can prevent this kind of confusion by making such announcements as the change happens.

Helpdesks do not need to be purely physical locations but instead can be virtual. Problems can be reported and replies returned via email. Telephone, text-based, and audio chat systems can also be used.

Self-help systems are also popular but should not be considered a replacement for systems that involve human interaction. With the pervasiveness of the web,

there is no excuse not to have at least a simple repository of documentation for customers on such topics as how to get help or request service activation and solutions to common problems. A simple single-page web site with links to important documents, a wiki, or even a searchable blog (one post per FAQ) can be used. Web-based systems let customers help themselves. These systems can reduce the workload of helpdesk attendants but cannot provide interactive debugging or resolve workflow issues that require real-time interaction. There should be a phone number to call to report that the self-help system is down.

Ease Transitions

A 75-person department had a network decoupled from the centralized, corporate IT department. People who were more knowledgeable about the systems did more of the SA duties, and others did less. One semi-technical clerk on staff, named Karen (not her real name), took care of backups and was trained to do most installations and other semi-automated tasks. Nearly every bit of user documentation included the instruction, “Email Karen to get started.” As a result of business changes, the department was eventually required to use the centralized support that other departments in that building used. Customers were frustrated that instead of sending email to Karen, they had to send email to “help.” The personal touch had been lost. Rather than deal with the emotional issues head-on, management simply kept pushing people to use the new process.

Karen had a lot of political clout in the department because everyone knew her, and she could spread pessimism effectively if she chose to do so. Because she was not made part of the process but was shoe-horned into it, she felt that she was being pushed out. Karen eventually quit, and it took a couple of years for the new helpdesk system to be fully accepted by the customers.

The problems could have been prevented if the transition had been handled better, first by understanding what the customers were used to and then by integrating that culture into the new process.

27.2 Offering a Friendly Face

A helpdesk should have a friendly face. For a physical helpdesk, the interior design should be pleasant and welcoming. A web-based virtual helpdesk should be equally welcoming, which often means a design that uses soothing colors and readable fonts with the most commonly selected items at the top left of the first

page.

The faces of the staff members also should be welcoming and friendly, as should their personalities. Some people have personalities that are suited for customer service; others don't. That factor should be a consideration when hiring people for your staff. The tone set by the staff will reflect that set by the supervisor. A supervisor who yells at the staff will find staff yelling at customers. A good-natured supervisor who can laugh and is always friendly will attract similar staff, who will reflect such an attitude with customers. It is easier to build a reputation for being friendly initially than to restore a bad reputation. In short, if you are the supervisor, be the friendly person you want your staff to be. Be a role model.

27.3 Reflecting Corporate Culture

The look and feel of your helpdesk should reflect corporate culture. Often a helpdesk doesn't garner respect in a company when people working at the helpdesk buck the corporate culture. For example, a company that is very strict and formal may reflect this with strict dress codes and ways of conducting business, but the people at the helpdesk wear logo T-shirts and jeans, and a visitor hears the sound of a video game being played in the background. A little asking around will find that the helpdesk has a reputation of being a bunch of slackers, no matter how hard they work or how high the quality of the service they provide.

The opposite happens, too. When Tom was at Bell Labs, he worked in an area with very creative, free-thinking people. The dress code was simply "You must be dressed," and protocol was very relaxed. Tom's SA team was tasked with being a buffer between these researchers and corporate IT. Anytime these two groups of people interacted directly, it was like a bad sitcom episode.

Take the time to consider the culture and "look" of your helpdesk as compared to that of the customers they serve. Try to evolve to a culture that suits the customers served.

27.4 Having Enough Staff

A helpdesk can be helpful only if it has enough people to serve customers in a timely manner. Otherwise, people will look elsewhere for their support.

Sizing a helpdesk staff is very difficult because it changes from situation to situation. Universities often have thousands of students per helpdesk attendant. Corporate helpdesks sometimes have a higher ratio or sometimes a lower ratio.

In a commercial computer science research environment, the ratio is often 40:1, and the first-tier SAs have a similar skill level as second-tier SAs at other helpdesks, to meet the more highly technical level of questions. E-commerce sites usually have a separate helpdesk for internal questions and a “customer-facing” helpdesk to help resolve issues reported by paying customers. Depending on the services being offered, the ratio can be 10,000:1 or 1,000,000:1.

Ratios are a no-win situation. Management will always push to have a higher ratio; customers will always demand a lower ratio. You can always increase the ratio by providing less service to the customers, which usually costs the organization more because the customers spend time doing their own SA work inefficiently.

Rather than focus on customer-to-attendant ratios, it is better to focus on call-volume ratios and time-to-call completion. For example, you can monitor the rate at which customers receive busy signals or how long they wait to receive a response to their email or the number of minutes issues take to be resolved—minus, of course, time spent in “customer wait,” as described in [Section 28.6](#).

These metrics focus on issues that are more important to the customer. Customer-to-attendant ratios are an indirect measurement of benefit to the customers. In metric-based management, direct metrics are better.

Managing resources based on call volume also presents a more diverse set of potential solutions. Instead of one solution—headcount management—companies can invest in processes that let customers help themselves without the need for human intervention. For example, new automation can be created that empowers customers to do tasks that previously required privileged access, online documentation can be provided, new services can be provisioned automatically via web interfaces, and so on.

You need to have appropriate metrics to make decisions about improving processes. Metrics can reveal good candidates for new automation, documentation, or training for both SAs and customers. Metrics can reveal which processes are more effective, which are used heavily, or which are not used at all.

Case Study: Human Web Browsers

Making the customers more self-sufficient can backfire if not done correctly. One company established a web site to give easy access to all the documentation and FAQs that previously had been the domain of the

helpdesk staff. Upon examining the web site access logs, the management discovered an unexpected trend. At first, the web site was wildly successful. The hits to the site were coming from the customers. The volume of phone calls was reduced, and everything was happening as planned. However, by the third month, the logs indicated a new trend: The web site was seeing an increasing number of hits from the helpdesk itself.

Investigation showed that people had returned to calling the helpdesk and that the helpdesk attendants were reading answers off the web site. The helpdesk was acting as a human web browser! The situation was rectified when the helpdesk attendants were instructed to try to refer people to the appropriate web page rather than give answers directly. Customers were reminded to visit the web site before calling the helpdesk.

27.5 Defining Scope of Support

A helpdesk should have a policy defining the scope of support. This document explains what an SA group is and isn't responsible for. The components of scope are *what*, *who*, *where*, *when*, and *how long*:

- **What is being supported?** Only the PCs or the LAN itself? Are all PCs supported, no matter which OS is being used, or only certain OSs and certain revisions? Which applications are being supported? How are unsupported platforms handled?
- **Who will be supported?** A particular department, building, division, enterprise, university? What if a person has offices in multiple buildings, each with its own helpdesk? Are only people who pay supported? Are only people of a certain management level and higher (or lower) supported?
- **Where are the customers?** This question is similar to *who* if one is supporting, for example, everyone in a particular building or location. However, *where* also includes support of traveling customers, customers visiting third-party sites, customers performing demos at trade shows, and people working from home.
- **When is support provided?** Are the hours of operation 8 AM to 6 PM, Monday through Friday? How are things handled outside of these hours? Do people have to wait until the helpdesk reopens, or is there a mechanism to reach SAs at home? If there is no support in the off-hours, what should facilities management do if environmental alarms sound or if a fire occurs?
- **How long should the average request take to complete?** Certain

categories of requests should be instant; others will take longer. Establishing these goals sets expectations for the staff and customers. Customers expect everything to be immediate if they aren't told that certain tasks should be expected to take longer (see [Section 49.1.3](#)). A tiered structure might list certain things that are to be fast (5 minutes), slow (1 hour), and multiple days (requests for new service creation).

Having a written scope-of-support policy is one of the best gifts management can give to an SA group. Without it, the group either will become overworked from trying to do everything for customers, or will infuriate customers by not having time to help. If you are a manager, it is your responsibility to clearly communicate when it is okay to decline to help and when it isn't. This should be done in writing, with this policy then being communicated to all people who work for you. The same policy should also be publicly accessible on the internal IT web site to set expectations with customers.

Overworked SAs often do not have a written scope-of-support policy. SAs should create a written log of what they worked on for a week and write down all their open tasks. Showing this list to the SAs' managers often makes them realize that their people are spread too thin or are working on tasks that are not department priorities. Often, managers are surprised to find that their SAs are, in essence, doing their customers' jobs for them. Writing a policy defining the scope of support empowers the manager to clearly communicate priorities to the group, and the staff to communicate those same priorities to the customers.

When we investigate SA teams that have gained reputations for being unhelpful curmudgeons, we often discover that the root problem is a lack of written scope policy. Here, having a written policy sets a higher expectation for what the team's responsibilities are.

Without a written policy, the SAs are simply basing their responses on the accumulation of verbal edicts, and newer SAs simply follow the folklore they hear from their team members. New SAs, trying to be helpful, can undo a precedent without realizing it when they instead go against the ad hoc policy. Even worse, they may break something that shouldn't have been touched or step on a different department's toes.

Case Study: Wide Scope, Narrow Responsibility

Scope of support also means scope of responsibility. The New Jersey engineering division of a computer-aided design company was entirely in one building. The helpdesk's policy was that no question was

inappropriate, but the SAs had a sharply defined scope of responsibility. This worked because they understood where their responsibility ended. They had complete responsibility for certain issues: If someone reported a workstation problem, the SAs would fix it. For other issues, they would advocate on behalf of the customer: If the problem was with a WAN link that they didn't control, they would take responsibility for contacting the corporate network operations center and seeing that it got fixed. With other issues, they acted as a referral service: Someone reporting a burned-out light in the office was referred to facilities management, and the SAs would not take responsibility for seeing the issue to completion. They became a clearinghouse for requests and information.

To save money, the regional sales office was in the same building. The helpdesk was funded by the engineering division, not the sales office, and therefore the scope of responsibility for the sales staff was different. The helpdesk could refer the sales engineers to the proper documentation or suggest where they could get more training, but they could not be involved in helping to set up machines for demos or presentations. The sales staff used the central email server that was run by the engineering group, so email support was complete only if the salesperson was using the email client supported by the engineering group. However, because the support of the sales staff was free, there usually was a limit to how far the helpdesk staff would go to support the sales staff's requests.

Having a clearly defined scope of responsibility prevented the helpdesk from taking on more work than it could handle, yet let it provide an extremely friendly referral service. It also prevented the sales group from abusing its service by giving the helpdesk the ability to say "no" in a way that had management support.

A helpdesk must have a good process for dealing with requests about technologies that are out of scope. Of course, the helpdesk could simply state that the request is out of scope and refuse to help, but that is an unfriendly response. It is much better to clearly state the scope of what the helpdesk can and can't do for the person and then offer a little help but give a time limit before that assistance begins. For example, you might say, "We don't support systems with that video card, but I'll try my best for 30 minutes. If I can't fix it, you are on your own." You might spend 45 minutes on the problem and then politely tell the customer that you've reached your limit. The customer will appreciate your effort.

Setting Expectations When Working Outside of Job Scope

One of Jay Stiles's favorite stories takes place before DHCP made network configuration automatic. At one point, Jay worked in a place that had one set of technicians who installed network jacks in offices and a different set of technicians who configured the PC to connect it to the network. The customers often asked the first group of technicians to configure their PCs. The technicians weren't trained on how to do this but sometimes felt pressured and tried. Rarely were they successful, and more often than not, they corrupted some other configuration while trying to configure the PC. When they made this kind of mistake, their boss would be called and put in a difficult position: How do you answer a complaint about an employee who wasn't supposed to do a particular task and didn't do it correctly?

Later, the technicians learned that if they were asked to configure a PC, they should stop what they were doing, back away from the machine, and explain, "Well, that really isn't my job, but I happen to know a little about these PC computers, and I can give it a try. However, if I can't figure it out, I'm going to ask you to wait for the people who are supposed to do that kind of thing. Okay?" If they said those magic words, the result was very different. If they were successful, the customer was very happy, knowing that the technician had gone above the call of duty. If they weren't successful, the customer appreciated that the technician had tried.

The boss started receiving calls that were compliments: "The technician tried to configure my PC and couldn't, but I want to thank him for trying so hard!"

It's all in how you sell it.

27.6 Specifying How to Get Help

The companion to the scope-of-support document is a document that specifies how to get help: by phone, email, a ticket system, and so on. Certain types of requests may be directed to certain departments, or a unified helpdesk might be the single point of contact that forwards requests as appropriate to individual departments.

Such a document should be a few hundred words at most. Ideally, a shorter version that can fit on a sticker should be put on every new PC deployed. The same image can appear on default Windows background wallpaper images:

“CompanyName IT helpdesk: [phone number] [email address] [web site].”

This is one of the most important things IT management can do to help the staff save time. If customers have not been given clear directions on the proper way to get help, they will contact SAs directly, interrupting them at inappropriate times, and making it impossible to get larger projects done. Even worse, SAs may be contacted at home!

27.7 Defining Processes for Staff

Helpdesk staff should have well-defined processes to follow. In a smaller environment, this is not as important, because the processes are more ad hoc or are undocumented because they are being used by the people who built them. However, for a large organization, the processes must be well documented.

Very large helpdesks use *scripts* as part of their training. Every service supported has an associated flow of dialogue to follow to support that service. For example, the script for someone calling to request remote access service captures the appropriate information and tells the operator what to do, be it enable remote access directly or forward the request to the appropriate service organization. The script for a request to reset a password would, for security reasons, require callers to prove who they are, possibly by knowing a unique piece of personal information, before a new password would be set.

[Chapter 28](#), “[Handling an Incident Report](#),” discusses a formal process that helpdesk attendants can use to process individual trouble reports.

27.8 Establishing an Escalation Process

Escalation is a process by which an issue is moved from the current staff person to someone with more expertise. The first line of operators should be able to handle 80 percent to 90 percent of all calls and escalate the remaining calls to a second tier of support. The people at this second tier may have more experience, more training, and, possibly, other responsibilities. Larger organizations may have four or more tiers; the higher tiers may include the people who built or currently maintain the service in question.

It is common to have a policy that the first tier of support should escalate all calls that get to the 15-minute mark. This has a carryover effect in that the second-tier people, who may be responsible for project-oriented work, now have less time for projects. This situation can be alleviated by designating one second-tier person to sit with the first-tier people each week—that is, to make the helpdesk his or her project for the week. Although upper-tier staff people usually

dislike this policy, a sizable organization will assign people to this duty only once every 6 weeks or so. A side benefit of this strategy is that the first-tier staff will learn from the second-tier person. The second-tier person will also get a better understanding of the kinds of issues coming in to the helpdesk, which will help determine which new projects will be the most help to the first tier and the customers.

The escalation process is also what customers use when they are dissatisfied with the support they are receiving. One hopes that this happens as infrequently as possible, but inevitably someone will want to talk to a manager. The helpdesk should be prepared for this kind of request. Large numbers of calls being escalated to the second tier is a warning sign of a larger, systemic problem. Usually, it indicates that the first-tier staff people need more training or do not have the tools to do their job properly. If large numbers of calls are escalated to management, there may be systemic problems with the support the helpdesk is providing.

Escalating for Results

Escalation should not exist simply to pacify angry customers. One small ISP's helpdesk often receives calls from angry individuals who demand to speak to a manager. In that case, the person hands the phone to the person on his or her left, who then claims to be the manager. Although this practice works in the short term or when business growth is exploding, it is not a sustainable way of maintaining a helpdesk.

27.9 Defining “Emergency” in Writing

Often, SAs are overloaded because every customer claims to have an emergency that requires immediate attention. SAs may feel that customers are using this claim to boss them around, which decreases morale and increases stress levels.

Having a written policy empowers SAs to know when to push back and gives them a document to point to when they need it. If the customer still disagrees with this assessment, the SA can pass the issue up to someone in management, who can make the decision. This lets the SA focus on technical duties and lets management focus on setting priorities and providing resources.

Every company should be able to define what constitutes an emergency. At a factory, an emergency is anything that stops the assembly line. At a web-based service or ISP, an emergency might be anything that will prevent the service from meeting an SLA. A sales organization might define an emergency as

anything that will prevent a demo from happening, end-of-quarter revenues from being booked, or commissions from being processed. At a teaching institution, which has tightly scheduled lecture times that cannot be simply moved or delayed owing to a system outage, an emergency might be anything that would disrupt scheduled technology-dependent lectures, as well as other matters related to the seasons of the school year: new-student arrival, exam deadlines, grade publication, graduation, new-student recruitment deadlines, and so on.

It may sound simple, but writing a definition of what constitutes an emergency can become a political battle. It may be part of a larger SLA document, or a stand-alone policy written to help helpdesk personnel make the right decision. This definition is often included in an escalation policy.

Planning Well

When Tom was in college, the business office put up a poster that said, “Bad planning on your part does not constitute an emergency.” The students were offended, but the sign stayed. The truth is that this statement could be one of the most important lessons the university could teach its students before they hit the real world.

27.10 Supplying Request-Tracking Software

Every helpdesk needs some kind of software to help it manage requests. The alternative is a collection of notes written on scraps of paper. Although it is simple in the beginning and sufficient for environments with one or two SAs, a system based on notes on paper doesn’t scale. Requests get lost, and management has no ability to oversee the process to better allocate resources. Those are the first qualities that you need in helpdesk software. As a helpdesk grows, software can help in other areas. The scripts mentioned in [Section 27.7](#) can be displayed automatically and can be made “smart” by being part of the information-gathering process rather than simply a static screen.

Helpdesk software should permit some kind of priority to be assigned to tickets. This not only helps meet customer expectations but also helps SAs manage their time. An SA should be able to easily list the top-priority issues that have been assigned to him or her.

Another important aspect of helpdesk software is that it collects logs about which kinds of requests are made and by whom. Statistical analysis of such logs can be useful in managing the helpdesk. However, if the software doesn’t capture that information, one can’t gain the benefits of such statistics. This often

happens when there is a lot of walk-up and phone traffic. In such cases, it can be useful for the software to permit logging common questions or issues with one click. Caller ID can be used to populate fields with the caller's information.

Helpdesk software can also automate the collection of data on customer satisfaction. Every day, the software can select a random sample of the previous day's customers and survey them about the service they received.

Case Study: From Good to Bad

A software company with approximately 1,500 employees was using an enhanced version of a freely available call-tracking system. It was very simple to use, with a few different interfaces, the most popular of which was the email interface. The system tracked the customer, department, category, status, who the call was assigned to, how much time had been spent on it and by whom, priority, due date, and so on. Custom scripts produced metrics that management could use to track how things were going. Customers could use a web interface to see the history of their calls and all the other associated fields. Customers could also look at the call queue for the person to whom the call was assigned and see where it was in the person's priority list. Although the system wasn't glitzy, everyone was comfortable with it and could get the needed information out of it.

The management information systems (MIS) group, which provided support for databases and the applications that ran on top of them, and which was not a part of the SA group, was commissioned to build a new call-tracking system for the customer-support center. The management chain of that group expanded the scope of the project to make this into one unified call-tracking system that would also be used by the operations group, MIS, and the SA group. Neither the operations group nor MIS had a call-tracking system, so they were designing a system without knowing what makes a good system. No one in the SA group was told of the project, so its needs and those of its customers were not taken into consideration in the design.

The graphical user interface (GUI) system that resulted had no email interface and no command-line interface. Creating a new call involved bringing up ten different windows, each a pop-up that was slow to appear and required mouse movements to chase to the right place on the screen. Updating a call required five or six different pop-ups. It was impossibly slow for many SAs who dialed in over modems to use the system from home. The system didn't work for anyone with a Mac or Unix system

since the client was made only for Microsoft Windows. It frequently took longer to open a trouble ticket than it took to solve the problem, so the numerous small calls were no longer tracked. What was once a quick email process had become a ten-minute endeavor. Several SAs went back to tracking projects on pieces of paper or in their heads.

Customers complained because they could no longer see the status of their calls or where those calls were in the priority queues. They also complained because they couldn't open a ticket via email any more and because the new system sent them far too much email whenever a small field in the call was changed. All of these complaints were predicted by the SA group when they were suddenly presented with the new system that they would have to start using, but it was too late to change anything.

The system was supposed to provide better tools for producing metrics. However, because a lot of the data was no longer entered into the system, it clearly didn't, even though the tools it provided for metrics may have been better.

It is critical that helpdesk software match the workflow of the people who use it. If one ticket is opened per week, it is reasonable for the creation of a ticket to take a long time. However, if you expect hundreds of tickets per day, initiating the new ticket should be almost instantaneous, such as sending email. Do not use helpdesk software to introduce radical new workflow concepts. However, if some requests require approvals from the line manager or someone responsible for a particular application, the helpdesk software should be able to get those approvals before passing the ticket to the SA team.

Choosing helpdesk software is not an easy process. Most software will need a lot of customizing for your environment. When you decide to invest in helpdesk software, you need to be prepared to invest in the customizations as well, so that the SAs can use it effectively. If it is a burden to use, they will not use it or will use it only for large projects.

27.11 Statistical Improvements

Many sophisticated statistics can be gathered about a helpdesk. For example, you can monitor the rate of escalations to determine where more training is needed. However, when dealing with upper management for budgeting and planning purposes, historical statistics become much more valuable. You can make a better case for your budget if you can show multiyear trends of customer

growth, call volume, types of calls, technologies used, services provided, and customer satisfaction. When you are asked to support a new technology or service, you can use past data to predict what the support costs may be.

The value of statistics increases as the organization grows, because the management becomes less directly involved in the work being done. It is often difficult to collect statistics in small organizations, because practices are often less automated and can't be instrumented to collect data. As an organization grows, statistics are easier to collect, and it becomes more important that they be collected.

Identifying the Top Requesters

SAs are usually detail-oriented people. Statistics gloss over the details to find general trends. As a consequence, SAs are often not the best people to generate statistics about helpdesk requests.

When asked to generate statistics from a helpdesk request system, Tom was stymied. How could useful statistics be generated if the organization didn't first change the software to ask SAs how many minutes of work a ticket had required, provide a classification of the type of work, and tell which department the work was for? Then, after a year of collecting this data, he could produce excellent charts and graphs for a complete analysis.

The problem was that his boss wanted results in 24 hours. His boss suggested a very different tack: The entire process could be simplified if one assumed that all requests took the same amount of time. Tom was horrified but was eventually convinced that, on average, all their requests took an average amount of time.

His boss then suggested a database query that would determine who generated the most request tickets. It turned out that three customers had created 10 percent of all tickets in the past year.

Rather than meticulously classifying all tickets, Tom and his boss looked at a few from each of the three top customers and spoke with the SAs who had helped them the most. They learned that one person was continually requesting help with a product that was out of scope; the SAs were nicely helping the person anyway. The manager put his foot down and told the SAs that the product was intentionally not supported because it was too difficult to do so. Future requests for help were to be denied. The manager visited the customer to explain that he had to either become

self-sufficient or permit the helpdesk to help him convert to the corporate standard product. The helpdesk didn't usually provide this kind of conversion service but would be willing to do so in this case.

The second customer was asking a lot of basic questions. Tom's manager spoke to the customer's manager about getting the person more training. The person's manager was aghast at the level of help this person had needed, and took care of the issue.

The third customer was, to put it politely, getting an SA to do the person's job. This issue was raised to the person's manager, who took care of it.

In the absence of any statistics, some basic rough estimates were still extremely useful. This one technique was able to eliminate approximately 10 percent of all the tickets entered into the system. That's a big improvement!

27.12 After-Hours and 24/7 Coverage

As computers become critical to an ever-expanding list of business processes, customers are asking for 24/7 coverage more often. Although a full three-shift staff may be required in some organizations, some very simple ways to provide 24/7 coverage are not as expensive.

For example, you can set up a voicemail box that alerts a pager when new messages arrive. The pager can be rotated among various staff members. The responsibility of the staff person may not be to fix the problem but simply to alert the appropriate person or keep calling various people until someone is found. This requires all staff to have a list of everyone's home phone number.

A variation on this technique is to have all managers of the customer groups know the home phone number of the helpdesk's supervisor, who then takes responsibility for calling SAs in turn until one is found. This has the benefit of limiting personal information to fewer people but can wear down a helpdesk supervisor and doesn't take into account the supervisor's vacations. However, local solutions can be found, such as rotating this duty among a couple of supervisors.

After-Hours Coverage at T. J. Watson

The modern equivalent of a researcher's laboratory catching fire at night is a major file server being down. In turn, you can treat this issue the same

way other industries treat fire alarms. Security personnel have a call list in case of alarms, fires, and so on. You can send alerts to the security personnel, and give them a call list for IT problems. They can start at the top of the list and keep calling numbers until they find someone. Depending on the issue, that person may tell the security guard whom to call. Of course, this method means that the person whom is at the top of the list may never get a good night's sleep!

In the late 1990s, IBM's T. J. Watson facility extended the process for handling fire and other alarms to the major computer systems. If a major computer was down, customers could call the security desk and report the issue. The security guards had a separate list of people to call for a computer-related problem.

No matter how SAs are contacted after hours, the person must be compensated. Some organizations have a salary incentive for oncall time, equivalent to a fraction of the employee's salary and time and a half if the person is called. Other organizations issue compensation time either officially or unofficially. Comp time allows the employee to take off that much time—or 1.5 times that amount, in some cases—without claiming it against vacation time.

27.13 Better Advertising for the Helpdesk

Defining your policies and providing announcements online is nice, but rarely will anyone seek them out. In this section, we talk about getting your policies and announcements “out there” and understood.

With the Internet, it is easy to make all policies accessible to all customers. There is no excuse for not providing this material online. However, you must get customers to that web site. Some SA organizations choose to have a portal web site that is the gateway to all their services, policies, and documentation. Because customers will already be using the site to receive information that is important to them, they also will know to go there for information that is relevant to the helpdesk.

Pick the right message. Talk with customers to find out what is important to them. It's difficult to get people to read something that isn't important to them. They may not care that server3 will be down during the weekend, but knowing that the database stored on server3 won't be accessible all weekend will draw their attention.

New policies can be emailed to customers or sent via paper memo if they are

particularly critical. Portals can highlight a “policy of the month.” If the message will benefit from repetition, put posters in appropriate places. A physical helpdesk should have its hours posted at all entrances. People spend a lot of time staring at the wall while waiting for their turn; fill those blank walls with the message you want them to remember. Posters that say “Change your password every 30 days!” or “Server3 is being decommissioned on May 1” give good advice and warn of upcoming changes.

Messages are most effective when received at the right time. If server3 is being decommissioned in a month, inform people of that fact every time they use server3.

27.14 Different Helpdesks for Different Needs

When an organization grows, it may make sense to have two separate helpdesks: one for requesting new services and another for reporting problems that arise after the service has been successfully enabled. Often, a third group deals with installing the new service, especially if it requires physical work. This third group may be an internal helpdesk that installers all over the organization can call to escalate installation problems. It is not uncommon, though, for this third group to be the second tier of one of the other helpdesks.

The benefit of dividing the helpdesk this way is that the two or three groups can be placed under different supervisors. A supervisor can effectively manage only a certain number of people. This division of labor makes it clear where to place various supervisors. They should all report to the same manager to make sure that communication happens and finger-pointing doesn't.

Another benefit is that the different groups can be separately trained for the different skills required for their task. This tends to be less expensive than hiring people who are experienced enough to be able to do all the tasks.

Providing a provisioning service is a process that should be the same for all customers. The initial collection of data can be done by someone trained to ask the right questions. This person may have more sales experience than the other staff. Solving installation problems is a highly technical issue but has a narrow focus, and training can be customized to those issues. The separate helpdesk for reporting problems requires people with wider technical experience and background. This division of labor is critical to scaling the organization to very large sizes.

A provisioning helpdesk should have a web-based request system. This can save a lot of data entry and prevent mistakes. Having the customer enter the data

reduces the number of typos that can occur when a third party enters data. The system can check for common errors and reject inconsistent or conflicting requests. Phone requests can be replaced by phone assistance for people who need help filling out the web form.

You do not need to create an entirely new software system to support this kind of functionality. The form data can simply be submitted into the regular request-tracking system, which can then handle the workflow.

27.15 Summary

For many customers, the helpdesk is the only way they interact with the IT organization. There may be dozens of subteams behind it, but the helpdesk is the only part that people see. Often they assume it is the entire organization.

The helpdesk is the “face” of the IT organization, so make it a happy, friendly one, no matter whether it is physical or virtual. Properly sizing the helpdesk is important and affects not only your budget but also customer satisfaction. In planning your helpdesk, you must define what is supported, who is supported, where the helpdesk is located, when you provide support, and how long customers should expect an average call to last. Constructing accurate budget and staffing plans is made easier by collecting the statistics mentioned in [Section 27.10](#).

Processes should be defined for staff to follow regarding how they provide support for various services and how issues are escalated. Software must be used to collect statistics on all calls and to track issues that last longer than a single call.

Once those functions are established, a helpdesk can grow in other areas. After-hours coverage can be instituted; policies can be better advertised; and, with high growth, the helpdesk can be split into separate groups for new-service provisioning and trouble reporting.

We discussed a lot of topics in this chapter, and every issue in some way touched on communication. The helpdesk is how customers communicate with your organization, yet it is often the role of the helpdesk to communicate to customers how, when, and why things are done. Such communication can determine whether IT personnel are perceived as friendly or unfriendly. The statistics that are collected help communicate to management the needs of the organization during planning cycles. Escalation procedures keep the communication flowing when things stall. Having a written after-hours support policy sets expectations with customers and prevents the frustration of

unexpected calls late at night.

Exercises

1. Describe your helpdesk staff structure.
2. How many attendants does your helpdesk have at any given moment?
How is that number selected?
3. Which helpdesk attendant in your organization is perceived as the least friendly by the customers? How do you know? How would you help that person improve?
4. How hands-on are you with your helpdesk? Which statistics do you use to manage it, and which statistics are given to your management? If you were one step less hands-on, which new statistics would you need then?
5. Figure out which customers generate the top 10 percent—or 1 percent for larger sites—of all requests. Which trends do you see in those requests, and how can this information be used to improve your helpdesk?
6. Report a problem tonight at 10 PM. Describe how well it went.

Chapter 28. Handling an Incident Report

This chapter is about handling incident reports. In a broad sense, requests from customers are either a service request, an incident report, or a complaint. We can further explain this distinction by way of examples: A service request is someone requesting a new PC. An incident report is when someone reports that his or her PC isn't working and needs to be fixed. A complaint is when someone reports that he or she is unhappy waiting so long for the PC to be fixed.

Handling an incident report can be difficult. We must interview the person to understand the problem, identify the issue, and follow through until the issue is resolved.

It may surprise you to learn that this process is similar to the sales process. Sales involves understanding a person's needs, proposing solutions, and working with the customer to provide a solution. We can learn a lot from the history of modern sales.

At the close of World War II, the United States found itself with a huge excess of manufacturing capacity. As a result, companies started producing hundreds of new products, giving households and businesses unprecedented choices. Thousands of returning service people found jobs selling these new products. All these factors combined to produce a new era for the U.S. economy.

Along with the new choices came greater competition. Companies found that it was no longer sufficient merely to have a large sales force; a *good* sales force was needed. They started to ask what makes high-performing salespeople different from the others.

Industry encouraged business schools to increase their study of the sales process. Research showed that the better salespeople—whether or not they realized it—used a specific, structured method involving specific phases or steps. Mediocre salespeople deviated from these phases in varying ways or performed certain phases badly. The low performers had little or no consistency in their methods.

The methods, now identified, could be taught. Thus, sales skills went from an intuitive process to a formal process with well-defined parts. Previous sales training had consisted mostly of explaining the product's features and qualities. Subsequently, training included exploration of the selling process itself.

This deconstruction of the process into individual steps permitted further examination and therefore further improvement. Each step could be studied,

measured, taught, practiced, and so on. Focus was improved because a single step could be studied in isolation. Also, the entire flow of steps could be studied, in a holistic approach.

More than likely, if anyone explained the structured process to the high-performing salespeople, it would sound strange. To them, it came naturally. To the beginners, however, this framework gave structure to a process they were learning. After they mastered it, they could modify or customize it for their situation. Without first learning one structured system, it is difficult to get to the place where you need to be to invent your own.

In the 1990s, system administration began a similar journey. Previously, it was a craft or an art practiced by few people. With the explosive growth of corporate computing and intranet/Internet applications, the demand for SAs similarly skyrocketed. A flood of new SAs arrived to meet this need. The quality of their work varied. Training often took the form of teaching particular product features. Other training methods included exploring manuals and documentation, trial by fire, and training by social and professional institutions.

System administration needed to mature in ways similar to how the sales process matured. The late 1990s saw an increase in the academic study of system administration ([Burgess 2000](#)). In fact, this book's inspiration comes from that same need to provide training founded on non-platform-specific themes, principles, and theoretical models rather than on specific details about particular technologies, vendors, and products ([Limoncelli, Hogan & Chalup 2017](#)).

28.1 Process Overview

SAs spend much of their time responding to support requests from customers. In this chapter, we present a structured process defining how customer requests are gathered, evaluated, fixed, and verified. This process is based on a paper by Tom ([Limoncelli 1999](#)). Responding to customer requests is a more specific task than the general issues that surround running a helpdesk.

Customer requests are the trouble tickets, calls, problem reports, or whatever your site calls them. These requests may take the form “I can’t print,” “The network is slow,” or “A program that compiled yesterday won’t compile anymore.”

SAs perform many tasks, but often customers see only the parts that involve responding to their requests, not all the back-office work. (They shouldn’t have to see it.) Therefore, how well you respond to customer requests is critical to maintaining your organization’s reputation.

The method for processing these customer requests has nine steps, which can be grouped into four phases:

- Phase A: The Greeting (“Hello!”)
 - Step 1: The Greeting
- Phase B: Problem Identification (“What’s wrong?”)
 - Step 2: Problem Classification
 - Step 3: Problem Statement
 - Step 4: Problem Verification
- Phase C: Planning and Execution (“Fix it.”)
 - Step 5: Solution Proposals
 - Step 6: Solution Selection
 - Step 7: Execution
- Phase D: Verification (“Verify it.”)
 - Step 8: Craft Verification
 - Step 9: Customer Verification/Closing

This method gives structure to what is, for newer SAs, a more haphazard process. It helps SAs solve problems more efficiently by keeping them focused and helps them avoid mistakes. It introduces a common set of terminology that, when used by the entire SA team, increases the ability to communicate within the group.

This tool does not bestow any additional technical expertise on the people using it, but it may help the junior people gain insight into how the senior SAs approach problem solving. Creativity, experience, the right resources and tools, and personal and external management are still important.

If customers understand this model, they can become more skilled in getting the help they desire. They will be prepared with the right information and can nudge the SA through the process, if necessary. As [Figure 28.1](#) shows, the phases deal with

- A.** Reporting the problem
- B.** Identifying the problem
- C.** Planning and executing a solution
- D.** Verifying that the problem resolution is complete

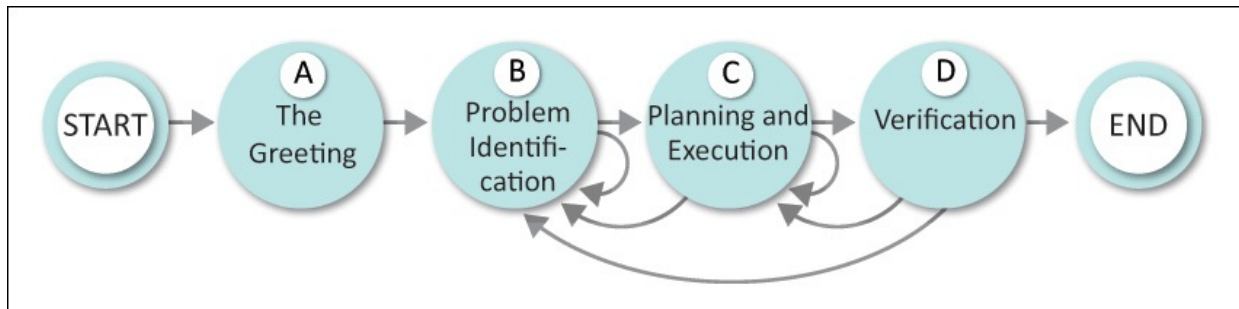


Figure 28.1: General flow of problem solving

Sometimes of these steps may be iterated as required. For example, during step 4 (problem verification), the SA may realize that the issue has been misclassified and that it is necessary to return to step 2 (problem classification). This kind of a-ha moment can happen at any step and requires returning to a previous step.

Trouble-Tracking Software

We cannot overemphasize the importance of using a software package to track problem reports. In the 1980s and early 1990s, SAs rarely used software to track such requests. Today, however, installing such software is profoundly transformational, affecting your ability to manage your time and to deliver consistent results to customers. If you find a site that has no trouble-tracking software, simply install whatever you were comfortable with at a previous site or software that has an Internet mailing list of active supporters.

28.2 Phase A—Step 1: The Greeting

Phase A has only one deceptively simple step: Issues are solicited from the customers ([Figure 28.2](#)). This step includes everything related to how the customer's request is solicited—from someone saying, "How may I help you?" on the phone to a web site that collects problem reports. Step 1 should welcome the customer to the system and start the process on a positive, friendly, helpful note.

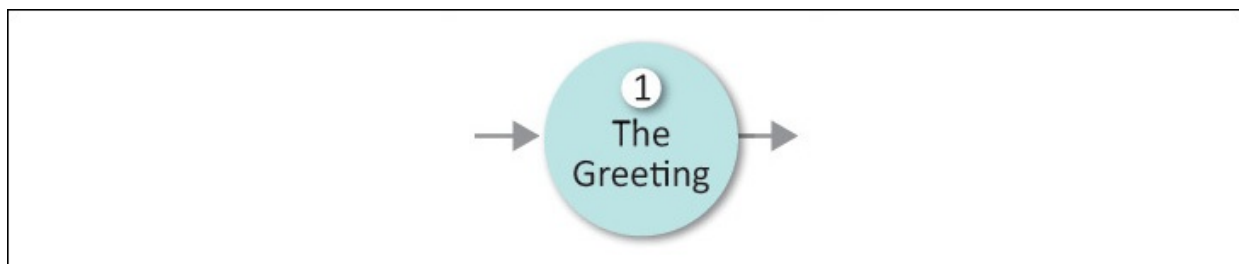


Figure 28.2: Phase A: Hello!

The person or system that responds to the requests is called a **greeter**. Greeters may be people in a physical helpdesk, on the phone, or accessible via email or other instant technology; a phone-response system; or even a web form that takes the data. Multiple ways to collect reports are needed for easy and reliable access, ensuring that the customer can report the problem.

Sometimes, problems are reported by automated means rather than by humans. For example, network-monitoring tools, such as Nagios, Big Brother, Bosun, HP OpenView, and Tivoli, can notify SAs that a problem is occurring. The greeting process is the same, although some of the steps may be expedited by the tool.

Every site and every customer is different. The appropriate way to report issues is different for every part of every organization. Is the customer local or remote? Is the customer experienced or new? Is the technology being supported complicated or simple? These questions can help when you select which greeters to use.

Finding Greeters

How do customers know how to find help? Advertise the available greeters by signs in hallways, newsletters, stickers on computers or phones, and even banner advertisements on internal web pages. The best place is where customers' eyes are already looking: on a sticker on their PC, in an error message, and so on.

Although this list certainly isn't complete, greeters typically include email, phone, walk-up helpdesk, visiting the SA's office, submission via web, submission via custom application, and report by automated monitoring system.

28.3 Phase B: Problem Identification

Phase B is focused on classifying the problem and recording and verifying it ([Figure 28.3](#)).

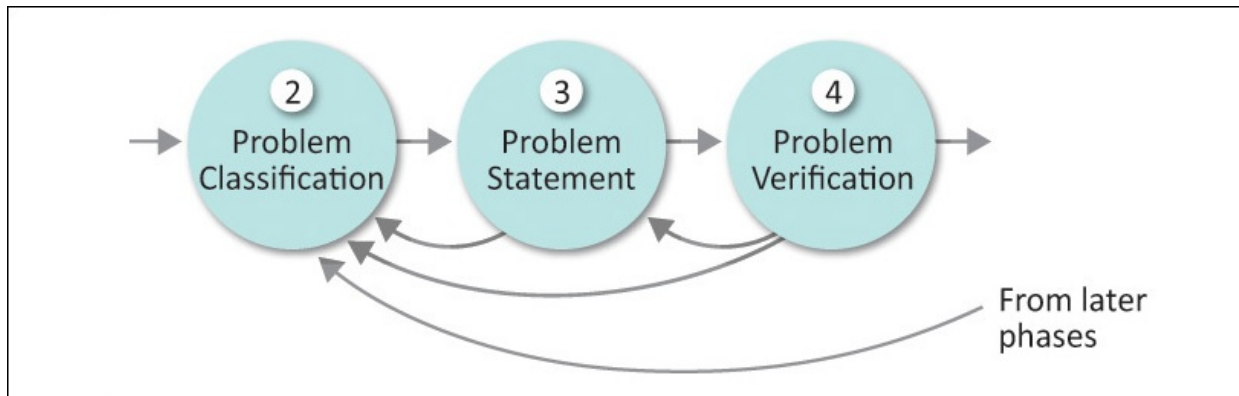


Figure 28.3: Phase B: What's wrong?

28.3.1 Step 2: Problem Classification

In step 2, the request is classified to determine who should handle it. This **classifier** role may be performed by a human or may be automated. For example, at a walk-up helpdesk, staff might listen to the problem description to determine its classification. A phone-response system might ask the user to press 1 for PC problems, 2 for network problems, and so on. If certain SAs help certain customer groups, their requests may be automatically forwarded, based on the requester's email address, a manually entered employee ID number, or the phone caller's caller ID information.

When the process is manual, a human must be responsible for classifying the problem from the description or asking the customer more questions. A formal decision tree may be used to determine the right classification. Formal processes and decision trees are used to improve results in other fields. For example, a hospital found that a patient's odds of surviving a heart attack improved when guesswork was eliminated by giving the doctors a formal process to follow ([Gladwell 2007](#)).

SAs need to ask more questions when they aren't as familiar with the customer's environment. This is often the case at the helpdesks of e-commerce sites or extremely large corporate helpdesks.

No matter how the classification is performed, the customer should be told how the request is classified, creating a feedback loop that can detect mistakes. For example, if a classifier tells a customer, "This sounds like a printing problem; I'm assigning this issue to someone from our printer support group," the customer stays involved in the process. The customer might also point out that the problem is more pervasive than simply printing, leading to a different classification.

If a phone-response system is used, the customer has classified the request already. However, a customer may not be the best person to make this decision. The next person who speaks with the customer should be prepared to validate the customer's choice in a way that is not insulting. If the customer has misclassified the request, this error should be remedied in a polite manner. The best way to do so is to give the customer the correct phone number to call or button to press, and then the SA should transfer the call to the right number. Some companies do one or the other, but doing both is better.

When asking a customer to classify the problem, the choices presented must be carefully constructed and revised over time. You should gather statistics to detect mismatches between customers' perceptions of what the classifications mean and what you intended them to mean, or at least you should monitor for customer complaints.

Marketing-Driven Customer Support

Phone menus should use terminology that the customers expect to hear. A large network equipment manufacturer once had its phone menu based on the marketing terminology that segmented its product lines rather than on the technical terminology that most of its customers used. This caused no end of confusion because the marketing terminology had little basis in reality from the typical technician's point of view. It was particularly confusing for customers of any company that was acquired by this company, because the acquired company's products were reclassified into marketing terms unfamiliar to the acquired company's customers.

Many requests may be transferred or eliminated at this stage. A customer requesting a new feature should be transferred to the appropriate group that handles requests for features. If the request is outside the domain of work done by the support group, the customer might be referred to another department. If the request is against policy and therefore must be denied, the issue may be escalated to management if the customer disagrees with the decision. For this reason, it is important to have a well-defined scope of service and a process for requesting new services.

Classifying problems can get complex. At very large sites, you are more likely to find yourself acting on behalf of your customer, coordinating between departments or even the helpdesks of different departments! Complicated problems that involve network, application, and server issues can require the helpdesk attendant to juggle conversations with three or more organizations.

Navigating such a twisty maze of passages for the customer is a valuable service you can provide.

28.3.2 Step 3: Problem Statement

In step 3, the customer states the problem in full detail, and the **recorder** takes this information down. Often, the recorder is also the classifier. The skill required by the recorder in this step is the ability to listen and to ask the right questions to draw out the necessary information from the customer. The recorder extracts the relevant details and records them.

A problem statement describes the problem being reported and records enough clues to reproduce and fix the problem. A bad problem statement is vague or incomplete. A good problem statement is complete and identifies all hardware and software involved, as well as their locations, the last time they worked, and so on. Sometimes, not all that information is appropriate or available.

An example of a good problem statement is this: “PC talpc.example.com (a PC running Windows 7) located in room 301 cannot print from MS Word 2010 to printer ‘rainbow,’ the color printer located in room 314. It worked fine yesterday. It can print to other printers. The customer does not know whether other computers are having this problem.”

Certain classes of problems can be completely stated in simple ways. Internet routing problems can best be reported by listing two IP addresses that cannot ping each other but can communicate to other hosts; including a traceroute from each host to the other, if possible, helps considerably in pinpointing the problem.

More information is usually better than less. However, customers may be annoyed when required to provide information that is obviously superfluous, such as which OS they are using when the issue is a smoking monitor. Yet we continually see web-based trouble-reporting systems requiring that no fields be left blank.

It is unreasonable to expect problem statements from customers to be complete. Customers require assistance. The problem statement cited earlier comes from a real example in which a customer sent an SA email that simply stated, “Help! I can’t print.” That is about as ambiguous and incomplete as a request can be. A reply was sent asking, “To which printer? Which PC? Which application?”

The customer’s reply included a statement of frustration. “I need to print these slides by 3 PM. I’m flying to a conference!” At that point, the SA abandoned email and used the telephone. This permitted a faster back-and-forth between the

customer and the classifier. No matter the medium, it is important that this dialogue take place and that the final result be reported to the customer.

Sometimes, the recorder can perform a fast loop through the next steps to accelerate the process. The recorder might find out whether the device is plugged in, whether the person has checked the manual, and so on. However, such questions as “Is it plugged in?” and “Have you checked the manual?” make customers defensive. They have only two possible answers, and only one of them is clearly right. Avoid making customers feel compelled to lie. Instead, ask which outlet it’s plugged into; ask for confirmation, while you’re on the phone, that the cable is firmly seated at both ends. Tell the customer that you’ve checked the manual and that, for future reference, the answer is on page 9, if the problem comes up again.

You also should make sure to never make the customer feel like an idiot. Hearing that a helpdesk attendant informed a customer that “an eight-year-old would understand” what he is explaining is cringe-worthy. Instead, reassure customers that they’ll get better at using computers as they gain experience.

Helping the Customer Save Face

Finding ways to let customers save face can be very beneficial. An SA in London once took a call from a person who was in a panic about not being able to print his monthly reports on a printer used almost exclusively for this purpose. After a series of tests, the SA found that the printer was unplugged. He explained to the customer that the cleaning staff must have unplugged the printer when needing an outlet for the vacuum.

A month later, the same person called the SA with the same problem and pointed out that this time, he had checked to make sure that the printer was plugged in. Investigation showed that it had been turned off at the switch. The customer felt so embarrassed that he’d missed such obvious faults both times that he bought the SA a beer. After that, the problem never occurred again.

By not criticizing the person and by keeping him in the loop about what the problem was, the customer learned to solve his own problems, the two colleagues remained on friendly terms, and the SA got a free beer.

Flexibility is important. In the previous example, the customer indicated that there was an urgent need to have the slides printed prior to a flight. Here, it

might be appropriate to suggest using a different printer that is known to be working rather than fixing the problem right now. This accelerates the process, which is important for an urgent problem.

Large sites often have different people recording requests and executing them. This added handoff introduces a challenge because the recorder may not have the direct experience required to know exactly what to record. In that case, it is prudent to have preplanned sets of data to gather for various situations. For example, if the customer is reporting a network problem, the problem statement must include an IP address, the room number of the machine that is not working, and which particular thing the person is trying to do over the network that is not working. If the problem relates to printing, the SA should record the name of the printer, the computer being used, and the application generating the print job.

It can be useful if your trouble-ticket software records different information, depending on how the problem has been classified.

28.3.3 Step 4: Problem Verification

In step 4, the SA takes on the **reproducer** role and tries to duplicate the problem. If the problem cannot be reproduced, perhaps it is not being properly communicated, and the SA must return to step 3. If the problem is intermittent, this process becomes more complicated but not impossible to follow.

Nothing gives you a better understanding of the problem than seeing it in action. This is the single most important reason for doing problem verification, yet naive SAs skip it all the time. If you do not verify the problem, you may work on it for hours before realizing that you aren't even working on the right issue. Often, the customer's description is misleading. A customer who doesn't have the technical knowledge to accurately describe the problem can send you on a wild goose chase. Just think about all the times you've tried to help someone over the phone, failed, and then visited the person. One look at the person's screen and you say, "Oh! That's a totally different problem!" And a few keystrokes later, the problem is fixed. What happened was that you weren't able to reproduce the problem locally, so you couldn't see the whole problem and therefore couldn't figure out the real solution.

It is critical that the method used to reproduce the problem be recorded for later repetition in step 8. Encapsulating the test in a script or a batch file will make verification easier. One of the benefits of command-driven systems, such as Unix, is the ease with which such a sequence of steps can be automated. GUIs make this phase more difficult when there is no way to automate or encapsulate

the test.

The scope of the verification procedure must not be too narrowly focused or too wide or misdirected. If the tests are too narrow, the entire problem may not be fixed. If the tests are too wide, the SA may waste time chasing non-issues.

It is possible that the focus is misdirected. Another, unrelated problem in the environment may be discovered while trying to repeat the customer's reported problem. Some problems can exist in an environment without being reported or without affecting users. It can be frustrating for both the SA and the customer if many unrelated problems are discovered and fixed along the way to resolving an issue. If an unrelated problem that is not in the critical path is discovered, it should be recorded so that it can be fixed in the future. However, determining whether it is in the critical path is difficult, so fixing it may be valuable. Alternatively, it may be a distraction or may change the system enough to make debugging difficult.

Sometimes, direct verification is not possible or even required. If a customer reports that a printer is broken, the verifier may not have to reproduce the problem by attempting to print something. It may be good enough to verify that new print jobs are queuing and not being printed. Such superficial verification is fine in that situation.

At other times, exact duplication *is* required. The verifier might fail to reproduce the problem on his or her own desktop PC and may need to duplicate the problem on the customer's PC. Once the problem is reproduced in the customer's environment, it can be useful to try to duplicate it elsewhere to determine whether the problem is local or global. When supporting a complicated product, you must have a lab of equipment ready to reproduce reported problems.

Verification at E-commerce Sites

E-commerce sites have a particularly difficult time duplicating the customer's environment. Although Java and other systems promise that you can "write once, run anywhere," the reality is that you must be able to duplicate the customer's environment for a variety of web browsers, web browser versions, and even firewalls.

One company needed to test access to its site with and without a firewall. The company's QA effort had a PC that was live on the Internet for such testing. Because the PC was unprotected, it was isolated physically from other machines, and the OS was reloaded regularly.

28.4 Phase C: Planning and Execution

In Phase C, the problem is fixed. Doing so involves planning possible solutions, selecting one, and executing it ([Figure 28.4](#)).

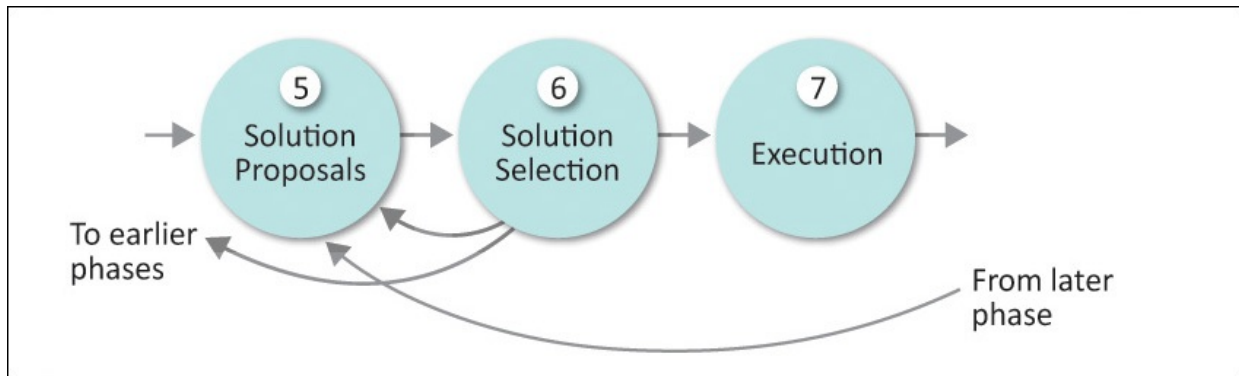


Figure 28.4: Phase C: Fix it.

28.4.1 Step 5: Solution Proposals

Step 5 is the point at which the subject-matter expert (SME) enumerates possible solutions. Depending on the problem, this list may be long or short. For some problems, the solution may be obvious, with only one proposed solution. At other times, many solutions are possible. Often, verifying the problem in the previous step helps to identify possible solutions.

The “best” solution varies, depending on the context. At a financial institution, the helpdesk’s solution to a client-side NFS problem might be to reboot. That step could be faster than trying to fix the problem, and it gets the customer up and running quickly. In contrast, in a research environment, it would make sense to try to find the source of the problem, perhaps by unmounting and remounting the NFS mount that reported the problem.

Case Study: Radical Print Solutions

In our earlier printing example, because the customer indicated that he needed to leave for the airport soon, it might have been appropriate to suggest alternative solutions, such as recommending a different printer known to be working. If the customer is an executive flying from New Jersey to Japan with a stopover in San Jose, it might be reasonable to transfer the file to an office in San Jose, where it could be printed while the customer is in flight. A clerk could hand the printout to the executive while he waits for his connecting flight at the San Jose airport. Tom witnessed such a solution being used. The printer, in this case, was a very expensive plotter. Only one such plotter was available at each company

location.

Some solutions are more expensive than others. Any solution that requires a desk-side visit is generally going to be more expensive than one that can be handled without such a visit. Understanding this cost difference can be useful in making purchasing decisions. Lack of remote-support capability affects the total cost of ownership of a product. Both commercial and noncommercial tools are available that add remote support to many products.

An SA who does not know any possible solutions should escalate the issue to other, more experienced SAs.

28.4.2 Step 6: Solution Selection

Once the possible solutions have been enumerated, one of them is selected to be attempted first—or next, if we are looping through these steps. This role is also performed by the SME.

Selecting the best solution tends to be either extremely easy or extremely difficult. However, solutions often cannot and should not be applied simultaneously, so possible solutions must be prioritized.

The customer should be included in this prioritization. Customers have a better understanding of their own time pressures. A customer who is a commodities trader will be much more sensitive to downtime during the trading day than, say, a technical writer or even a developer, provided that he or she is not on deadline. If solution A fixes the problem forever but requires downtime and solution B is a short-term fix, the customer should be consulted as to whether A or B is “right” for the situation. The SME is responsible for explaining the possibilities, but the SA should appreciate the priorities, based on the environment. There may be predetermined service goals for downtime during the day. SAs on Wall Street know that downtime during the day can cost millions of dollars, so short-term fixes may be selected and a long-term solution scheduled for the next maintenance window. In a research environment, the rules about downtime are more relaxed, and the long-term solution may be selected immediately. Some sites centralize their helpdesks to a bizarre extreme that results in SAs no longer knowing into which category their customers fall. This is rarely a good thing.

When dealing with more experienced customers, it can be a good idea to let them participate in this phase. They may have useful feedback. In the case of inexperienced customers, it can be intimidating or confusing to hear all these

details. It may even unnecessarily scare them. For example, listing every possibility from a simple configuration error to a dead hard disk may cause the customer to panic and is a generally bad idea, especially when the problem turns out to be a simple typo in `CONFIG.SYS`.

Even though customers may be inexperienced, they should be encouraged to participate in determining and choosing the solution. This can help educate them so that future problem reports can flow more smoothly and even enable them to solve their own problems. It can also give customers a sense of ownership—the warm fuzzy feeling of being part of the team/company, not simply “users.” This approach can help break down the us-versus-them mentality common in industry today.

28.4.3 Step 7: Execution

The solution is attempted in step 7. The skill, accuracy, and speed at which this step is completed depend on the skill and experience of the person executing the solution.

The term **craft worker** refers to the SA, operator, or laborer who performs the technical tasks involved. This term comes from other industries, such as telecommunications, in which one person may receive the order and plan the provisioning of the service, but the craft workers run the cables, connect circuits, and so on, to provide the service. In a computer network environment, the network architect might be responsible for planning the products and procedures used to give service to customers, but when a new Ethernet interface needs to be added to a router, the craft worker installs the card and configures it.

Sometimes, the customer becomes the craft worker. This scenario is particularly common when the customer is remote and using a system with little or no remote control. In that case, the success or failure of this step is shared with the customer. A dialogue is required between the SA and the customer to make the solution work. Has the customer executed the solution properly? If not, is the customer causing more harm than good?

Adjust the dialogue based on the skill of the customer. It can be insulting to spell out each command, space, and special character to an expert customer. It can be intimidating to a novice customer if the SA rattles off a complex sequence of commands. Asking, “What did it say when you typed that?” is better than “Did it work?” in these situations. Be careful, however, not to assume too much; some customers are good at sounding as though they are more experienced than they are.

This kind of communication is not an innate skill, but rather must be learned. Training is available to hone SAs' skills in this area. Workshops that focus on this area often have titles that include buzzwords such as active listening, interpersonal communication, interpersonal effectiveness, or simply advanced communication.

At this point, it is tempting to think that we have finished. However, we haven't finished until the work has been checked and the customer is satisfied. That brings us to the final phase.

28.5 Phase D: Verification

At this point, the problem *should* have been remedied, but we need to verify that. Phase D isn't over until the customer agrees that the problem has been fixed ([Figure 28.5](#)).

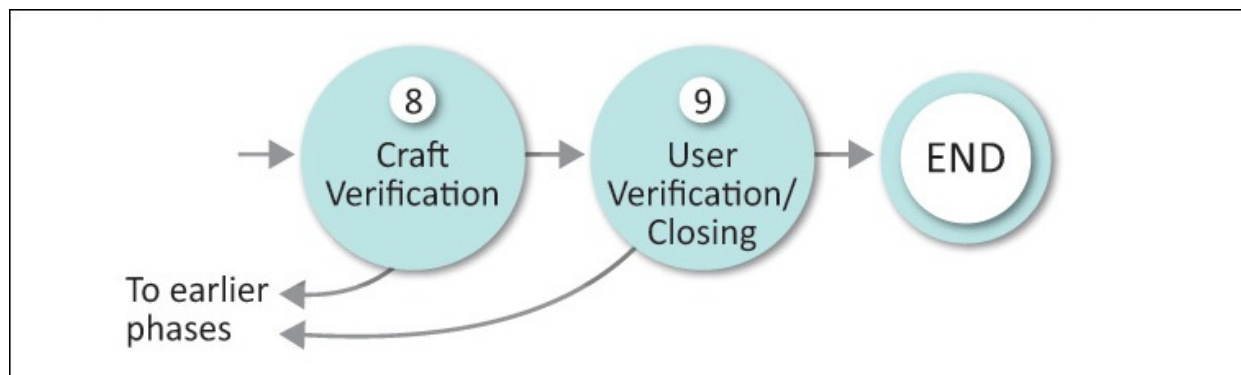


Figure 28.5: Phase D: Verify it.

28.5.1 Step 8: Craft Verification

In step 8, the craft worker who executed step 7 verifies that the actions taken to fix the problem were successful. If the process used to reproduce the problem in step 4 is not recorded properly or not repeated exactly, the verification will not happen correctly. If the problem still exists, return to step 5 or, possibly, an earlier step.

The Unix `diff` Program

The Unix command `diff` can be useful in this situation; this program displays the difference between two text files. Capture the output generated when the problem is reproduced. As attempts are made to fix the problem, run the program again, capturing the output to a new file. Run `diff` against the two captures to see whether there is any difference. Alternatively, you might copy the output that demonstrates the problem to

a new file and edit it to be the way it should on a working system. (You might have a working system to generate sample “good” output.) The `diff` program can then be used to compare the current output with the corrected output. You’ll know you’ve made the right changes when `diff` claims that the files are the same. Some systems do not generate output that is well suited to `diff`, but Perl and other tools can pare down the output to make it more palatable to `diff`.

28.5.2 Step 9: Customer Verification/Closing

The final step is for the customer to verify that the issue has been resolved. If the issue hasn’t been resolved, the job isn’t done. This role is performed by the customer.

Presumably, if the craft worker verified that the solution worked (step 8), this step should not be needed. However, customers often report at this point that the problem still exists. This is such a critical problem that it must be a separate step.

Customer verification reveals mistakes made in previous phases. Perhaps the customer did not properly express the problem, the SA did not understand the customer, or the SA did not properly record the problem—all communication problems. Errors may have crept into the planning phase. The problem that was verified in step 4 may have been a different problem that also exists, or the method that verified the problem may have been incomplete. The solution may not have fixed the entire problem or may have simply made the problem intermittent.

In either case, if the customer does not feel that the problem has been fixed, there are many possible actions. Obviously, step 4 should be repeated to find a more accurate method to reproduce the problem. However, at this point, it may be appropriate to return to other steps. For example, the problem could be reclassified (step 2), or restated (step 3), or escalated to more experienced SAs (step 5). If all else fails, you may have to escalate the problem to management.

It is important to note that the verification step isn’t used to verify that the customer is happy, but rather that the customer’s request has been satisfied. Customer satisfaction is a metric to be measured elsewhere.

Once customer verification is complete, the issue is closed.

28.6 Perils of Skipping a Step

Each step is important. If any step in this process is performed badly, the process

can break down. Many SAs skip a step, either because of lack of training or because of an honest mistake. Many stereotypes about bad SAs are the result of SAs skipping a particular step. We've assigned Seinfeldesque names to each of these stereotypes and list possible ways of improving the SAs process:

- **The Ogre:** Grumpy, caustic SAs are trying to scare customers away from step 1 and are preventing the greeting from happening. *Suggestion:* Management must set expectations for friendliness. The scope of responsibility must be a written policy communicated to both SAs and customers.
- **The Midelegator:** If you've called a large company's technical support line and the person who answered the phone refused to direct your call to the proper department (step 2), you know what it's like to deal with a midelegator. *Suggestion:* Design a formal decision tree indicating which issues are delegated where.
- **The Assumer:** SAs who skip step 3 simply assume that they understand what the problem is when they really don't. *Suggestion:* Coach the person on active listening; if that fails, send the person to a class on the topic.
- **The Non-verifier:** An SA who skips problem verification (step 4) is usually busy fixing the wrong problem. One day, Tom was panicked by the news that "the network is down." In reality, a nontechnical customer couldn't read his email and reported the potential problem. This claim hadn't been verified by the newly hired SA, who hadn't yet learned that certain novice customers report all problems that way. The customer's email client was misconfigured. *Suggestion:* Teach SAs to replicate problems, especially before escalating them. Remind them that it isn't nice to panic Tom.
- **The Wrong Fixer:** Inexperienced SAs sometimes are not creative or are too creative in proposing and selecting solutions (steps 5 and 6). But skipping these steps entirely results in a different issue. After being taught how to use an Ethernet monitor (a network sniffer), an inexperienced but enthusiastic SA was found dragging out the sniffer no matter which problem was being reported. He was a Wrong Fixer. *Suggestion:* Provide mentoring or training. Increase the breadth of solutions with which the SA is familiar.
- **The Executioner:** Incompetent SAs sometimes cause more harm than good when they execute solutions incorrectly. It is quite embarrassing to apply a fix to the wrong machine; however, it happens. *Suggestion:* Train

the SA to check what has been typed before pressing ENTER or clicking OK. It can be vital to include the hostname in your shell prompt.

- **The Hit-and-Run SA:** This SA walks into a customer's office, types a couple of keystrokes, and waves goodbye while walking out the door and saying, "That should fix it." The customers are frustrated to discover that the problem was not fixed. In all fairness, what was typed really should have fixed the problem, but it didn't. *Suggestion:* Management needs to set expectations on verification.
- **The Closer:** Some SAs are obsessed with "closing the ticket." Often, however, SAs are judged on how quickly they close tickets. In that case, the SAs are pressured to skip the final step. We borrow this name from the term used to describe high-pressure salespeople focused on "closing the deal." *Suggestion:* Management should not measure performance based on how quickly issues are resolved but on a mixture of metrics that drive the preferred behavior. Metrics should not include time waiting for customers when calculating how long it took to complete the request. Tracking systems should permit a request to be put into a customer wait state while customers complete actions, and that time should be subtracted from the time-to-completion metrics.

Team of One

The solo SA can still benefit from using this model to make sure that customers have a well-defined way to report problems; that problems are recorded and verified; that solutions are proposed, selected, and executed; and that both the SA and the customer have verified that the problem has been resolved. When an organization has a single SA, problems with specific applications can be escalated to that vendor's support lines.

28.7 Optimizing Customer Care

Once the basic process is understood, there are ways to improve it. On the micro level, you can look into improving each step; on the macro level, you can look at how the steps fit together.

28.7.1 Model-Based Training

Internal training should be based on this model so that the SA staff members consistently use it. After the initial training, more experienced staff should mentor newer SAs to help them retain what they have learned. Certain steps can

be helped by specific kinds of training.

Improvements can be made by focusing on each step. In fact, entire books could be written on each step. This has happened in other professions that have similar models, such as nursing, sales, and so on.

A lack of training hurts the process. For example, an ill-defined delineation of responsibilities makes it difficult for a classifier to delegate the issue to the right person. Inexperienced recorders may not gather the right information in step 3, which makes further steps difficult and may require contacting the customer unnecessarily. A written chart of who is responsible for what, as well as a list of standard information to be collected for each classification, will reduce these problems.

28.7.2 Holistic Improvement

In addition to focusing on improving each step, you may focus on improving the entire process. Transitioning to each new step should be fluid. If the customer sees an abrupt, staccato handoff between steps, the process can appear amateurish or disjointed.

Every handoff is an opportunity for mistakes and miscommunication. The fewer handoffs that take place, the fewer opportunities there are for mistakes.

A site small enough to have a single SA has zero opportunities for this class of error. However, as systems and networks grow and become more complicated, it becomes impossible for a single person to understand, maintain, and run the entire network. As a system grows, handoffs become a necessary evil. This explains a common perception that larger SA groups are not as effective as smaller ones. Therefore, when growing an SA group, you should focus on maintaining high-quality handoffs. Alternatively, you might choose to develop a single point of contact or customer advocate for an issue. That results in the customers seeing a single face for the duration of a problem.

28.7.3 Increased Customer Familiarity

If a customer talks to the same person whenever he or she calls for support, the SA will likely become familiar with the customer's particular needs and be able to provide better service. There are ways to improve the chance of this happening. For example, SA staff subteams may be assigned to particular groups of customers rather than to the technology they support. Or, if the phone-answering staff is extremely large, the group might use a telephone call center system, whereby customers call a single number and the call center routes the

call to an available operator. Modern call center systems can route calls based on caller ID, using this functionality, for example, to route the call to the same operator the caller spoke to last time, if that person is available. With such a system, customers are likely to reach the same person each time. It can be very comforting to speak to someone who recognizes your voice.

28.7.4 Special Announcements for Major Outages

During a major network outage, many customers may be trying to report problems. If customers report problems through an automatic phone response system (“Press 1 for ..., press 2 for ...”), such a system can usually be programmed to announce the network outage before listing the options. “Please note the network connection to Denver is currently experiencing trouble. Our service provider expects it to be fixed by 3 PM. Press 1 for ..., press 2 for....”

Hearing the announcement reassures the customer that SAs are aware of the problem, and that someone is working to resolve it. This cuts down on duplicate problem reports that need to be resolved individually.

28.7.5 Trend Analysis

Spend some time each month looking for trends, and take action based on them. Here are some trends to look for:

- **Does a customer report the same issue over and over?** Why is it recurring? Does the customer need training, or is that system really so badly broken?
- **Are there many issues in a particular category?** Is that system difficult to use? Could it be redesigned or replaced, or could the documentation be improved?
- **Are many customers reporting the same issue?** Can they all be notified at once? Should such problems receive higher priority?
- **Can some categories of requests become self-service?** Often, a customer request states that an SA is needed because something requires privileged access, such as superuser or administrator access. Look for ways to empower customers to help themselves. Many of these requests can become self-service with a little bit of web programming.
- **Who are your most frequent customers?** Calculate which department generates the most tickets or who has the highest average tickets per member. Calculate which customers make up your top 20 percent of requests. Do these ratios match your funding model, or are certain

customer groups more “expensive” than others?

- **Is a particular time-consuming request one of your frequent requests?**

If customers often accidentally delete files and you waste a lot of time each week restoring files from tape, you can invest time in helping the users learn about `rm -i` or how to use other safe-delete programs.

Alternatively, maybe it would be appropriate to advocate for the purchase of a system that supports snapshots or lets users do their own restores. If you can generate a report of the number and frequency of restore requests, management can make a more informed decision or decide to talk to certain users about being more careful.

This chapter does not discuss metrics, but a system of metrics grounded in this model might be the best way to detect areas needing improvement. The nine-step process can be instrumented easily to collect metrics. Developing metrics that drive the right behaviors is difficult. For example, if SAs are rated based on how quickly they close tickets, one might accidentally encourage the Closer behavior described earlier. As SAs proactively prevent problems, reported problems will become more serious and time-consuming. If average time to completion grows, does that mean that minor problems were eliminated or that SAs are slower at fixing all problems? SA-generated tickets for proactive fixes and planned projects make the SA contributions clearer.

Case Study: Who Generates the Most Requests?

Trend analysis does not need to be overly complex.

At one site we simply looked at which customers opened the most tickets in the last year. We found that 3 of the 600 people opened 10 percent of all tickets. That’s a lot!

Looking into the tickets opened by these three people revealed some obvious improvements we could make.

One person opened so many tickets because he was pestering the SAs for workarounds to the bugs in the old version of the L^AT_EX typesetting package that he was using. He refused to upgrade to the latest version, which fixed most of the problems he was reporting. We worked with his manager, who agreed to take responsibility for convincing the person to upgrade.

The next person’s tickets were mostly asking for help with basic computer tasks. We worked with his manager to arrange for training to make him more self-sufficient.

The last person was basically asking SAs to do his job for him. There was an obvious disagreement over which tasks were within the scope of the SA responsibilities and which were not. We worked with this person's manager to better define the scope and support us when we started enforcing better boundaries.

28.7.6 Customers Who Know the Process

A better-educated customer is a better customer. Customers who understand the nine steps that will be followed are better prepared when reporting the problem. These customers can provide more complete information when they call, because they understand the importance of complete information in solving the problem. In gathering this information, they will have narrowed the focus of the problem report. They might have specific suggestions on how to reproduce the problem. They may have narrowed the problem down to a specific machine or situation. Their additional preparation may even lead them to solve the problem on their own! Training for customers should include explaining the nine-step process to facilitate interaction between customers and SAs.

Preparing Customers at the Department of Motor Vehicles

Tom noticed that the New Jersey Department of Motor Vehicles had recently changed its on-hold message to include which four documents should be on hand if the person was calling to renew a vehicle registration. Now, rather than waiting to speak to a person only to find out that you don't have, say, your insurance ID number, there is a better chance that once connected, you will have everything needed to complete the transaction.

28.7.7 An Architecture That Reflects the Process

Architectural decisions may impede or aid the classification process. The more complicated a system is, the more difficult it can be to identify and duplicate the problem. Sadly, some well-accepted software design concepts, such as delineating a system into layers, are at odds with the nine-step process. For example, a printing problem in a large Unix network could be a problem with DNS, the server software, the client software, a misconfigured user environment, the network, DHCP, the printer's configuration, or even the printing hardware itself. Typically, many of those layers are maintained by separate groups of people. To diagnose the problem accurately requires either the SAs to be experts

in all those technologies or the layers to do cross-checking.

You should keep in mind how a product will be supported when you are designing a system. The electronics industry has the concept of design for manufacture; we should think in terms of design for support.

28.8 Summary

This chapter is about communication. The process recommended in this chapter helps us think about how we communicate with customers, and it gives us a base of terminology to use when discussing our work. All professionals have a base of terminology to use to effectively communicate with one another.

This chapter presents a formal, structured model for handling requests from customers. The process has four phases: greeting, problem identification, planning and execution, and fixing and verifying. Each phase has distinct steps, summarized in [Table 28.1](#).

Phase	Step	Role
Phase A: "Hello!"	1. The Greeting	Greeter
Phase B: "What's wrong?"	2. Problem Classification	Classifier
	3. Problem Statement	Recorder
	4. Problem Verification	Reproducer
Phase C: "Fix it."	5. Solution Proposals	Subject-matter expert
	6. Solution Selection	Subject-matter expert
	7. Execution	Craft worker
Phase D: "Verify it."	8. Craft Verification	Craft worker
	9. Customer Verification/Closer	Customer

Table 28.1: Overview of Phases for Problem Solution

Following this model makes the incident-reporting process more structured and formalized. Once it is in place, it exposes areas for improvement within your organization. You can integrate the model into training plans for SAs, as well as educate customers about the model so they can be better advocates for themselves. The model can be applied for the gathering of metrics. It enables trend analysis, even if only in simple, ad hoc ways, which is better than nothing.

It is critical that you use helpdesk issue-tracking software rather than trying to remember the requests in your head, using scraps of paper, or relying on email boxes. Automation reduces the tedium of managing incoming requests and

collecting statistics. Software that tracks tickets for you saves time in real ways. Tom once measured that a group of three SAs was spending an hour a day per person to track issues. That is a loss of two staff days per week!

The process described in this chapter brings clarity to the issue of customer support by defining which steps must be followed for a single successful call for help. We showed why these steps are to be followed and how each step prepares you for future steps.

Although knowledge of this model can improve an SA's effectiveness by leveling the playing field, it is not a panacea—nor is it a replacement for creativity, experience, or having the right resources. The model does not replace the right training, the right tools, and the right support from management, but it must be part of a well-constructed helpdesk.

Many SAs are naturally good at customer care and react negatively to structured techniques like this one. It's great that they have found their own structure and use it with consistently great results. It likely has many of the rudiments discussed here. Do what works for you. However, to grow the number of SAs in the field, more direct instruction will be required. For the millions of SAs who have not found the perfect structure for themselves, consider this structure a good starting point.

Exercises

1. Are there times when you should not use the nine-step model?
2. What are the tools used in your environment for processing customer requests, and how do they fit into the nine-step model? Are there ways they could fit better?
3. What are all the ways to greet customers in your environment? Which ways could you use but don't? Why?
4. In your environment, you greet customers by various methods. How do the methods compare in terms of cost, speed (faster completion), and customers' preference? Is the most expensive method the one that customers prefer the most?
5. Some problem statements can be stated concisely, such as the routing problem example in step 3. Dig into your trouble-tracking system to find five typically reported problems. What is the shortest problem statement that completely describes the issue?
6. Query your ticket-tracking software and determine who were your top 10 ticket creators overall in the last 12 months; then sort them by customer

group or department. Next, determine which customer groups have the highest per-customer ticket count. Which customers make up your top 20 percent? Now that you have this knowledge, what will you do? Examine other queries from [Section 28.7.5](#).

7. Which is the most important of the nine steps? Justify your answer.

Chapter 29. Debugging

In this chapter, we dig deeply into what is involved in debugging problems. In [Chapter 28](#), “[Handling an Incident Report](#),” we put debugging into the larger context of customer care. This chapter, in contrast, is about *you* and what you do when faced with a single technical problem.

Debugging is not simply making a change that fixes a problem. That’s the easy part. Debugging begins with you understanding the problem, finding its cause, and then making the change that makes the problem disappear for good. Temporary or superficial fixes, such as rebooting, that do not fix the cause of the problem merely guarantee more work for *you* in the future. We continue that theme in [Chapter 30](#), “[Fixing Things Once](#).”

Since anyone reading this book has a certain amount of smarts and a level of experience (while we’re at it, you’re good looking and a sharp dresser), we do not need to be pedantic about this topic. You’ve debugged problems; you know what it’s like. This chapter is intended to make you conscious of the finer points of the process and then discuss some ways of making it even smoother. We encourage you to be systematic. It’s better than randomly poking about.

29.1 Understanding the Customer’s Problem

The first step in fixing a problem is to understand, at a high level, what the customer is trying to do and which part of that effort is failing. In other words, the customer is doing something and expecting a particular result, but something else is happening instead.

For example, customers may be trying to read their email and aren’t able to. They may report this in many ways: “My mail program is broken” or “I can’t reach the mail server” or “My mailbox disappeared!” Any of those statements may be true, but the problem also could be a network problem, a power failure in the server room, or a DNS problem. These issues may be beyond the scope of what the customer understands or should need to understand. Therefore, it is important for you to gain a high-level understanding of what the customer is trying to do.

Sometimes, customers aren’t good at expressing themselves, so care and understanding must prevail. “Can you help me understand what the document should look like?”

It’s common for customers to use jargon but in an incorrect way. They believe

that's what the SA wants to hear. They're trying to be helpful. It is valid for the SA to respond along the lines of, "Let's back up a little. What exactly is it you're trying to do? Just describe it without being technical."

The complaint doesn't usually convey the actual problem. A customer might complain that the printer is broken. This sounds like a request to have the printer repaired. However, an SA who takes time to understand the entire situation might learn that the customer needs to have a document printed before a shipping deadline. In that case, it becomes clear that the customer's complaint isn't about the hardware but about needing to print a document. Printing to a different printer becomes a better solution.

Some customers provide a valuable service by digging into the problem before reporting it. A senior SA partners with these customers but understands that there are limits. It can be nice to get a report such as "I can't print; I think there is a DNS problem." However, we don't take such reports at face value. You understand the system architecture better than the customers do, so you still need to verify the DNS portion of the report, as well as the printing problem. Maybe it is best to interpret the report as two possibly related reports: Printing isn't working for a particular customer, and a certain DNS test failed. For example, a printer's name may not be in DNS, depending on the print system architecture. Often, customers will ping a host's name to demonstrate a routing problem but overlook the error message and the fact that the DNS lookup of the host's name failed. Or they may try to ping a host to show that it is down but not realize that there is a firewall that blocks the ping, but allows some other protocols through.

Finding the Real Problem

One of Tom's customers was reporting that he couldn't ping a particular server located about 1,000 miles away in a different division. He provided traceroutes, ping information, and a lot of detailed evidence. Rather than investigating potential DNS, routing, and networking issues, Tom stopped to ask, "Why do you require the ability to ping that host?" It turned out that pinging that host wasn't part of the person's job; instead, the customer was trying to use that host as an authentication server. The problem to be debugged should have been "Why can't I authenticate off this host?" or even better, "Why can't I use service A, which relies on the authentication server on host B?"

By contacting the owner of the server, Tom found the very simple answer: Host B had been decommissioned, and properly configured clients should have automatically started authenticating off a new server.

This wasn't a networking issue at all, but a matter of client configuration. The customer had hard-coded the IP address into his configuration but shouldn't have. A lot of time would have been wasted if the problem had been pursued as originally reported.

In short, a reported problem is about an expected outcome not happening. Now let's look at the cause.

29.2 Fixing the Cause, Not the Symptom

To build sustainable reliability, you must find and fix the cause of the problem, not simply work around the problem or find a way to recover from it quickly. Although workarounds and quick recovery times are good things, and sometimes necessary, fixing the root cause of a problem is better.

Often, we find ourselves in a situation like this: A co-worker reports that there was a problem and that he fixed it. "What was the problem?" we inquire.

"The host needed to be rebooted."

"What was the problem?"

"I told you! The host needed to be rebooted."

A day later, the host needs to be rebooted again.

A host needing to be rebooted isn't a problem, but rather a solution. The problem might have been that the system froze, buggy device drivers were malfunctioning, a kernel process wasn't freeing memory and the only choice was to reboot, and so on. If the SA had determined what the true problem was, it could have been fixed for good and would not have returned.

The same goes for "I had to exit and restart an application or service" and other mysteries. Many times, we've seen someone fix a "full-disk" situation by deleting old log files. However, the problem returns as the log files grow again. Deleting the log files fixes the symptoms, but activating a script that would rotate and automatically delete the logs would fix the problem.

Even large companies get pulled into fixing symptoms instead of root causes. For example, Microsoft got a lot of bad press when it reported that a major feature of Windows 2000 was that it would reboot faster. In fact, users would have preferred that it didn't need to be rebooted so often in the first place.

29.3 Being Systematic

It is important to be methodical, or systematic, about finding the cause and

fixing it. To be systematic, you must form hypotheses, test them, note the results, and make changes based on those results. Anything else is simply making random changes until the problem goes away.

The process of elimination and successive refinement are commonly used in debugging. The **process of elimination** entails removing different parts of the system until the problem disappears. The problem must have existed in the last portion removed. **Successive refinement** involves adding components to the system and verifying at each step that the desired change happens.

The process of elimination is often used when debugging a hardware problem, such as replacing memory chips until a memory error is eliminated or pulling out cards until a machine is able to boot. Elimination is used with software applications, such as eliminating potentially conflicting drivers or applications until a failure disappears. Some OSs have tools that search for possible conflicts and provide test modes to help narrow the search.

Successive refinement is an additive process. For example, to diagnose an IP routing problem, the `tracert` command reports connectivity one network hop away and then reports connectivity two hops away, then three, then four, and so on. When the probes no longer return a result, we know that the router at that hop wasn't able to return packets. The problem is at that last router. When connectivity exists but there is packet loss, a similar methodology can be used. You can send many packets to the next router and verify that there is no packet loss. You can successively refine the test by including more distant routers until the packet loss is detected. You can then assert that the loss is on the most recently added segment.

Sometimes, successive refinement can be thought of as **follow-the-path** debugging. To use this method, you must follow the path of the data or the problem, reviewing the output of each process to make sure that it is the proper input to the next stage. It is common on Unix systems to have an assembly-line approach to processing. One task generates data, the others modify or process the data in sequence, and the final task stores it. Some of these processes may happen on different machines, but the data can be checked at each step. For example, if each step generates a log file, you can monitor the logs at each step. When debugging an email problem that involves a message going from one server to a gateway and then to the destination server, you can watch the logs on all three machines to see that the proper processing has happened at each place. When tracing a network problem, you can use tools that let you snoop packets as they pass through a link to monitor each step. When dealing with Unix software that uses the shell `|` (pipe) facility to send data through a series of programs, the

`tee` command can save a copy at each step.

Shortcuts and optimizations can be used with these techniques. Based on past experience, you might skip a step or two. However, this is often a mistake, because it might result in you jumping to a conclusion.

If you *are* going to jump to a conclusion, the problem is often related to the most recent change made to the host, network, or whatever is having a problem. This sort of problem usually indicates a lack of testing. Therefore, before you begin debugging a problem, ponder for a moment which changes were made recently: Was a new device plugged into the network? What was the last configuration change to a host? Was anything changed on a router or a firewall? Often, the answers can direct your search for the cause.

Six Classes of Network Bugs

Six classes of bugs can limit network performance:

- Packet losses, corruption, congestion, bad hardware
- IP routing, long round-trip times
- Packet reordering
- Inappropriate buffer space
- Inappropriate packet sizes
- Inefficient applications

Any one of these problems can hide all other problems. This is why solving performance problems requires a high level of expertise. Because debugging tools are rarely very good, it is “akin to finding the weakest link of an invisible chain” ([Mathis 2003](#)). Therefore, if you are debugging any of these problems and are not getting anywhere, pause a moment and consider that it might be one of the other problems.

29.4 Having the Right Tools

Debugging requires the right diagnostic tools. Some tools are physical devices; others, software tools that are either purchased or downloaded; and still others, home-grown. Of course, knowledge is always the most important tool.

Diagnostic tools let you peer into a device or system to see its inner workings. However, if you don’t know how to interpret what you see, all the data in the world won’t help you solve the problem.

29.4.1 Training Is the Most Important Tool

Training usually involves learning how the system works, how to view its inner workings, and how to interpret what you see. For example, when training someone how to use an Ethernet monitor (sniffer), teaching the person how to capture packets is easy. Most of the training time is spent explaining how various protocols work so that you understand what you see. Learning the tool is easy. Getting a deeper understanding of what the tool lets you see takes considerably longer.

Although manuals are great, formal training sets you apart from others. Formal training has a number of benefits:

- Off-site training takes you away from the interruptions of your job and lets you focus on learning new skills.
- Formal training usually covers all the features, not only the ones with which you've had time to experiment.
- Instructors often reveal bugs or features that the vendor may not want revealed in print.
- Access to a lab of machines allows you to try things that, because of production requirements, you couldn't otherwise try.
- You can list the training on your resume; this can be more impressive to prospective employers than actual experience, especially if you receive certification.

29.4.2 Understanding the Underlying Technology

Unix systems have a reputation for being very easy to debug, most likely because so many experienced Unix SAs have in-depth knowledge of the inner workings of the system. Such knowledge is easily gained. Unix systems come with documentation about their internals; early users had access to the source code itself. Much of the system is driven by scripts that can be read easily to gain an understanding of what is going on behind the scenes. Many books dissect the source code of Unix kernels for educational purposes. These include *Lions' Commentary on UNIX 6th Edition, with Source Code* by Lions ([1996](#)), and *The Design and Implementation of the FreeBSD Operating System, 2nd ed.*, by McKusick, M. K., Neville-Neil, G. V. & Watson, R. N. M. ([2014](#)).

Microsoft Windows developed an early reputation for being a difficult system to debug when problems arose. Rhetoric from the Unix community claimed that it was a black box with no way to get the information you needed to debug

problems. In reality, there were mechanisms, but the only way to learn about them was through vendor-supplied training. From the perspective of a culture that is very open about information, this was difficult to adjust to. It took many years to disseminate the information about how to access Windows internals and how to interpret what was found. Book series such as *Windows® Internals, Part 1, 6th ed.*, ([Russinovich, Solomon, D. A. & Ionescu 2012](#)) enable you to gain a deeper understanding of what's going on under the Windows hood.

Understanding Why a Tool Draws a Conclusion

It is important to understand not only the system being debugged, but also the tools being used to debug it. Once, Tom was helping a network technician with a problem: A PC couldn't talk to any servers, even though the link light was illuminated. The technician disconnected the PC from its network jack and plugged in a new handheld device that could test and diagnose a large list of LAN problems. However, the output of the device was a list of conclusions without information about how it was arriving at them. The technician was basing his decisions on the output of this device without question.

Tom kept asking, "The device claims it is on network B, but how did it determine that?" The technician didn't know or care. Tom stated, "I don't think it is really on network B! Network B and C are bridged right now, so if the network jack were working, it should claim to be on networks B and C at the same time." The technician disagreed, because the very expensive tool couldn't possibly be wrong, and the problem must be with the PC.

It turned out that the tool was guessing the IP network after finding a single host on the LAN segment. This jack was connected to a hub, which had another workstation connected to it in a different office. The uplink from the hub had become disconnected from the rest of the network. Without knowing how the tool performed its tests, there was no way to determine why a tool would report such a claim, and further debugging would have been a wild goose chase. Luckily, there was a hint that something was suspicious—it didn't mention network C. The process of questioning the conclusion drew Tom and the technician to the problem's real cause.

29.4.3 Choosing the Right Tools

What makes a good tool? Small, combinable tools are usually better than large,

complicated tools. The best tool is one that provides the simplest solution to the problem at hand, and can be easily combined with other tools. The more sophisticated a tool is, the more likely it will get in its own way or simply be too big to carry to the problem.

NFS mounting problems can be debugged with three simple tools: `ping`, `traceroute`, and `rpcinfo`. Each does one thing, and does that one thing well. If the client can't mount from a particular server, for example, make sure that they can ping each other. If they can't, it's a network problem, and `traceroute` can isolate the problem. If `ping` succeeded, connectivity is working correctly, and there must be a protocol problem. From the client, the elements of the NFS protocol can be tested with `rpcinfo`. For example, try `rpcinfo -T udp servername portmap` in Solaris or `rpcinfo -u servername portmap` in Linux. You can test the `portmap`, `traceroute`, `mountd`, `nfs`, `nlockmgr`, and `status`. If any of these tests fail, you can deduce that the appropriate service isn't working. If all of them succeed, you can deduce that it is an export permission problem, which usually means that the name of the host listed in the export list is not exactly what the server sees when it performs a reverse DNS lookup. These extremely powerful diagnostics are carried out with extremely simple tools. You can use `rpcinfo` for all Sun RPC-based protocols ([Stern, Eisler & Labiaga 2001](#)).

Protocols based on TCP often can be debugged with a different triad of tools: `ping`, `traceroute`/`tracert`, and `telnet`. These tools are available on every platform that supports TCP/IP (Windows, Unix, and others). Again, `ping` and `traceroute` can diagnose connectivity problems. Then `telnet` can be used to manually simulate many TCP-based protocols. For example, email administrators know enough of SMTP to TELNET to port 25 of a host and type the SMTP commands as if they were the client; you can diagnose many problems by watching the results. Similar techniques work for FTP, HTTP, and other TCP-based protocols. *TCP/IP Illustrated, Volume 1: The Protocols, 2nd ed.*, by Fall & Stevens ([2012](#)) provides an excellent review of how the protocols work.

There is always room for new tools that improve on the old ones. Keeping up-to-date on the latest tools can be difficult, preventing you from being an early adopter of new technology. Several forums, such as USENIX LISA conferences, as well as web sites and mailing lists, can alert you to these new tools as they are announced. Some SaaS vendors have very interesting offerings in this space.

Sometimes, the best tools are simple home-grown tools or a combination of

other small tools and applications, as the following anecdote shows.

Finding the Latency Problem

Once, Tom was tracking reports of high latency on a network link. The problem happened only occasionally. He set up a continuous (once per second) ping between two machines that should have demonstrated the problem and recorded this output for several hours. He observed consistently good (low) latency, except that occasionally there seemed to be trouble.

A small `perl` program was written to analyze the logs, extract pings with high latency—latency more than three times the average of the first twenty pings—and highlight missed pings. Tom noticed that no pings were being missed but that every so often a series of pings took much longer to arrive. He used a spreadsheet to graph the latency over time. Visualizing the results helped him notice that the problem occurred every five minutes, within a second or two. It also happened at other times, but every five minutes he was assured of seeing the problem. He realized that some protocols do certain operations every five minutes. Could a route table refresh be overloading the CPU of a router? Was a protocol overloading a link?

By process of elimination, he isolated the problem to a particular router. Its CPU was being overloaded by routing table calculations, which happened every time there was a real change to the network plus every five minutes during the usual route table refresh. This agreed with the previously collected data. The fact that it was an overloaded CPU and not an overloaded network link explained why latency increased but no packets were lost. The router had enough buffering to ensure that no packets were dropped. Once Tom fixed the problem with the router, the ping test and log analysis were used again to demonstrate that the problem had been fixed.

The customer who had reported the problem was a scientist with a particularly condescending attitude toward SAs. After confirming with him that the problem had been resolved, the scientist was shown the methodology, including the graphs of timing data. His attitude improved significantly once he found respect for their methods.

29.4.4 Evaluating Tools

When evaluating new tools, assess them based on which problems they can solve. Try to ignore the aspects that are flashy, buzzword-compliant, and full of hype. **Buzzword-compliant** is a humorous term meaning that the product applies to all the current industry buzzwords one might see in the headlines of trade magazines, whether or not such compliance has any benefit.

Ask, “Which real-life problem will this solve?” It is easy for salespeople to focus on flashy, colorful output, but does the flash add anything to the utility of the product? Is the color used intelligently to direct the eye at important details, or does it simply make the product pretty? Are any of the buzzwords relevant? Sure, the device supports SNMP, but will you integrate it into your SNMP monitoring system? Or is SNMP simply used for configuring the device?

Ask for an evaluation copy of the tool, and make sure that you have time to use the tool during the evaluation. Don’t be afraid to send it back if you didn’t find it useful. Salespeople have thick skins, and the feedback you give helps them make the product better in future releases.

29.5 End-to-End Understanding of the System

All sites should have at least one person who understands, end to end, how the system works. On a small system, that’s easy. As systems grow larger and more complex, however, people specialize and end up knowing only their part of the system. Having someone who knows the entire system is invaluable when there is a major outage. In a big emergency, it can be best to assemble a team of experts, each representing one layer of the stack.

How do you retain employees who have this kind of end-to-end knowledge? One way is to promote them. Synopsys promoted people who had end-to-end knowledge into “architect” positions in each technology area. The architects knew more than simply their technology area in depth, and they were good crossover people. Their official role was to track the industry direction: predict needs and technologies two to five years out and start preparing for them (prototyping, getting involved with vendors as alpha/beta customers, and helping to steer the direction of the vendors’ products); architect new services; watch what was happening in the group; steer people toward smarter, more scalable solutions; and so on. This role ensured that such people were around when end-to-end knowledge was required for debugging major issues.

Mysterious Deletions

One example of when end-to-end knowledge was required was when a

customer revealed that some of his files were disappearing. To be more specific, he had about 100 MB of data in his home directory, and all but 2 MB had disappeared. The environment had a system that let users restore files from backups without SA intervention, so he was able to restore his files. However, a couple of days later, the same thing happened; this time, a different set of files disappeared. Again, the files remaining totaled 2 MB. He then sheepishly revealed that this had been going on for a couple of weeks, but he found it convenient to restore his own files and felt embarrassed to bother the SAs with such an odd problem.

The SA's first theory was that there was a virus, but virus scans revealed nothing. The next theory was that someone was playing pranks on him or that there was a badly written cron job. The customer was given pager numbers to call the moment his files disappeared again. Meanwhile, network sniffers were put in place to monitor who was deleting files on that server. The next day, the customer alerted the SAs that his files were disappearing. "What was the last thing you did?" Well, he had simply logged into a machine in a lab to surf the web. The SAs were baffled. The network-monitoring tools showed that the deletions were not coming from the customer's PC or from a rogue machine or mis-programmed server. The SAs had done their best to debug the problem using their knowledge of their part of the system, yet the problem remained unsolved.

Suddenly, one of the senior SAs with end-to-end knowledge of the system, including both Windows and Unix and all the various protocols involved, realized that web browsers keep a cache that gets pruned to stay lower than a certain limit, often 2 MB. Could the browser on this machine be deleting the files? Investigation revealed that the lab machine was running a web browser configured with an odd location for its cache. The location was fine for some users, but when this user logged in, the location was equivalent to his home directory because of a bug (or feature?) related to how Windows parsed directory paths that involved nonexistent subdirectories. The browser was finding a cache with 100 MB of data and deleting files until the space used was less than 2 MB. That explained why every time the problem appeared, a different set of files remained. After the browser's configuration was fixed, the problem disappeared.

The initial attempts at solving the problem—virus scans, checking for cron jobs, watching protocols—had proved fruitless because they were testing the parts. The problem was solved only by someone having end-to-

end understanding of the system.

Sometimes, even having end-to-end knowledge of a system is insufficient. In two famous cases, knowledge of physics was required to track down the root cause of a problem.

The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage ([Stoll 1989](#)) documents the true story of how Cliff Stoll tracked down an intruder who was using his computer system. By monitoring network delays and applying some physics calculations, Stoll was able to accurately predict where in the world the intruder was located. The book reads like a spy novel but is all real!

The famous story “The Case of the 500-Mile Email” ([Harris 2002](#)) documents Trey Harris’s effort to debug a problem that began with a call from the chairman of his university’s statistics department, claiming, “We can’t send mail more than 500 miles.” After explaining that “email really doesn’t work that way,” Harris began a journey that revealed that, amazingly enough, this in fact was the problem. A timeout was set too low, which was causing problems if the system was connecting to servers that were far away enough that the round-trip delay was more than a very small number. The distance that light would travel in that time was 3 millilightseconds, or about 558 miles.

29.6 Summary

Every SA debugs problems and typically develops a mental catalog of standard solutions to common problems. However, debugging should be a systematic, or methodical, process that involves understanding what the customer is trying to do and fixing the root cause of the problem, rather than smoothing over the symptoms. Some debugging techniques are subtractive—process of elimination—and others are additive—successive refinement. Fixing the root cause is important because if the root problem isn’t repaired, the problem will recur, thus generating more work for the SA.

Although this chapter strongly stresses fixing the root problem as soon as possible, you must sometimes provide a workaround quickly and return later to fix the root problem. For example, you might prefer quick fixes during production hours and have a maintenance window reserved for more permanent and disruptive fixes. (See [Chapter 34](#), “[Maintenance Windows](#).”)

Better tools let you solve problems more efficiently without adding undue complexity. Formal training on a tool provides knowledge and experience that

you cannot get from a manual. Finally, in a major outage or when a problem seems peculiar, nothing beats the brainpower of one or more people who together have end-to-end knowledge of the system.

Simple tools can solve big problems. Complicated tools sometimes obscure how they draw conclusions.

Debugging is often a communication process between you and your customers. You must gain an understanding of the problem in terms of what the customer is trying to accomplish, as well as the symptoms discovered so far.

Exercises

1. Pick a technology that you deal with as part of your job function. Name the debugging tools you use for that technology. For each tool, is it home-grown, commercial, or free? Is it simple? Can it be combined with other tools? Which formal or informal training have you received on this tool?
2. Describe a recent technical problem that you debugged and how you resolved it.
3. In an anecdote in [Section 29.4.3](#), the customer was impressed by the methodology used to fix his problem. How would the situation be different if the customer were a nontechnical manager rather than a scientist?
4. Which tools do you not have that you wish you did? Why?
5. Pick a debugging tool you use often. In technical terms, how does it do its job?

Chapter 30. Fixing Things Once

One of our favorite mantras is “fix it once.” Fixing something once is better than fixing it over and over again. Although this sounds obvious, it sometimes isn’t possible, given other constraints. You find yourself fixing something over and over without realizing it, or the quick fix is simply emotionally easier. By being conscious of these things, you can achieve several goals. First, you can manage your time better. Second, you can become a better SA. Third, if necessary, you can explain better to the customer why you are taking longer than expected to fix something.

[Chapter 29](#), “[Debugging](#),” described a systematic process for debugging a problem. This chapter is about a general day-to-day philosophy.

30.1 Story: The Misconfigured Servers

Once, Tom was helping an SA reconfigure two large Sun Solaris servers. The configuration required many reboots to test each step of the process. After each reboot, the SA would log in again. The root account on this host didn’t have `TERM`, `PATH`, and other environment variables set properly. This usually wouldn’t burn him. For example, it didn’t matter that his `TERM` variable was unset, because he wasn’t using any curses-based tools. However, this meant that his shell wouldn’t support command line editing. Without that, the SA was doing much more typing and retyping than would normally be required. He was on the console, so he didn’t even have a mouse with which to cut and paste. Eventually, he would need to edit a file, using a screen-based editor (`vi` or `emacs`), and he would set his `TERM` variable so that the program would be usable.

It pained Tom to see this guy manually setting these variables time and time again. In the SA’s mind, however, he was being very efficient because he spent time setting a variable on demand—only right before it was required if it was required. Occasionally, he would log in, type one command, and reboot—a big win considering that none of the variables needed to be set that time. However, for longer sessions, Tom felt that the SA was distracted by having to keep track of which variables hadn’t been set yet, in addition to focusing on the problem at hand. Often, the SA would do something that failed because of unset variables; then he would set the required variables and retype the failed command.

Finally, Tom politely suggested that if the SA had a `.profile` that set those variables, he could focus more on the problem rather than on his environment.

Fix the problem once, Corollary A: Fix the problem permanently.

The SA agreed and started creating the host's `.profile` from scratch. Tom stopped him and reminded him that rather than inventing a `.profile` from scratch, he should copy one from another Solaris host. *Fix the problem once, Corollary B: Leverage what others have done; don't reinvent the wheel.* By copying the generic `.profile` that was on most other hosts in that lab, the SA was leveraging the effort put into the previous hosts. He was also reversing the entropy of the system, taking one machine that was dissimilar from the others and making it the same again.

As the SA copied the `.profile` from another machine, Tom questioned why they were doing this at all. Shouldn't the automated installation process they used (Solaris JumpStart) have already installed the fine `.profile` that all the other machines had? In [Chapter 8, "OS Installation Strategies,"](#) we saw the benefits of automating OS installation. This environment had a JumpStart server; why hadn't it been used?

It turned out that this machine came from another site, and the owner simply updated its network configuration and got it minimally working on the current site's network. He didn't use JumpStart. This was to save time, because it was unlikely that the host would stay there for more than a couple of days. A year later, it was still there. Tom and the SA were paying the price for customers who wanted to save their own time. The customer saved time but wasted Tom's and the SA's time.

Then Tom identified a few more problematic issues. If the machine hadn't been JumpStarted, it was very unlikely that it got added to the list of hosts that were automatically patched. This host had not been patched since it arrived. It was insecurely configured, had none of the recent security patches installed, and was missed by the Y2K scans: It was sure to have problems on January 1, 2000.

The original problem was that Solaris includes a painfully minimalist `.profile` file. The site's solution was to install a better one at install time via JumpStart. The problem was fixed for all hosts at the same time by making the fix part of the install procedure. If the file needed to be changed, the site could use the patch system to distribute a new version to all machines.

All in all, the procedure that Tom and his co-worker were there to do took twice as long because the host hadn't been JumpStarted. Some of the delays arose because this system lacked the standard, mature, and SA-friendly configuration. Other delays stemmed from the fact that the OS was a minefield of half-configured or misconfigured features. *Fix the problem once, Corollary C:*

Fix a problem for all hosts at the same time.

This was another reminder about how getting the basics right makes things so good that you forget how bad things were when you didn't have the basics done right. Tom was accustomed to the luxury of an environment in which hosts were configured properly. He had been taking it for granted.

30.2 Avoiding Temporary Fixes

The previous section is fairly optimistic about being able to make the best fix possible in every situation. However, that is not realistic. Sometimes, constraints on time or resources require a quick fix until a complete fix can be scheduled. Sometimes, a complete fix can require an unacceptable interruption of service in certain situations, and a temporary fix has to suffice until a maintenance window can be scheduled. Sometimes, temporary fixes are required because of resource issues. Maybe software will have to be written or hardware installed to fix the problem. Those things take time. If a disk is being filled by logs, a permanent fix might be to add software that would rotate logs. It may take time to install such software, but in the meantime old logs can be manually deleted.

It is important that temporary fixes be followed by permanent fixes. To do this, some mechanism is needed so that problems don't fall through the cracks. Returning to our example of the full log disk, it can be tempting on a busy day to manually delete the older logs and move on to the next task without recording the fact that the issue needs to be revisited to implement a permanent fix. Recording such action items can be difficult. Scribbled notes on paper get lost. You might not always have your day planner or to-do notebook on hand. It is much easier to email a reminder to yourself. Even better is to have a helpdesk application that permits new tickets to be created via email. If you can create a ticket via email, it becomes possible to create a ticket wherever you are so you don't have to remember to do this later. Typically, you can send email from your cellphone, tablet, or laptop.

Unix systems can generally be configured to send email from the command line. There's no need to wait for an email client to start up. Simply type a sentence or two as a reminder, and work the ticket later, when you have more time. At Bell Labs, Tom had a reputation for having nearly as many self-created tickets as tickets from customers. It should be noted, however, that many sites configure their Unix systems to properly deliver email only if they are part of the email food chain. As a result, email sent from the command line on other machines does not get delivered. It is very easy to define a simple "null client" or "route all email to a server" configuration that is deployed as part of the

default configuration. Anything else is amateurish and leads to confusion, as email gets lost.

Temporary fixes are emotionally easier than permanent fixes. We feel as though we've accomplished something in a small amount of time. That's a lot easier on our egos than beginning a large project to permanently fix a problem or adding something to our never-ending to-do list.

Fixing the same small things time after time is habit forming. With Pavlovian accuracy, we execute the same small fix every time our monitoring systems warn us of the problem. When we are done, we admonish ourselves to do it properly: "Next time I'll have the time to do the permanent fix!" We get the feeling that we are busy all day but don't feel as though we are accomplishing anything. We ask our boss or co-worker to look at our day with new eyes. When the boss or co-worker does so, he or she sees that our days are spent mopping the floor rather than turning off the faucet.

We have grown so accustomed to the quick fix that we are now the experts at it. In shame, we discover that we have grown so efficient at it that we take pride in our efficiency, pointing out the keyboard macros we have written and other time-saving techniques we have discovered.

Such a situation is common. To prevent it, we must break the cycle.

Case Study: Staying Late

Tom used to run many mailing lists using Majordomo, a very simple mailing list manager that is driven by email messages to a command processor requesting that subscriptions be started and discontinued. At first, he diligently researched the occasional bounce message, finding that an email address on a particular list was no longer valid. If it remained invalid for a week, he removed the person from the mailing list. He became more efficient by using an email filtering program to send the bounces to a particular folder, which he analyzed in batches every couple of days. Soon, he had shell scripts that helped hunt down the problem, track who was bouncing, and determine whether the problem had persisted for an entire week, as well as macros that efficiently removed people from mailing lists.

Eventually, he found himself spending more than an hour every day with this work. It was affecting his other project deadlines. He knew that other software such as GNU Mailman would manage the bounces automatically, but he never had time to install such software. He was

spending so much time mopping the floor that he didn't have time to turn off the faucet.

The only way for Tom to break this cycle was to ignore bounces for a week and stay late a couple of nights to install the new software without interruption and without affecting his other project deadlines. Even then, a project would have to slip at least a little. Once he finally made this decision, the software was installed and tested in about five hours total. The new software reduced his manual intervention to about one hour a week, saving him about four hours a week, the equivalent of gaining a half-day every week. Tom would still be losing nearly a month of workdays every year if he hadn't stopped his quick fixes to make the permanent fix.

Let's dig deeper into fixing the problem permanently. It certainly sounds simple; however, we often see someone fix a problem only to find that it reappears after the next reboot. Sometimes, knowing which fixes are permanent and which need to be repeated on reboot is the difference between a new SA and a wizard.

Many OSs run scripts, or programs, to bring the machine up. The scripts involved in booting a machine must be edited from time to time. Sometimes, a new daemon—for example, an Apache HTTP server—must be started. Sometimes, a configuration change must be made, such as setting a flag on a new network interface. Rather than running these commands manually every time a machine reboots, they should be added to the start-up scripts. Be careful when writing such scripts. If they contain an error, the system may no longer boot. Always reboot a machine shortly after modifying any start-up scripts; that way, you detect problems now, rather than months later when the machine is rebooted for another reason.

Case Study: Permanent Configuration Settings

The Microsoft Windows registry solves many of these problems. The registry contents are permanent and survive reboots. Every well-written program has its settings and configuration stored in the registry. There's no need for every program to reinvent the wheel. Each service, or what Unix calls daemons, can fail to start without interrupting the entire boot process.

Microsoft has fixed it once by providing software developers with the

registry, the Services Control Panel, and the Devices Control Panel rather than requiring them to reinvent something similar for each product.

30.3 Learn from Carpenters

Carpenters say, “Measure twice; cut once.” Measuring a second time prevents a lot of errors. Wood costs money, so a little extra care is a small cost compared with wasted wood. SAs have a lot to learn from carpenters. They’ve been building and fixing things for a lot longer than we have.

Carpenters also understand how to copy things. A carpenter who needs to cut many pieces of wood all the same size cuts the first one to the proper length and uses that first piece over and over to measure the others. This is much more accurate than using the second piece to measure the third piece, the third piece to measure the fourth piece, and so on. The latter technique easily accumulates errors.

SAs can learn a lot from these techniques. Copying something is an opportunity to get something right once and then replicate it many times. Measuring things twice is also a good habit to develop. Double-check your work before you make any changes. Reread that configuration file, have someone else view the command before you execute it, load-test the capacity of the system before you recommend growing, and so on. Test, test, and test again.

Deleting Files Carefully

Unix shells make it easy to accidentally delete files. This is illustrated by the classic example of trying to delete all files that end with `.o` but accidentally typing `rm *. o`—note the space accidentally inserted after the `*`—and deleting all the files in the directory. Luckily, Unix shells also make it easy to “measure twice.”

You can change `rm` to `echo` to simply list which files will be deleted. If the right files are listed, you can use command line editing to change `echo` to `rm` to really delete the files.

This technique is an excellent way to “measure twice,” by performing a quick check to prevent mistakes. The use of command-line editing is analogous to using the first block of wood to measure the next one. We’ve seen SAs who use this technique but manually retype the command after seeing the `echo` results came out correctly, which defeats the purpose of the technique. Retyping a command opens up the possibility of an

accumulation of errors. Invest time in learning command-line editing for the shell you use.

Intel has a philosophy called Copy Exact. Once something is done right, it is copied exactly at other sites. For example, if a factory is built, additional capacity is created by copying the factory exactly at other locations. There's no need to reinvent the wheel. The SAs adopt this policy as well. Useful scripts that are distributed to other sites are used without change, rather than ending up with every site having a mess of slightly different systems. This philosophy forces all SAs to maintain similar environments, develop code that works at all sites without customization, and feed back improvements to the original author for release to the world, thus not leaving any site behind.

You'll never hear a carpenter say, "I've cut this board three times, and it's still too short!" Cutting the same board won't make it any longer. SAs often find themselves trying the same thing over and over, frustrated that they keep getting the same failed results. Instead, they should try something different. SAs complain about security problems and bugs, yet put their trust in software from companies without sufficient QA systems. SAs run critical systems without firewalls on the Internet. SAs fix problems by rebooting systems rather than by fixing the root cause.

Tip: Excellent Advice

In the famous Unix Room at Bell Labs, a small sign on the wall simply states: "Stop Doing Things That Don't Work."

Case Study: Exact Copies Multiplies Learning

All 56 nuclear power plants in France use the exact same design. Because all the designs are the same, they were less expensive to design. Safety is easier to manage because "the lessons from any incident at one plant could be quickly learned by managers of the other 55 plants" ([Palfreman 1997](#)). This is not possible in the United States, with its many different utility companies with many different designs.

System administrators can learn from this when designing remote office networks, server infrastructures, and so on. Repetition makes things easier to manage and multiplies the impact of learning.

30.4 Automation

You can also use automation so that you do not have to perform the fix yourself. One type of automation fixes symptoms and alerts an SA so that he or she can fix it permanently. The other type of automation fixes things permanently, on its own.

Automation that fixes problems can be worrisome. We've seen too much bad science fiction in which the robot "fixes" a problem by killing innocent people or blowing up Earth. Therefore, automation should be extremely careful in what it does and should keep logs so that its work can be audited.

Automation often fixes symptoms without fixing the root cause. In that situation, it is critical that the automation provide an alert that it has done something so that an SA can implement a permanent fix. We have seen automation that reacts to a full-disk situation by deleting old log files. This works fine until the consumers of disk space outpace the log files, and suddenly there are very few logs to delete. Then the automation requests immediate SA intervention, and the SA finds an extremely difficult problem. If the automation had alerted the SA that it was making temporary fixes, it would have given the SA time to make a more permanent fix.

However, now we risk the "boy who cried wolf" situation. It is very easy to ignore warnings that a computer has implemented a temporary fix and that a longer-term fix is needed. If the temporary fix worked this time, it should work the next time, too. It's usually safe to ignore such an alert the first time. It's only the time after that when the permanent fix is done. Because an SA's workload is almost always more than the person has time for, it is too easy to hope that the time after that won't be soon. In a large environment, it is likely that different SAs will see the alerts each time. If all of them assume they are the first to ignore the alert, the situation will degenerate into a big problem.

Fixing the real problem is rarely something that can be automated. Automation can dig you out of a small hole, but it can't fix buggy software. For example, it can kill a runaway process but not the bug in the software that makes it run away.

Sometimes, automation can fix the root problem. For example, large systems with virtual machines can allocate additional CPUs to overloaded computations, grow a full-disk partition, or automatically move data to another disk. Some types of file systems let you grow a virtual file system in an automated fashion, usually by allocating a spare disk and merging it into the volume. That doesn't help much if the disk was running out of space as the result of a runaway process

generating an infinite amount of data, because the new disk also will fill up quickly. However, it does fix the daily operational issue of disks becoming filled to capacity. You can add spares to a system, and automation can take care of attaching them to the next virtual volume that is nearly full. This is no substitute for good capacity planning, but it would be a good tool as part of your capacity-management system.

The solution is policy and discipline, possibly enforced by software. It takes discipline to fix things rather than to ignore them.

Sometimes, the automation can take a long time to create. In most cases, however, it can be done a little bit at a time. The essentials of a five-minute task can be integrated into a script. Later, more of the task can be added. It might seem as though five-minute tasks are taking an hour to automate, but you will be saving time in the long run.

Case Study: Makefiles

A makefile is a series of recipes that instruct the system how to rebuild one file if the files that were used to create it got modified. For example, if a program is made up of five C++ files, it is easy to specify that if any one of those files is updated, the program must be recompiled to make a new object file. If any of the object files are changed, they must be relinked to remake the program. Thus, one can focus on editing the source files, not on remembering how to recompile and make the program.

System administrators often forget that this developer tool can be a great boon to them. For example, you can create a makefile that specifies if `etcaliases` has been changed, the `newaliases` program must be run to update the indexed version of the file. If that file has to be copied to other servers, the makefile recipes can include the command to do that copy. Now you can focus on editing the files you want, and the updates that follow are automated.

This is a great way to record the institutional knowledge about processes so that other people don't have to learn them.

30.5 Summary

Fixing something once is better than fixing something many times over. Ultimately, fixes should be permanent, not temporary. You should not reinvent the wheel and should, when possible, copy solutions that are known to work. It

is best to be proactive; if you find a problem in one place, fix it on all similar hosts or places. It is easy for an SA to get into a complicated situation and not realize that things should be fixed the right way. Sometimes, however, limited resources or end-user requirements leave an SA with no choice other than to implement a quick fix and to schedule the permanent fix for later. At the same time, SAs must avoid developing a habit of delaying such fixes and taking the emotionally easier path of repeating small fixes rather than investing the time necessary to produce a complete solution. In the end, it is best to fix things the right way at the right time.

This chapter was a bit more philosophical than the others. In the first anecdote, we saw how critical it is to get the basics right early on. If automating the initial OS load and configuration had been done, many of the other problems would not have happened. Many times, the permanent fix is to introduce automation. However, automation has its own problems. It can take a long time to automate a solution; while waiting for the automation to be completed, SAs can develop bad habits or an emotional immunity to repeatedly fixing a problem. Nevertheless, good automation can dramatically lessen your workload and improve the reliability of your systems.

Exercises

1. Which things do you fix often rather than implement a permanent fix? Why hasn't a permanent fix been implemented?
2. How do you use the carpenter's techniques described in [Section 30.3](#)?
3. Describe a situation in which you had to delay a permanent fix because of limited resources.
4. Does your monitoring system emulate "the boy who cried wolf"?

Chapter 31. Documentation

In system administration terms, documentation means keeping records of where things are, explaining how to do things, and making useful information available to customers. In general, SAs dislike writing documentation: There's hardly enough time to do the work; why write documentation, too? The reason is that documentation helps in many ways, and a lack of documentation hurts SAs' ability to do a good job.

It can be difficult to decide what to document. Be selfish and use documentation as a tool to make your work easier. Is the helpdesk flooded with the same few questions over and over? Provide documentation so that customers can help themselves. Are there tasks that you dislike doing? Document them so it is easier to delegate them or to have a junior SA take them over. Do you find it difficult to relax while on vacation? Do you skip vacations altogether? Document the processes that only you can do, so that others can do them in your absence. Do you spend days fixing mistakes that could have been prevented? Maintain checklists and hint sheets to improve repeatability.

Documentation is a way of creating an institutional memory that lets an SA team increase its knowledge and skill level. Think of documentation as RAID for the SA staff: It gives you redundancy in the group. Some SAs fear that documentation will make them more replaceable and therefore refuse to document what they do. As a result, they often eventually feel trapped, cornered, and unable to leave their position. They get stressed and, ironically, quit out of frustration. The truth is that, because it simultaneously retains and shares knowledge, documentation enables an organization to promote people from within.

A tiny team can communicate verbally. As the team grows, however, you must change from an oral tradition to a written tradition ([Goldfuss 2015](#)). To do this, we need to create a culture of documentation that includes standard document templates, a storage repository, and ways to make that repository both accessible and searchable.

This chapter offers advice about how to create documentation, how to store it and make it available, and how to manage larger and larger repositories. We stress techniques for making it easy to create documentation by reducing the barriers—real and mental—that prevent people from maintaining documentation.

31.1 What to Document

The things that are most important to document tend to be either things that are complicated and unpleasant or things that you explain all the time. Sometimes, an item falls into both categories—for example, how to access the corporate intranet while traveling. We use the phrase *complicated and unpleasant* to refer to both individual processes and the consequences if you make a mistake. One good example is the process for setting up the IT presence of a new employee: setting up the new hire's computer; creating the accounts he or she needs, including any department-specific aspects; determining whom to notify; and so on.

Thus, if a process has a lot of steps, especially ones for which the order is significant—or messing up requires you to call your manager—it's a good idea to document it as soon as possible. You will be saving someone a lot of trouble, and that someone could be you.

Documenting Disliked Processes

Tom finds that he's bad at doing processes he dislikes. He gets distracted, forgets steps, and so on. By documenting them as a checklist, he is less likely to skip a step and less likely to make mistakes. It's also easier to delegate the process to someone else once the difficult work of creating the process is done.

Documenting the tasks that you dislike makes it easier to find someone else to do them. Often, the difficult part is to create the process itself. Doing the process then becomes easier. When we get permission to hire an additional system administrator for our team, we are often tempted to hire someone with the same skill set we have and to divide the work evenly. It can be difficult to hire someone so senior. If we have documented the tasks that we dislike doing, we can hire a junior person to do them and mentor the person over time and promote from within. A junior person is less expensive and will eventually be someone with the knowledge of how the company and your IT department work and the competency you've fostered.

A job description usually has two parts: the list of responsibilities and the list of required skills. Generate the list of responsibilities by enumerating the processes that you dislike and that have been documented. Generate the list of required skills by drilling down into each document and cataloging the skills and technologies an SA needs to be familiar with to understand the documents. The

entire job description basically writes itself.

31.2 A Simple Template for Getting Started

The most difficult part of creating a document is getting started. Here's a simple formula: Identify the four basic elements of a piece of documentation—*title*, *metadata*, *what*, and *how*. Create a document template or outline that contains the following sections, and then fill everything in:

- 1. Title:** A simple title that others will understand.
- 2. Metadata:** The document's author's contact information, and the revision date or history. People reading the document will be able to contact the author with questions, and when you get promoted, your successors will honor you as the person who gifted them with this document. The revision date and history will help people understand whether the document is still relevant.
- 3. What:** A description of the contents of the document or the goal that someone can achieve by following the directions. One or two sentences is fine.
- 4. How:** The steps to take to accomplish the goal. For any step that seems mysterious or confusing, you might add a *why*, such as "Re-tension the tape (*Why?* Because we find that on a full backup, we get tape errors less often if we have done so, even with a new tape)."

Next, do some basic quality assurance (QA) on the document. Accuracy is important. A typo or a skipped step could result in someone creating more problems than the document seeks to fix.

Follow the steps in the document yourself, typing each command and so on. Next, have someone else use your document to accomplish the goal and to give you feedback on where he or she had any problems. Give the person a copy on paper, making it more likely that the person will take notes to give back to you.

Although it is possible to do this interactively, with you watching and writing down the person's feedback, it's easy for this to turn into a training session, with you helping the person along with the process, while you take mental notes that you then have to write down when you get back to your desk. Be strict with yourself: If you don't let the person do the work, he or she is not testing your instructions. If the person is comfortable with you watching, sit out of view and watch, taking notes on what causes confusion.

After a few people have successfully used this documentation, use it to create a shorter "quick guide" that summarizes the steps. This guide simply helps the

experienced SA not to forget anything. The SA should be able to cut and paste command lines from this document into a shell so that the process goes quickly.

Case Study: Two Sets of Documentation

At Bell Labs, Cliff Miller created a system for maintaining a software depot for homogeneous Unix environments. An important part of the depot's design was the namespace used for the various packages, some of which would be visible only on certain platforms or certain individual hosts. The process for adding software packages into the namespace was a little tricky; although it was well documented, that documentation contained a lot of hand-holding verbiage. This was great the first time a package was added but was cumbersome after that. Cliff's solution was to create a brief "quick guide" that included only the commands to be entered, with brief reminders of what the person was doing and hypertext links to the appropriate section of the full documentation. Now, both new and experienced SAs could easily execute the process and receive the right amount of documentation.

31.3 Easy Sources for Documentation

One way to make documenting your tasks easy is to take notes the next time you do the task. Even if this slows down the process, it is easier than writing the document from memory.

31.3.1 Saving Screenshots

The next time you are doing something that you'd like to have documented, take screenshots as you perform each step. If you create the document from the screenshots, all you need is a line or two of text for each image, describing what is being done in that step. Suddenly, you have a multimedia document, complete with visual aids. It's another cross-check for correctness, since anyone using the document can compare the screenshot to the real monitor at each step and be satisfied that everything looks like it is supposed to before moving on.

31.3.2 Capturing the Command Line

If you are working on a command line rather than with a graphical interface, copy and paste the terminal or console window into the document. Some terminal or console window programs include a capture-to-file feature; the Unix `script` command captures entire sessions to a file.

The Unix `history` command lists the last commands that were executed. These saved commands can be the starting point for documenting a process, as you can convert them into an automated script. Documentation is the first step to automation: If you don't know exactly how you do something, you can't automate it.

The Unix command `script` will capture all output sent to a terminal, including any control characters or escape sequences. These can be cleaned up using the `col` command or other utilities, and cross-checked against the output of `history`.

Whether you capture the output automatically or cut and paste the result afterward, it's better than retyping things from scratch. It is certainly less work and more accurate.

31.3.3 Leveraging Email

How often do you have an email conversation with your co-workers about a given task? Here's a ready source of ad hoc documentation that can be gathered together with a little work.

The two major problems with simply using email for your documentation are that email is not easily shareable and it is likely not well organized. Other people cannot access your email, and you probably can't always find what you want. You're unlikely to have a single piece of email that contains everything you want to have in a document. There is usually a larger message and some back-and-forth smaller messages in which something is solved or communicated. Combine those separate messages in one file, and put it where your co-workers can find it.

If you already save a copy of every email you send, congratulations! You have the material for a number of easy and helpful documents just waiting for you to cut and paste them into the template mentioned in [Section 31.2](#).

You can look in your Sent Mail folder for messages that can be used to assemble a document on a topic. Use your email client's ability to sort by conversations or thread to find good potential emails to turn into documentation. If there were more than one or two exchanges on a topic, the conversation probably contains enough information to be worth turning into a document. Even if your email client does not support conversations, sort by the message subject.

31.3.4 Mining the Ticket System

Another good source of potential documentation is your organization's trouble-ticket or request-tracking system. Some of these systems include tools to create a

solutions database or **solution documents** that can be sent out when new requests arrive that are similar to those already solved.

Many sites use software with a solutions database feature but never use this feature or even enable it. It gets easier to use after the first few times, so enable it and use it now. If you know that you'll be using your ticket system as a resource when writing documentation, you can make the process easier and the ticket information more useful. If your SAs note how they resolved an issue, including pasting commands and output related to that solution, and enter that information in the ticket work log, it quickly improves the value of those logs as shared documentation.

You might customize your ticket system with a flag or “knowledge base” check box and a comment field where you can note the process that needs to be documented. You'll also want a way to get a summary of those flagged tickets—a weekly email report, for example. If your ticket system is not easily extensible in this way, you can create a separate ticket queue for adding things to the knowledge base and simply clone or duplicate tickets into that queue. The original customer-request ticket can be closed and the duplicate left in the separate queue until someone has a chance to turn it into a document. The ticket data enables SAs to quickly write formal documentation. Using these methods usually leads to more documentation being written and fewer complaints about writing documentation. That's a win-win situation!

31.4 The Power of Checklists

A good way to start creating documentation and to function more efficiently in general is to use **checklists**, or lists of things arranged so that each line or paragraph contains only one step. As each item is completed, it can be checked off by making a mark next to it.

Checklists let you complete complex tasks and then verify that you have completed the steps. It's very obvious that you skipped a step if you are marking them off as each one is completed. A checklist can be used by someone less experienced to ensure that every aspect of the process has been finished. Requiring junior staff to file completed checklists or simply to hand them to their manager is a good way to create accountability. For really important processes, especially those that may involve the cooperation of several people or departments to accomplish, a signed checklist can be a good way to document that all the steps have been performed by the appropriate parties. Many SAs use checklists in their own daily task management to keep track of all the steps in complex tasks and all the tasks that need to be done.

Some typical checklists include:

- Tasks to be done for each new hire
- Tasks to be done for each employee termination
- Installation tasks for each operating system used at a location
- Processes for archiving and for off-site storage of data as required by the legal department
- Process for securing the OS before deploying a machine

Add these kinds of checklists to regular documentation, especially for tasks that are performed frequently. After having used the full document many times, the person will need only the checklist.

Automating entire processes can be difficult; automating the most error-prone steps from a checklist can be more manageable. If this process is repeated often enough, the entire sequence will be eventually automated.

Setting up a Shared Directory

When Tom worked for Mentor Graphics, his group maintained a directory on a central server, *homeadm/docs*, that contained informal instructions on various topics. File names were long and descriptive: *how-to-create-accounts.txt* or *reasons-why-printer-p32-fails.txt*. Searching was done with Unix command-line tools, such as *ls* and *grep*. In a very primitive attempt to maintain accuracy, everyone always checked the date of the file to determine whether the information was outdated. Today, a version control system would be used instead.

Although unsophisticated, this system served its purpose well. It was easy to create a new document, easy to edit an old one, and easy to find needed information.

31.5 Wiki Systems

A **wiki** is a web-based publishing and collaboration tool that has revolutionized documentation repositories. The name is derived from *Wikiwiki*, colloquial Hawaiian for *quick*. The first wiki software was called WikiWikiWeb, which begat the generic term *wiki*.

A wiki is a web-based documentation repository that makes it easy for anyone with appropriate access to add and edit documents. Documents may contain plaintext, text with HTML, or text with wiki-specific embedded formatting tags

or commands. A wiki usually has a built-in source code control system to check document files in and out and to retain revision history. More advanced wiki environments include user authentication, automated tagging of updates (date, time, user), per-user locking of files, access control by user or group, and various degrees of granularity of read/modify/publish permissions on individual document directories or subdirectories.

The most powerful aspect of wikis is that anyone (with permission) can update any page. If something is wrong or obsolete, the person who spots the error can correct it or add a note asking for someone to verify and fix the information. Documents magically evolve and stay current. One might wonder what prevents someone from simply deleting random text and adding wrong information; all changes are tracked by user, so someone doing mischief will be identified. In a corporate environment, social pressure will keep such events rare. If they do occur, pages can be reverted to a previous state, using the full revision histories. It is possible to protect pages so that only certain people can modify them—a good precaution for highly sensitive documents.

The wiki formatting commands are very easy to learn and are much easier for nontechnical people to use than HTML is. Many email conventions they already use are supported, such as *this text would appear bold* and *this text underlined*. URLs are automatically turned into hypertext links. Writing the name of another wiki page is detected and turned into a link. Wiki pages are usually named in CamelCase—also known as WikiWords or StudlyCaps—so the software is able to detect them easily. Using a WikiWord that does not relate to a wiki page generates a link that will prompt the user to create the page.

The ability to create placeholder pages for things as you realize that you will need them is extremely useful. It's so useful that until you have experienced it, you might not think it's a big deal. You can create an index or table of contents for a document repository and fill in documents as they are created.

Together, these features create an ideal collaboration tool. It can be very satisfying to see how quickly documents take shape and the repository begins to look real and useful. This evolution encourages other people to contribute and keeps the momentum up.

Using a Wiki

The ease with which documentation of various forms can be created with a wiki was made especially obvious to Strata when she was working at a

small start-up in an old factory building in Seattle. A number of employees, including Strata, were from out of town and tended to travel frequently, taking meals in the office and keeping random schedules. Without any specific requests or directive, bits of useful information started showing up on the intranet wiki site, much as information often shows up on whiteboards in a shared office.

The difference was that the out-of-town employees could participate as easily as those in the office, and the information ranged from contact information for the small company that shared the Internet connection to people's travel schedules to self-rescue advice on the rickety old service elevator that sometimes got stuck between floors. There was even a recipe page for the office rice cooker. The wiki site was an elegant testament to the power of technology to assist self-organizing systems rather than getting in the way.

31.6 Findability

Make it easy to find documents. People don't use documents they can't find. Anything you can do to make it easier for people to find the document they need will increase the adoption rate of your documentation.

People today would prefer to search than read through an index or table of contents. Most wiki systems now have a full text search capability.

Another way to make a documentation system searchable is to have one long table of contents that contains titles and descriptions. People can use their web browser's ability to search within a page to find what they need. The description can include synonyms for technical terms, as well as the technical name for something, the brand name, and the common name. For example, if a document is titled "URL Redirector" but people commonly refer to it as "the link shortener," be sure the description includes the common name.

A **URL redirector** is a web server that takes short URLs as input and redirects them to longer URLs. Users can register their own short-to-long translations. For example, if the system is called `s.company.com`, someone might register that <http://s.company.com/books> should be redirected to <http://www.everythingsysadmin.com/books.html>. Short names are assigned on a first-come, first-served basis. Having an internal URL redirector makes it easy for people to create easy-to-remember short names for wiki documents.

31.7 Roll-Out Issues

A crucial aspect to delivering a new document repository is getting buy-in from the community that will be using it. The users are also the authors, and if a high percentage of folks dig in their heels, the efficacy of the entire project becomes questionable. Ironically, it's many of the old-guard techies who are the most resistant. They've lived in an email-centric world for ages and don't feel that having to go out on the web to participate is respectful of their time. There's certainly an argument to be made for having one inbox of new stuff to monitor!

One way to solve this issue is to make certain that a Recent Changes area is automatically generated as part of the wiki. Some systems permit this page to be periodically emailed to a list of concerned people, or it can be configured to provide the information as an RSS feed so that it can be read as a blog. Providing many ways to access the same information makes it easier for people to adopt.

Adopting Better Documentation Practices

The power of wikis is that they lower the barriers to entry for first-time users, so people are drawn to using them. At one site, Tom hadn't gotten buy-in for a wiki-based system yet, so he installed one on a server and started using it to store his own documentation. Soon, when people asked him where to find certain information, the answer would include a link to the appropriate wiki page.

As people started seeing the wiki used more and more, they started asking for training so they, too, could use it. Tom would resist a little and then "give in." The more he resisted, the more people wanted to use the system. Soon, enough people were using it so that new people assumed that this was the official place for documentation. And then it was.

31.8 A Content-Management System

A **content-management system** (CMS) is a publication system for web sites. For example, a newspaper CMS might facilitate the process of reporters creating stories, which are then queued to editors, edited, and approved for publication. The CMS releases the article at a specific time, placing it on the web site, updating tables of contents, and taking care of other details. For an IT site, the CMS might permit plug-ins that give portal features, such as summaries of recent outages.

A number of features are required to implement a functional CMS. A CMS specifically consists of three layers: repository, history, and presentation. The repository layer is generally a database but may also be a structured file system with metadata. The content is stored in this layer. The history layer implements version control, permissions, audit trails, and such functionality as assigning a global document identifier to new documents. The history layer may be a separate journal or database or may be stored in the repository. The presentation layer is the user interface. In addition to allowing document interactions, such as browsing or editing, the presentation layer may implement document controls, such as read-only access or permissions.

Advanced wiki systems have many of the features of a full CMS. Many CMSs now include wiki-like features. Two popular open source CMSs are MediaWiki and DokuWiki.

31.9 A Culture of Respect

A living documentation site needs a culture of respect; otherwise, people become hesitant to post or feel that they may post only “approved” stuff. Such tradeoffs are inherent in implementing a moderated posting process. Not just anyone should be editing the standby-generator test instructions, for instance, yet the approved author may have missed or changed something in the original posting. This problem can be addressed in most cases by enabling comments to a page and promoting comments into content when they seem valid to the original author or to enough other commenters. When a genuine mistake is made, revision control can enable access to an unaltered version. When it is unclear which version is authoritative, at least revision control can show a list of changes and provide some context with which to make a judgment call or to request clarification via phone or email.

Expect to put in a bit of management time on the wiki until people get the hang of it and match it to your particular culture. There’s a level of detail that will be “right” for your group. As with any kind of writing, there’ll be some bracketing on either side before things start looking as you feel they should look. Some groups have a documentation system that is merely pasted emails in no particular order under topics; other groups have elaborate runbook-style pages for each important piece of infrastructure. The lovely thing about using a wiki or similar living documentation system as an interim CMS is that you can transition from an ad hoc system to a more formalized system as documents mature. Pages and entries can be reorganized and cleaned up during slow days and perhaps promoted to new areas or exported to the more formal CMS.

31.10 Taxonomy and Structure

Wiki systems tend to impose very little structure. Some simply provide navigation.

At the outset, don't spend time worrying about structure. A major killer of wiki projects is the imposition of too much structure during the early stages. The notion of a *low barrier to entry* spawned the wiki. Updating a page should be as easy as sending an email; otherwise, people won't use the wiki. It's far better to have to clean up some pages for readability than to have the information sequestered in someone's mail folders where no one can find it when he is on vacation. If a particular type of document will be written many times by many people—for example, design proposals or feature requests—provide a template to keep those more formal documents in the same format.

Differentiate between writing documentation and organizing documentation. As your content grows, folks will increasingly use the search feature rather than categories to look for things. You can always create structured categories and refactor the pages or their links as the wiki grows.

31.11 Additional Documentation Uses

Here are some more ways to use a living document system. Many wiki systems have templates or plug-ins specific to these applications:

- **Self-help desk:** This area contains current status, major upcoming changes, and the ability to create tickets. It also contains links to other sources, such as the HOWTOs, FAQs, and monitoring system.
- **Internal group-specific documents:** Each group may want to have internal documentation that describes how to perform tasks that only that group can execute. This documentation has a narrow audience and may be less polished than documents aimed at a wider audience.
- **How-to documents:** A how-to or HOWTO document is a short document that describes how to accomplish a particular task.
- **Frequently asked questions:** The FAQs list the most common queries about a particular topic, along with their answers. The answers may point to a HOWTO document or self-help tools.
- **Reference lists:** Reference lists are accumulated lists of things that are not accessed often but that serve a specific purpose. For example, there might be a list of corporate acronyms, vendor contact information details, and hardware compatibility lists.

- **Procedures:** Have a repository for procedures, checklists, runbooks, and scripts. Also have a location for storing pre-and post-testing results and any other documents required for compliance with operational standards.
- **Technical library:** Store vendor documentation, technical articles, and so on, in a single repository that everyone can access.

31.12 Off-Site Links

Users will want to link to external web sites. This is very useful, but it introduces an information security problem, which can be fixed by using an **anonymizing redirection service**.

When most browsers request a web page, the request includes a reference to the page that linked to it. This lets sites track who is linking to them and calculate which of these links are the most successful. If the referring page name is from an internal web site, the referral site will learn that this site exists. The full URL might reveal secret project code names and other information. For example, the referring page might be <http://secret.example.com/project/quickfox/competitors-to-destroy-may27.html>. A site seeing referrals from that URL might give away the big surprise you are planning in May. Just a few calls to reporters, and suddenly newspapers are reporting that your company has a project called “quickfox,” and the big surprise is ruined. The solution is to make sure that the document repository redirects external links through a service that eliminates referral headers.

31.13 Summary

Documentation provides information to customers to enable them to be self-sufficient. Documentation for system administration processes improves consistency, reduces errors, makes it easier to improve those processes over time, and makes tasks easier to delegate.

Using a template to start new documents reduces the effort required to get started and improves consistency. Describing the process being documented is made easier by using screenshots, terminal session captures, email archives, and ticket system archives. Checklists are a good way to document multistep procedures to help you repeat them consistently, document what other groups expect from you, or provide a way for junior SAs to mark an assignment as complete and hand it in to a manager.

Creating a process is difficult and requires a high level of expertise. However,

once the process is documented, more junior people can perform the process. Thus, the process becomes easier to delegate.

Documents should be kept in a repository so they can be shared and maintained. Wikis are a very useful system for hosting repositories because they make it easy to create and update documents and do not require HTML knowledge. Plug-ins extend wikis to do more services.

It can be difficult to get people to start using a documentation repository. Providing help, training, and templates eases the barriers to entry.

Repositories can be useful for more than procedures. Repositories can become self-help desks and store HOWTO docs, FAQ managers, reference docs, and inventory lists.

High-functioning SA teams are generous about documenting processes and sharing information. A good repository can facilitate these practices. Documentation saves you and everyone else time, and leverages everyone's knowledge to make a better environment.

Exercises

1. Which topics seem to come up most often in customer requests at your site? What percentage might be handled by a self-help desk with some HOWTO documents?
2. Describe how you share information with other team members. What works best about it? What would you change?
3. Which documents in your organization would most benefit from having a template? Design the template.
4. Which items would go into a documentation template in your organization? Are any of them unusual or specific to your group?
5. On a scale of 1 to 10, with 10 being most important, how would you rate "ease of use" for a shared document system? Why? How would you rate "access control"? Why?
6. If your site or group does not have a document repository, ask three people why they would resist using it. If your site or group has one but it is under-utilized, ask three people what would make it easier to use. How would you mitigate these issues?
7. Of the solutions for shared-document systems discussed in this chapter, which ones best fit with your answers to the two previous questions?

Part VII: Change Processes

Chapter 32. Change Management

This chapter is about formal change-management processes such as the ones that exist in large enterprises. Big changes require a lot of care. Many people making many changes at the same time requires coordination. An organization's change-management process seeks to make all of this work.

Most of what SAs do involves making changes. We fix someone's PC, we configure a port on a network switch, we plug in a cable, we write code and upload it to a server. Every time we run a program, we are, in fact, changing the system.

Most changes are small and have only a localized impact. They affect just a very small part of the system as a whole. You probably make hundreds or thousands of such changes every day.

Other changes are large and have a bigger impact. They may affect hundreds, thousands, or millions of users. They have increased visibility. If they require a service outage, they can cost millions of dollars in productivity. Any unintended problems could result in millions of dollars in lost revenue.

In large enterprises there are often hundreds or thousands of engineers making changes at the same time, and they often work in many different IT organizations. In such a setting, some kind of coordination is required so that people aren't interfering with one another's work.

This complexity also extends to communicating about changes, both to other IT organizations and to the users themselves. Therefore most enterprises have a change review board (CRB), a committee made up of representatives who review change requests and approve them. The committee members are also responsible for communicating about changes to their divisions or organizations.

While the majority of this chapter focuses on the macro issue of running a CRB, it ends with a discussion of change management on the micro scale. Specifically, it considers how one team can share responsibility for systems in a coordinated fashion and not have team members step on each other's toes.

Case Study: IBM's Olympics Failure and Success

IBM built and operated the computer infrastructure to support the 1996 Summer Olympics in Atlanta and the 1998 Winter Olympics in Nagano. The 1996 experience was a technical and PR disaster. IBM redeemed itself in 1998.

In the 1996 Atlanta Games, IBM did not have a change-management review process, and many changes were made by programmers who were unaware of the impact their “small” changes would have on the rest of the system. Some systems were completed “just in time,” with no time for testing. Some were still being developed after the Games had started. There were many problems, all widely reported by the press, much to IBM’s embarrassment. The outages prevented the system from getting out information about the athletic events and left the press with little to write about except the fact that IBM’s computer system wasn’t working. It was a public relations nightmare for IBM.

A root-cause analysis was performed to make sure that these problems were not repeated at the 1998 Winter Olympics. It was determined that better change management was required. IBM implemented change-management boards that had up to ten representatives from various areas of the project to review change proposals. Through this mechanism, IBM successfully managed to prevent several similar small changes from occurring. In addition, all the hardware and software was completed and fully tested before the events, with many problems discovered and fixed in advance. The final result was that the information system for the 1998 Winter Olympics ran smoothly when the events started.

The full story can be found in Guth & Radosevich ([1998](#)).

32.1 Change Review Boards

A CRB is responsible for assuring that a business is able to make changes required by the business, in a coordinated manner, in a way that attempts to prevent unintended side effects. To achieve this, the CRB takes responsibility for the following tasks:

- Defining change-management processes
- Establishing definitions and standards for rating change risks
- Reviewing (rejecting or approving) change requests
- Coordinating post-change activities

The purpose of the CRB is to assure all changes being made are a success. It achieves this by stopping bad changes from being made and preventing conflicts, while letting good, well-prepared changes through. A mature change-management process can also take on overtones of project management, with a proposed change being carefully considered for its impact on not only other

systems but also other deadlines the group must meet. If making the change will cause other, more important deadlines to slip, it will be refused.

The CRB establishes a process by which change requests are formally proposed, evaluated, and approved. It establishes criteria for which type of changes require this kind of formal approval. People wishing to make such changes fill out a form explaining the change and the backout plan, and provide evidence such as testing criteria. The CRB reviews and approves these changes. The approval process creates an opportunity to verify processes are being followed, coordinate among many organizations, and plan appropriate communication about the change.

Most people think of a CRB as a committee that meets once a week to approve or reject change requests. However, a CRB does much more than that. It sets a high bar for the kind of preparation and planning that ensures changes will be successful.

The existence of a change process forces SAs to think through various aspects of a change before they implement it. Doing this kind of preparation, even if there isn't a CRB, improves the chance of success.

A CRB can also be thought of as a communication tool that makes sure everyone is on the same page when changes are made. It is also a way to assure that major changes are done consensually. If a CRB includes representatives from each division of a company, it assures informed consent among all stakeholders.

Small and medium-size organizations usually do not need a CRB. It is sufficient to have a consistent policy regarding which changes are big enough to be worthy of notification, and a consistent template for sending notifications to users.

Some describe a CRB as a business function whose goal is to minimize business risk. We prefer to think of a CRB as a function that increases success. If minimizing risk was the primary goal, the CRB would simply reject all change requests. The company would be paralyzed, but risk would be minimized. Alternatively, the CRB could simply make it very difficult to make any changes. The CRB would then become a speed bump or roadblock that hinders changes from being made. This would result in teams finding sneaky ways to circumvent the CRB, which means that they would not benefit from the coordination and quality assurances that a good CRB provides. The result would be more confusion, more outages, and increased risk. The process of submitting proposed changes and getting them approved should be as easy as possible so people

won't avoid the CRB.

32.2 Process Overview

The process for most CRBs generally follows a particular pattern. People write proposals on which changes will be made. The CRB meets weekly to discuss and approve each one. At the meeting each pending request is announced, the person presents the case for that change, the committee asks questions, and the request is approved or rejected.

The discussion of each item can be a major drag on the process. There are two ways to make it go faster. First, proposals should be submitted early enough that members can review them ahead of time. In fact, it should be expected that people will read proposals ahead of time rather than during the meeting. Reading them during the meeting wastes everyone's time.

Second, proposals should be pre-checked by a member of the committee before they are seen by the full board. This is called a quality review. During the review, typos and minor issues are fixed, common mistakes are addressed, and gaps are filled. Pre-checking is helpful to people who are new and need to be walked through the process. It helps everyone by assuring that meetings run more smoothly. The quality review is performed by a quality checker. This role may be the responsibility of the committee chair, or the responsibility may rotate among the committee members, or each proposal may be randomly assigned to a committee member.

32.3 Change Proposals

A change proposal should be a structured document or form. This assures that all the needed information is collected. It also makes the process less intimidating by walking people through the questions that will be asked during the change review process.

The SAs have to fill out a **change control** or **change proposal** form. The form should ask for a detailed explanation of which changes they will make, the systems and services affected, the reasons for the change, the risks, the test procedure, the backout plan, how long the change will take to implement, and how long the backout plan takes to implement.

Some organizations require the SA to list the exact commands they will type. The level of detail required varies from site to site and usually depends on how critical the affected machine or service is. For very critical machines, the SA cannot type anything that is not listed on the change-control form that was

approved. By comparison, the process and documentation requirements for less critical machines should be less stringent, or SAs will find that their hands are tied by change-management red tape and they are unable to work effectively.

32.4 Change Classifications

The CRB is responsible for defining change classifications so that they are consistent across all parts of the organization. Most CRBs use classifications similar to these:

- **Routine update:** There is no expected outage and a near-zero chance of unintended outages. The potential for such changes to create an outage is limited by the very nature of the task. Such changes can happen at any time. They do not require CRB approval. Examples include adding and deleting users, installing a new PC, helping an individual customer to customize his or her environment, debugging a problem with a desktop or a printer, and altering a script that processes log files to produce statistics.
- **Major update:** Many systems are affected or the change requires a significant system, network, or service outage. What is considered a major update varies from site to site. For most sites, anything that affects 30 percent or more of the systems is a large update. Examples include upgrading shared services such as the authentication system, the email or printer infrastructure, or the core network infrastructure. Major updates should be relatively rare. If they aren't, reconsider your classification system or investigate why your system requires so much churn.
- **Sensitive update:** No outage is expected, but the change is large and could result in an unintentional outage. Examples include altering one replica in a redundant set, and some changes to routers, global access policies, firewall configurations, and critical servers.
- **Emergency update:** This change must be done before the next CRB meeting. The change may require approval by one CRB member before executing it, followed by full approval at the next CRB. It is rolled back if it is not approved. Examples include responding to security attacks, reverting approved changes from the past where the negative impact is significant but not noticed until later, and fixing problems that result in revenue loss per minute, or regulatory or legal consequences.

When trying to classify updates in this way, take into account that some changes may be considered to be routine updates at some sites and sensitive updates at other sites or even in different areas at the same site. For example, at an e-

commerce company, attaching a new host to the corporate network may be considered a routine update, but attaching a new host to the customer-visible service network may be considered a sensitive update. Consider how various sorts of updates should be categorized in the various areas of your site, and institute a scheduling practice that reflects your decision.

CRB review and approval is not required for routine updates. In fact, the purpose of defining routine updates is to delineate between what is and isn't subject to change control.

32.5 Risk Discovery and Quantification

The better we understand the risk of a proposed change, the more intelligently we can make decisions related to the change. This process, which is called risk discovery and risk quantification, involves asking questions: Which systems and services can a change impact? What are the worst-case scenarios? How many of your customers could these scenarios affect?

Change requests should be rated as high, medium, or low risk, with different levels of review required for each category. The risk rating is based on a combination of the potential impact of a failure and the likelihood of that failure.

For example, in [Table 32.1](#) the impact rating indicates the impact if the change goes wrong—the worst-case scenario. If you already have an incident impact ranking system, it is easy to define: Which level of severity incident could you cause in the worst-case scenario? For example, a company might have severity 1 to 6, where 6 is no impact, 5 is single user impact, all the way up to 1, which is financial loss greater than \$1 million, severe damage to the company's reputation, and serious regulatory impact (for highly regulated industries). Therefore, changes with the potential to cause incidents of severities 1 and 2 are considered high impact, 3 and 4 are considered medium impact, and 5 and 6 are considered low impact. The likelihood indicates the chances of this worst-case scenario happening. You rate both the impact and the likelihood, and then use the change risk matrix to determine the risk rating of the change.

	Likelihood		
	Low	Medium	High
High Impact (Severity 1 and 2)	Medium	High	High
Medium Impact (Severity 3 and 4)	Low	Medium	High
Low Impact (Severity 5 and 6)	Low	Low	Medium

Table 32.1: Change Risk Matrix

Changes to systems that are critical to the business have a high impact rating. Suppose the chance of a failure occurring for a particular type of change is low. Using [Table 32.1](#), we would look at the high impact row and the low likelihood column, and see that the change should be rated as medium risk.

A rule change on a firewall that is used by 1 percent of the company might be classified as a low-impact change, with a low likelihood of problems, resulting in a low risk rating. However, this change is still subject to change-management processes, as the change is not classified as routine, but rather as a sensitive update.

The quality checker should review the risk quantification and verify that he or she agrees with the assessment. Then, based on this assessment, the CRB knows how thorough the review of the change request must be. The committee members should spend proportionally more time reviewing a high-risk change than a low-risk change.

32.6 Technical Planning

It is not sufficient to simply list what the change will be or which commands you intend to type to make the change. A complete plan has these five primary parts:

- **The change process:** Defines the steps of the change.
- **Testing documentation:** Shows that the change process has been successfully tested in a test or staging environment.
- **Success criteria:** Explains how the success or failure of the change will be assessed.
- **Backout plan:** Indicates how the system will be changed back to the original state or configuration if the change is not successful.
- **Decision point:** Defines how and when you should decide to implement a backout plan.

The completed change plan, which should include all of these parts, should be reviewed and approved by the CRB. The change process itself should be defined in a repeatable way. Repeatable means that it is documented in a way that someone else could follow the instructions and receive the same result.

The change is performed in advance in a test lab or test staging environment. Some networks are too complex to be duplicated in a lab. In such cases, simulators are used instead. The testing documentation should show that the testing was appropriate and successful. Ideally, this should be the output of

automated testing. The same tests that were used in the lab should be included as part of the success criteria.

Success criteria determine whether the change that you made was successful. Such criteria do not just mean verifying that the change happened. They also mean verifying that the change made had the intended result, and no unintended results. Usually this requires verifying that the service you changed is running properly, and that all services that depend on it are still working properly, or at least that they have not stopped working because of the change that you made.

A backout plan specifies the exact steps that will be taken if it is discovered that the change has not worked as expected. The steps for how to do this should be documented in advance, so that you do not need to try to figure them out while under pressure.

Some sites allow some debugging and corrective actions for changes that did not work correctly at first. Other sites require management approval before changes that were not in the approved runbook can be made, and require that the change ticket be updated with all the steps that were performed. Others may forbid such deviations from the plan—sometimes this prohibition is for regulatory reasons.

The decision point indicates how you will decide whether to continue debugging or to activate the backout plan. It may be based on a time window or other criteria.

Usually the decision point is simply based on the time window and the expected duration of the backout plan. For example, if the change was allocated a 2-hour window of time, and the backout plan takes 30 minutes to execute, the decision point is 90 minutes into the time window. If the change has not been successful, all debugging will stop and the backout plan will be activated at this 90-minute mark.

Sometimes the decision point may be based on other criteria. For example, there may be a known risk that the change will result in a particular problem, or the change is intended to test an assertion that can't be tested in a lab environment. The decision point may include that the backout plan will be activated if a known bug is triggered or if the change does not have the desired effect after five minutes. In such cases the exact criteria should be specified: a command that can be executed to verify that the bug has not been triggered, not simply that “everything looks good.”

The decision point is often the most difficult component to implement for the SA making the change. We often feel that if we could spend just five more

minutes, we could get it working. It is often helpful to have another SA or a manager to keep you honest and make sure that the backout plan is implemented on schedule, if the change is unsuccessful.

The CRB approves the plans after verifying that the preparation, test plan, backout plan, and decision points have been documented and are of the required quality. Technical planning is not the only criterion for their approval—there are also scheduling and communication concerns, which we discuss next.

32.7 Scheduling

When a change is executed depends on the type of change and corporate culture. Some companies may want major updates performed at off-peak times, and others may want all of them to happen in one concentrated maintenance window.

Sensitive updates should happen outside of peak usage times, to minimize the potential impact and to give you time to discover and rectify any problems before they affect your customers. Peak usage times may vary, depending on who your customers are. If you work at an e-commerce site that is used primarily by the public in the evenings and on the weekends, the best time for making changes may be weekdays at 9 AM.

In some environments major and sensitive updates are done late at night and on weekends. There are two problems with this. First, SAs might be sleepy late at night, which can lead to mistakes that cause big outages. Second, if the change causes a problem, finding people to help will be more difficult because the rest of the team may be unavailable. If they are available, they may not be in the best condition to work.

To strike a balance, ideally systems should be upgraded during the day, Monday through Friday. This avoids both of the previously mentioned problems. However, some systems are not designed with live upgrades in mind. Newer software systems are more likely to be designed to permit live upgrades. Networks and physical hardware tend to hinder live upgrades. In those cases a balance between outage risk, the alertness of the people performing the tasks, and the availability of people who may be pulled in to help must be achieved. In all these cases extensive testing ahead of time will improve the likelihood of success.

The DevOps movement advocates for automated, continuous testing as a way of improving this situation by increasing the confidence and velocity with which changes can be made. While these techniques started in the software world, the principles can be applied everywhere.

Case Study: Bell Labs' Demo Schedule

The Bell Labs research area had a very relaxed computing environment that did not require too much in the way of change management. However, there was a need for an extremely stable environment during demos. Therefore, the research area maintained a simple calendar of when stability was needed.

Researchers notified the SAs of demos via the usual helpdesk procedure, and the SAs paid attention to this calendar when scheduling downtime. They also avoided risky changes on those days and avoided any kind of group event that might take too many SAs away from the building. If the demo included CEOs or heads of state, an SA stood ready outside the door, poised to jump into action.

32.8 Communication

The bigger the change, the more communication is required. We must make sure that the IT organization knows what is happening and customers also know what is going on.

Other SAs need technical details. When everyone on a team is well informed about changes, the SAs can all keep their eyes and ears open for problems that may have resulted from the change. Any problems will be spotted sooner and can be fixed more quickly.

Customers need to understand how they will be impacted: the date and time the change will occur, any actions they'll need to take before the change, what they'll see after the change, and any new behavior required.

If the changes involve a hard cutover, after which the old service, system, or software will not be available, you must make sure that all your customers who use the old version will be able to continue to work after the new version is implemented. If a soft cutover is involved, with the old version remaining available for a time, you should ensure that everyone knows in advance when it is happening, how they can use the older version if they need to, and when or whether the old version will no longer be available. If you are adding a service, you need to ensure that the people who requested it, and those who might find it useful, know how to use it when it becomes available. In all three cases, let your customers know when the work has been successfully completed and how to report any problems that occur.

Take care not to flood your customers with too many messages. If you do, the

customers will ignore them, thinking them irrelevant. Targeting the correct groups for each service requires understanding your customer base and your services so as to make appropriate mappings. For example, if you know that group A uses services A to K and group B uses services B, D, and L to P, you need to let only group A know about changes to service A, but you should let both groups, A and B, know about modifications to service B. This task may seem tedious, but when it is done well, it makes a huge difference to your customers. Large companies will automate this communication through their change-management tool, recording which groups rely on which services, and maintaining a list of people to notify for each group. A small team of people in each group is responsible for reviewing changes that affect the group, and for disseminating the information within their group, as appropriate.

The most effective communication method will vary from company to company and depends on the company culture. For example, in some companies, a mailing list that people choose to subscribe to may be the most effective tool. Other companies may use other means of communication, such as internal corporate social media systems like Facebook at Work. For significant changes, it's best to send a message out to people ("push") rather than to require your customers to check a certain web page every few days ("pull").

Larger companies have change-management tools that require explicit approval from certain groups, depending on the service being affected. Other groups will be notified of the change, and can ignore it, approve it, reject it, or ask for more information. An explicit approval from these groups is not required, but if they reject the change, it will be canceled. Since the change cannot go ahead without all the mandatory approvals, this forces the SAs to reach out to the people who have not yet reviewed the change to get approval. It is important not to have too many mandatory approvers for each change, as it introduces additional overhead for the SAs, who have to chase the approvals, and becomes a nuisance to the approvers, who are continually asked to approve changes that they may view as irrelevant. Getting the balance right for core infrastructure changes can be particularly tricky.

Presenting a Change Request to the CRB

Bringing your request to a CRB can be scary and intimidating. Being prepared and staying calm is key to getting your change request approved. Here are some tips that can greatly enhance your chance of success:

- **Never present before a CRB whose meetings you haven't attended previously.** Attend at least one CRB meeting ahead of time so that you

understand the process. You will learn from other people's mistakes. Should you be humble or confident? Do the board members appreciate humor or are they overly serious? Do certain behaviors make the CRB members irritated? Happy? If you rarely present requests, observe a CRB meeting once a year just to stay current.

- **Understand the audience.** While officially the entire CRB must approve a change, in some organizations there's one person who is really running the show. Find out if people present to the entire room or there is really just one person everyone is trying to appease. If the meeting is held by teleconference, observe from the board chair's location to best see the power dynamic.
 - **Have your request reviewed ahead of time by someone on the CRB.** Bring your proposal to a CRB member ahead of time. He or she will be glad to review it and correct mistakes or issues that will cause the meeting to get bogged down with confusion. Board members don't enjoy attending long meetings, so anything that can be done to make them run more smoothly will be appreciated. Tell the CRB member you are nervous about going to the meeting: He or she may offer advice that will calm you down.
 - **When attending, keep it simple and be brief.** Describe the change as briefly as possible. In theory the CRB has read the request ahead of the meeting. Give a high-level overview in one sentence and wait quietly for questions—do not be tempted to fill the silence. If there are no questions, then the CRB will probably just approve your request and move on to the next change.
 - **For big changes, get buy-in ahead of time.** For large, complex changes, speak with all key stakeholders days ahead of time. Ideally you should walk into the CRB meeting with most of the CRB members already onboard with your change and ready to advocate for you.
-

32.9 Tiered Change Review Boards

Large companies have a tiered system of CRBs, providing different levels of review depending on the risk rating for the change. High-risk changes are subject to more levels of review than medium-risk changes, and low-risk changes require fewer levels of review.

The lowest-level CRBs are the technical management teams for each technology. For example, there might be a Core Infrastructure CRB, a Linux

Server CRB, a Windows Server CRB, and a Windows Desktop CRB. All changes within a technology stream that are subject to the change-management process are subject to review by the CRB for that business stream. The CRB members' primary function is to ensure that the change has been properly planned and tested, the backout plans are in place, the risk rating is appropriate, the change is scheduled for a suitable time, and the relevant people have been notified. They also look for changes that are particularly significant, or that may have impact across other streams. They "promote" these notable medium-risk changes to the next-tier CRB for review and coordination across the technology streams.

The next-level CRB includes higher-level managers within the IT organization. They review high-risk changes and any medium-risk changes that have been promoted by the technology stream CRBs. They will be aware of other medium-and high-risk changes, and can look for conflicts, prioritize the changes, and ask for some to be rescheduled, if necessary. In a large company there may be several of these CRBs, to spread the workload. Depending on how the company is structured, they may be organized by technology, business unit, or region. In a large organization with several CRBs at this tier, they decide which high-risk changes should be promoted to the next-tier CRB for further review and global coordination across all business units and technologies.

The final CRB tier reviews all promoted high-risk changes. This CRB has an overview of all the major initiatives in the company, both from an IT perspective and from a business perspective. This CRB reviews each change in light of the business needs and the other high-risk changes. This group may maintain a master calendar, where major IT changes can be scheduled. The same calendar would also identify key business events, such as month-end closing, tax preparation, product release dates, trade shows, and high-profile demos.

32.10 Change Freezes

CRBs often institute a change freeze—that is, a time when sensitive and major updates are not permitted. Change freezes typically occur at the end of a quarter and the end of the fiscal year. [Figure 32.1](#) is an example of one company's change-freeze announcement that is sent out to all SA staff and department heads. It includes a change-control form that is part of the change-management process. Some companies call this "network quiet time."

Subject: FYI - QUIET TIME IS COMING 09/25 - 10/06

Team

Just a reminder for you all that QUIET TIME will be here in three weeks.
It is scheduled to begin on 9/25 and go to 10/06.

Change control 96739 is below:

CHANGE SUMMARY DISPLAY		CHANGE: 00096739
Assignee Class/Queue...	GNSC	Change Status/Type OR/INF
Assignee Name.....	-----	IPL/Service Disrpt N/N
Requester Name.....	FRED/ADDAMS	Risk/Chg Reason... 1/QT
Enterer's Name.....	FRED/ADDAMS	Proc Ctr/Cntl Ctr. NET/GNS
Enterer's Phone.....	(555)555-8765	Problem Fixed..... -----
Enterer's Class/Queue..	GNSC	Business Unit..... ALL
Plan Start Date/Time...	09/25/2000 00:01	Location Code..... GLOBAL
Plan End Date/Time.....	10/06/2000 24:00	COI..... ALL
Date/Time Entered.....	04/10/2000 14:26	Approval Status... PENDING
Date/Time Last Altered	06/22/2000 16:02	User Last Altered. NCCOFHA
Date Closed.....	-----	Associated Doc.... N/A
System.....	-----	
Component/Application..	FISCAL-PROCESS&NTWK-QUIET-TIME	
Description.....	4Q00/1Q01 FISCAL EXTENDED AVAILABILITY	
System edited.....	-----	Loc added.... -----

Fiscal processing, email, and network quiet time to support
quarterly book close/open activities.

Changes that may impact access to or data movement
between server/mainframe applications or email should be
rescheduled. Only emergency changes to prevent or
fix outages will be reviewed for possible implementation.
All changes will require a change exception form be
submitted to the CMRB.

See URL for Quiet Time Guidelines and contact information:

<http://wwwin.foo.com/gnsc/quiet-time.html>

Customer Impact: None

Test plan: None

Contact and Phone/Pager Numbers:

JOHN SMITH.....(555)555-1234

JANE JONES.....(555)555-4321

ALICE WALTER.....(555)555-7890 800-555-5555 pin 123456

Backout Plan: None

*** BOTTOM OF DATA ***

Figure 32.1: Sample change freeze, or network quiet time announcement

Case Study: NetAid—Aided by Change Management

When it ran the first NetAid event in 1999, Cisco had roughly 4 weeks to build and run a distributed network that had to handle 125,000 simultaneous video streams across 50 ISPs. Cisco had to develop mechanisms that scaled across the planet. In the end, Cisco had nearly 1,000 pieces of hardware to manage and occasionally change. The company did this with a full-time staff of about 5 people and many volunteers.

Before this NetAid event, no one had ever scaled to this size. The staff knew operational challenges would require changing the router and server configurations. With lots of people and a volatile environment, it was important for the staff to maintain configuration control, particularly because the reasoning behind certain routing configurations was not intuitively obvious—for example, why did Paul put in that particular route filter? In addition, the staff used an intrusion-detection system to guard its e-commerce site.

In the 4-week time frame, hundreds or thousands of changes would be made by a team of people too large for everyone to be aware of every change. The change-management process provided a documentation trail that enabled everyone working on the system to understand what had been done before, why it had been done, and how their changes fit in with the others' work. Without this documentation, it would have been impossible to get the system working correctly in time. Such processes all contributed to the project's success.

Other DevOps pundits dismiss the notion of a change freeze, pointing out that there is no such thing as an actual change freeze. Every time a program sets a variable, it is a change. A real change freeze would involve powering off the network. The goal of using change freezes to reduce risks is noble, but DevOps seeks to reduce risk not by avoiding it, but by using techniques to improve the ability to make changes with confidence. As discussed in [Chapter 2](#), “[The Small Batches Principle](#),” many small changes are less risky than big changes. Continuous integration and delivery (CI/CD) builds confidence in our ability to successfully make changes.

Realistically, a hybrid approach is needed. Network equipment and legacy systems are not managed using DevOps strategies, so a change freeze becomes the default option. Even in an organization that is a DevOps utopia, change

freezes during the December holidays have the benefit of permitting everyone to align their vacation schedules.

32.11 Team Change Management

The CRB process coordinates the macro process of change management. On a personal or team level, there are other things you can do to assure every change is a successful change.

While not every change requires the formal planning involved in writing a change request, the elements of such a request should be kept in mind when doing any kind of a change. Develop and test the change in a lab or test area, and develop success criteria and a backout plan. Understand at which point you'll revert the change if something goes wrong or is progressing more slowly than expected. Having your plan reviewed by someone else, even a team member at a whiteboard, is a simple way to prevent outages. In other words, you can take the entire CRB process and do a mini-version in your head, or on a whiteboard, and obtain many of the same benefits.

Here is some advice on scheduling, preventing conflicting changes, and keeping a history of changes.

32.11.1 Changes Before Weekends

It is a good idea to avoid big changes on Friday or the day before you leave for a long trip. If the change turns out to cause a major outage, you may spend the next few days fixing it. In the best case you ruin your weekend; in the worst case your family leaves for vacation without you and you never live it down.

However, having an organizational policy against changes on particular days can also snowball into a bad situation. One SA team received software updates from a developer group to be deployed in production. After one failed deployment resulted in an unhappy weekend, Friday deployments were banned. Then the ban on Monday deployments came because SAs needed that day to catch up from issues that arose over the weekend. Over time the window of opportunity for upgrades became more and more narrow.

Such rules unintentionally (or passive-aggressively) send a message: Never make changes. This is bad. A company cannot improve if change is discouraged by policy or by implication.

The tacit implication that developers shouldn't ship code is the equivalent of telling them they can't do their job. As a result, people work around the system administrators. The configuration file includes a complete programming

language so that changes can be made without the SA's approval. Or, even worse, backdoors are installed that permit the developers to upgrade the system behind the SAs' backs.

A better solution is to work together and adopt techniques that permit rapid deployments without sacrificing risk. Refer to the DevOps techniques in [Chapter 20](#), "[Service Launch: DevOps](#)," as well as Google's Error Budgets techniques, documented in Volume 2 of this book series.

The Reboot Test

Before distributed computing, most departments (and sometimes entire organizations) had only one big computer that everyone accessed through terminals or modems. When Tom was in university, the one big computer was a VAX 11/750 running Digital's VMS operating system. Now, before you young whippersnappers yawn at the antique 1980s technology and skip to the next section, there is a lesson here that is still valuable today, so keep reading.

The script that executed on boot-up was rarely changed. However, if someone introduced an error into the script, the machine wouldn't boot and fixing it required editing the script from a paper console.

One day someone changed the script and didn't test it. The next reboot was weeks later and everyone had forgotten about the change. The failed reboot was assumed to be some other problem and people spent hours debugging the wrong problem.

After that, the SAs developed the following rule: If you change the startup script, first you have to reboot the machine with the script unchanged. Then make the change. Then reboot immediately after.

In these old days a reboot took 10 to 20 minutes. Meanwhile, hundreds of users had no access to the world. The benefit of this technique was that mistakes were discovered soon after they were made, when the change was still fresh in the SA's mind.

32.11.2 Preventing Injured Toes

If multiple people are administering the same system at the same time, there needs to be a way to assure that people aren't stepping on each other's toes. Two people making conflicting changes to the same system can be a disaster.

In the old days of large Unix servers, it was common to edit configuration

files using a mechanism that would lock the file so that only one person could edit it at a time. Today it is common to use a chat room to warn people of changes being made. Writing “CHANGE: Rebooting host ny-web01” or “CHANGE: I’m about to edit the load-balancer configuration” is informal, but such a message includes enough information to prevent collisions.

32.11.3 Revision History

A lot of system administration involves editing configuration files. Keeping a history of revisions to a configuration file makes it easier for a team to work together. They can see who made which changes, and most revision-control systems have the ability to document why a change was made.

It’s a good idea to maintain such files in a source code repository and have them pushed out to machines, or to generate the configuration files as mentioned in [Chapter 4](#), “[Infrastructure as Code](#).” Both of these approaches permit centralized revision control.

However, not all configuration files are always maintained that way. Sometimes we just want an easy way to track the changes to a particular machine’s `etc/selinux/semanage.conf`. The open source package EtcKeeper makes it easy to track the configuration files of a Linux system in Git, with a central repository that stores the Git history of all systems.

Tools for storing a revision history include Git, Mercurial, Subversion, Perforce, and Microsoft SourceSafe. One of these tools is probably already in use by developers in your organization.

32.12 Starting with Git

It’s easy to maintain the revision history of a file in Unix with Git. Start doing this the next time you edit any file and, before you know it, you’ll have a simple but efficient revision history for all your important configuration files.

If you have never used Git before, you will need to install it and do some basic one-time configuration to specify your user identity, favorite editor, and `diff` tool. Instructions for these simple steps are readily available online.

Suppose that you want to start maintaining a revision history for configuration files in `/etc`. First use the command `git init` while in the `/etc` directory, which will create a `.git` subdirectory with all the necessary files for you to get started. Now add files that you want to start tracking, with `git add file1 file2 ...` or simply `git add .` to add everything. Then use `git commit -m 'initial version'` to commit the current version.

Now you can edit any file, use `git status` to see which files have been changed, `git diff` to see the changes, `git add` to stage a modified file for committing to the repository, and `git commit` to commit any staged changes into the repository.

For example, if you want to modify the file named `named.conf`, you should first check whether there are any changes with `git diff named.conf` and if there are, stage them with `git add named.conf`. You can now edit the file with your favorite editor. At the end, verify that the changes are as expected with another `git diff named.conf`. If you are happy with the changes, stage them with `git add named.conf`. Check the status of the whole repository with `git status` and commit the changes with `git commit`.

If you see from the status check that there are other modified files, ask your co-workers if they are currently working on something, or might have made some changes in the past but not checked them in. Then, if no one is actively working on these files, review the differences, and then stage and commit the files to the repository.

Using a local repository allows you to get started now. You can always move to a more sophisticated centralized setup later. Refer to a good reference book, or an online source, for more complicated issues, such as backing out of changes. Create a simple text file to experiment with while reviewing the manual pages. You'll be an expert in no time.

Case Study: Revision History Saves the Day

A midsize software company had a script that automated the process of account creation. One day, the disk that contained the account database filled up while the program was rewriting the database. Because it did almost no error checking, the script failed to notice the problem. It proceeded to push out the new account database to all the authentication servers, even though it was missing most of the accounts.

The SAs quickly realized what had happened and were able to immediately clear some space and go back to the old account database as it existed immediately before the script was run. Without a revision history, they would have been forced to restore the file from backup tape, which would have taken much longer and meant that any password changes customers had made since the backup would have been lost.

The automated program had identified itself in the optional comment field, so it was easy to track down the entity responsible for the

truncation. The script was subsequently changed to do a lot more error checking.

32.13 Summary

Change management is a valuable tool that mature sites use to increase the reliability of the site, both by restricting when certain changes can happen and by having a process for reviewing changes in advance to catch any adverse effects that the SA may have missed or interactions that the SA might not have known about. Change management also helps with debugging problems, because changes are tracked and can be reviewed when a problem arises.

The frequency with which you have change-management meetings depends on the changes' scope and how rapidly the environment that they cover changes. Instituting a mechanism through which SAs check that the site is operating normally before they make their changes reduces the risk that a change made while debugging an existing problem will complicate the debugging process or make the site even less stable.

In counterpoint, a DevOps environment implements automated change control through rigorous testing that is itself automated and built into the continuous integration, continuous delivery, and continuous deployment systems. We should all strive to attain the DevOps model and its admirable level of confidence in changes. However, until we get there, the change-management practices described in this chapter can help you reduce change-related incidents.

Exercises

1. Describe the change-management process in your organization.
2. How would you define the off-peak times for your site?
3. Look at the sorts of tasks you perform in your job, and categorize them as routine, sensitive, or major updates.
4. Which kind of communication process would work best for your company? Would you have both “push” and “pull” mechanisms? Why? What would you use each of them for?
5. How would you organize change-management meetings in your company? Who do you think should attend? How often would you have these meetings?
6. To what extent would change management affect the way that you do your job?

7. Consider how various sorts of updates should be categorized in the various areas of your site, and institute a scheduling practice that reflects your decision.
8. Which problems do the people who use your system have with the current setup?
9. Would it be better to make big changes on Friday or before a vacation, or to make them during business days?

Chapter 33. Server Upgrades

This chapter is focused on the specific task of upgrading the operating system of a single host. This task is deceptively simple, although it requires a lot of preparation beforehand and a lot of testing afterward. This technique is a building block, which can be applied to many similar situations. Later we'll move on to larger upgrade projects, such as those described in [Chapter 34](#), "[Maintenance Windows](#)."

The upgrade itself can be performed in many ways, but there are two main strategies for upgrades: Upgrade the host in-place or build a new machine and migrate clients to it. Depending on the services being provided by the machine and the facilities available in an organization, the SA needs to choose one of these approaches. There are some variations within these two approaches, but at its core an upgrade is either done in-place or by migration.

A single tool is required to do this task successfully, no matter which OS is involved. This tool is a piece of paper that will be used to maintain a checklist. There is no excuse for not using this tool. Our publisher, Addison-Wesley, has graciously agreed to include a blank piece of paper in the back of this book for your convenience. (If you desire additional blank paper, we highly recommend purchasing additional copies of this book.)

Some people choose to simulate a piece of paper by using a web page, a wiki, or a spreadsheet. These high-tech solutions have many benefits, which are described later. However, the fundamental issue is the same: There is no excuse for upgrading a server without using a checklist to guide you. Grab a pencil. Let's begin.

33.1 The Upgrade Process

The fundamental goal of any OS upgrade is that, at minimum, all the services provided *before* the upgrade will work *after* the upgrade. While an upgrade may be performed to *add* functionality or reliability, it should not reduce these attributes. With this in mind, the process is as follows:

1. Develop a service checklist:

- Which services are provided by the server?
- Who are the customers of each service?
- Which software package(s) provide which service?

2. Verify that each software package will work with the new OS, or plan a software upgrade path.
3. For each service, develop a test to verify that it is working.
4. Choose an upgrade strategy.
5. Write a detailed implementation plan.
6. Write a backout plan, with specific triggers.
7. Select a maintenance window.
8. Announce the upgrade as appropriate.
9. Execute the tests developed earlier to make sure that they are still valid.
10. Lock out customers.
11. Do the upgrade with someone watching/helping (mentoring).
12. Repeat all the tests developed earlier. Follow the usual debugging process.
13. If tests fail, or other events occur that trigger the backout plan, execute the backout plan.
14. Let customers back in.
15. Communicate completion/backout to the customers.
16. Analyze what went right and what didn't; modify the checklist to reflect the experience.

And you thought it was all about installing the new OS, didn't you? Let's look a little more closely at each step.

33.2 Step 1: Develop a Service Checklist

The **service checklist** is a tool that you use to drive the entire process. The list should record *which* services are provided by the host, *who* the customers of each service are, and *which* software package provides each service.

Ideally, this information should be kept in the configuration-management database (CMDB) or inventory system. A good, well-maintained inventory system will identify all the services on a machine, and customer representatives for each of them. This makes the creation of the checklist much easier. Mature sites use a CMDB to track hardware models and configurations, OS or firmware versions, the software installed on each machine, and the services that it provides. The CMDB also tracks the software components that are required to provide a particular service, the owner of the service, the owner of each software component, and dependencies on other components.

Using the CMDB to Drive Changes

When the inventory system is the tool that is used to drive initial OS installation, software installation, patching, and upgrades, the data is always up-to-date and the processes can all be automated. The task of upgrading a server involves producing and verifying the automation for each component, and for the combination of the various components. The server can then be scheduled for an automated upgrade. At the type of large sites that maintain inventory systems, this effort is worthwhile because many servers with these components will need to be upgraded, and the time investment pays off.

In the absence of an up-to-date inventory system, spreadsheets are an alternative way to maintain such information. The biggest benefit of maintaining this information in electronic form is that it can be easily shared both within the team and with customers. You may also be able to use it to bring the inventory system up-to-date. Making the data accessible via the web is better than mailing it to individuals, because the web version can be rapidly updated. People will always see the latest updates. Be sure that there is a prominent version number and date on the checklist, so that everyone can easily see if they have a current version.

Using a Change Log

In the absence of an up-to-date inventory system, building the service checklist is much easier if you've kept a log of what's been added to the machine. For example, on a Unix system, simply keep a record of changes in a file called `varadm/CHANGES`. The easier it is to edit the file, the more likely people are to update it, so consider creating a shell alias or short script that simply brings up that file in a text editor.

Double-check your plans by having a review meeting with key representatives of the affected community. Walk the team through the plan, step by step, asking for them to verify your assumptions. It is most effective to begin the process with a meeting and then use email for updates, possibly having additional face-to-face meetings only at key points in the process.

Including the customers as part of the decision-making and planning processes gives them a feeling of participation and control. Customers are invested in the outcome and become part of the team, which generally leads to a more positive

experience for them and a better relationship between the SA and business units. Sharing dependency and status information with customers on the web and via email helps to maintain the working relationship.

Customer Dependency Check

An SA once held a meeting attended by ten experienced SAs, all of whom looked at a plan and agreed right away that it looked fine to them. When the SA started stepping through it, asking such specific questions as “What’s going to happen when we turn this off?” they started saying, “Oops, no, if you do that, the billing system won’t work anymore. I guess we need to add a step where we move the billing information.” The result was a completely different plan with three times as many steps. If they hadn’t held an in-person meeting to carefully go through each step specifically, the original plan would have created a major disaster.

A machine may be dedicated to providing a single service, or it may provide many services. Either way, many software packages may be involved in providing the complete service.

Usually, each service is directly related to a single software package. Sometimes a service is related to multiple packages, such as a calendar server that relies on an LDAP server. Document all of these interdependencies in the checklist.

It is also important to determine the key customers relying on various services. These customers may be directly relying on a service, using the service itself, or they may be indirectly relying on the service, interacting with services that rely on another service for data or input. Any people affected should be included in the process or at least notified that the process is happening. If other machines are dependent on the services, users of those machines should be included.

Often, you will find a service with no direct or indirect customers, and the service can be eliminated. These are always happy moments, but be careful: You might find the dependency after the service no longer exists. It is better to disable services on the old machine ahead of the upgrade to verify that it is no longer required. Also consider having the service in a ready-to-run but dormant state on the upgraded machine so that it is easy to bring up, if necessary. Make sure that you document why the service is there but not running, so that it gets cleaned up next time through, if it has not been reenabled by then. The best place

for this documentation is in one of the configuration files that will be edited to reenable the service.

33.3 Step 2: Verify Software Compatibility

The next step is to verify that each software package will work with the new OS and to plan an upgrade path for those that don't. Using the list developed earlier, contact the vendor and find out whether the software release in use will work after the upgrade. Vendors often list such information on their web sites.

You may wish to test the reliability of the information yourself or find another customer who has already performed the upgrade. Vendors' ideas about what it means for a version to work may not include the features your site needs or the exact configuration you're going to use. Doing the tests yourself can be expensive but possibly cheaper than a failed upgrade, and reduces the risk of failure. The point of this is risk management. If only one system is being upgraded and if the application is not critical, personally testing it might be a waste of time. If the upgrade will be repeated thousands of times, in an automated fashion and in a way that failure would be highly visible, testing is a requirement.

If the software release being used will work on the new OS release, document where you found this information for future reference. If the software isn't supported on the new OS, you have several options:

- Upgrade the software before the OS.
- Upgrade the software after the OS.
- Postpone the upgrade or change the software.

33.3.1 Upgrade the Software Before the OS

If there is a software release that works with the old and new OS, we can upgrade the software ahead of time. This approach minimizes how long the software will be unavailable. When the machine returns to service with the new OS, the software will be available. There will be an additional bit of downtime when the software is upgraded, but we would rather work in two small batches than one big batch.

Decoupling the software upgrade from the OS upgrade means one fewer thing to worry about during the heat of the moment when performing the OS upgrade. Also, the tests developed to make sure the application worked after the software upgrade can be repurposed as part of the testing done after the OS upgrades. If the test fails, you know the problem is related to the OS upgrade instead of

wondering which change caused the test to fail.

33.3.2 Upgrade the Software After the OS

If there is no software release that works with both the old and new OS, then the software must be upgraded after the OS. If the software is critical, perform the upgrade immediately after the OS is upgraded. This is convenient to the users because they see only one downtime period.

Alternatively, if the customers can be without the software for a period of time, you have the option of performing the software upgrade later. This is a matter of negotiation with the customers. For example, if a host is a busy web server, the customers may request that the new web server software be installed immediately because it is a major function of the host.

In contrast, if a little-used compiler requires an upgrade, the customers may simply request that it be upgraded in the next week or before a certain development cycle is complete. This is especially true if some other host can be used for compilation in the meantime.

33.3.3 Postpone the Upgrade or Change the Software

Sometimes we receive bad news from the software provider. The software just does not work with the new OS and will not be supported. Perhaps the vendor is out of business or no longer supports the product. In this case the OS upgrade is either blocked until a new software product can be identified or the software is abandoned. Again these decisions must be negotiated with the customer.

33.4 Step 3: Develop Verification Tests

As each service is identified, a test should be developed that will be used to verify that the service is working properly after the upgrade. The best scenario is to have all the tests recorded as scripts that can be run unattended. A master script can be written that outputs an OK or a FAIL message for each test. Tests can then be run individually as specific problems are debugged. For more complicated services, customers may write the tests or at least review them, or offer to be on call to execute their own set of manual tests. Some software packages have an installation test suite that can be run for verification. Sometimes these test suites are not available to customers, but may be acquired through a vendor representative.

All software packages should have such verification procedures, but they rarely do. Sometimes, it is best to write your own. The tests can be simple, such as testing a compiler by compiling a `Hello, World` program. One test is infinitely better than no testing at all.

Sometimes, a verification procedure is provided, but doesn't actually work. One supercomputer vendor was notorious for having bad verify databases, especially in the beta OS releases.

The software world uses the term **regression testing** to describe a particular way of doing verification. You capture the output of the old system, make a change, and then capture the output of the new system. The output should match exactly. If the new output is expected to be slightly different, you might edit the baseline output by hand to reflect expected changes, or use a *fuzzy match* algorithm. Simple tools can be used to compare the outputs. For example, the Unix `diff` is an extremely useful program that compares two text files and points out the differences between them. The `diff` tool has a limited fuzzy-match capability; the `-w` option makes all whitespace the same. More sophisticated regression-testing software can be programmed to ignore certain specific changes, usually based on a system of regular expressions. However, such complexity is not required. You can manually change the old output (make a backup first!) to reflect the differences that are expected in the new output. For example, you might change the version numbers in the output to match the new software. Excellent examples of regression testing are included in Kernighan & Pike (1999) as well as the installation procedure for `perl`. (Look at how `make tests` is implemented.)

Sometimes, the tests can be as simple as a `Hello, world!` program that is compiled and run to verify that a compiler works. Or perhaps the test consists of a particular sequence of commands or mouse clicks that is executed to see whether an expected result is displayed. However, be careful to make sure that the tests are not superficial.

Hello, World!

Tom was once responsible for maintaining a wide large range of compilers for many operating systems. He maintained a library of simple programs; most simply printed `Hello, world!` and then exited. He could always verify that a new compiler installation was at least fundamentally correct if the appropriate program(s) compiled and ran.

When new languages were added to the mix, he would often recruit the programmers to write the test programs. The programmers enjoyed being asked to help out!

You must subject the tests to the same level of scrutiny as any other service. When the tests are put into use, you don't want there to be any doubt whether a test failed because of the upgrade or because the test itself was flawed.

It is tempting to perform these tests manually. However, remember that each test will be done a minimum of three times, and more times if there are problems. There are benefits to automating the tests. If they are general enough, they can be reused during future upgrades. Ultimately, they can be reused on a regular basis simply to debug problems, or as monitoring tools to notice outages before your customers do.

Scripted tests work fine for programs that produce predictable text output, but they're much more difficult to develop for graphical programs, for network services, and for such physical issues as printing. For a network service such as NFS, you can try to access a file rather than test the protocol itself. Testing of network services that have simple text-based protocols, such as email (SMTP, POP, IMAP) or web HTTP services, can be automated with simple scripts using a tool such as `netcat` to send and receive protocol text on the appropriate network port.

For other programs and services, you can find specialized test systems, but they are usually extremely expensive. In these cases, you will simply end up testing by hand, documenting a few key features to test or a sequence of operations to perform. Everyone faces this situation at one time or another, and we should all complain to vendors until they instrument their products so that such testing can be automated.

Test-Driven Development (TDD)

TDD puts testing first, literally. Previously, developers wrote code and then wrote tests to verify the code. (Well, not really—rarely did anyone have time to write the tests.) TDD works in reverse: The tests are written first and then the code. This ensures that the tests get written for all new code. Since the tests are executed in an automated fashion, you build up a body of tests that stay with the project. As the code evolves, there is less risk that a change will break functionality without being noticed.

Developers are free to rewrite, or *refactor*, big or small parts of the code,

knowing that if they break something, it will be noticed right away. In turn, the resulting software has fewer bugs.

Tests are better than comments in code (documentation) because comments often become outdated without anyone noticing. Tests that cover all the edge cases are more thorough than documentation could ever be. Tests do not become obsolete, because they can be triggered as part of the build process to alert developers of bugs when they are introduced into the code.

It would be great to see the field of system administration learn from TDD and adopt these practices.

Although it is preferable to automate all tests, it is not always possible. Some tests are too difficult to automate or require physical observation. Even if you have automated all tests, if you get a hunch that an additional manual test might be useful, go for it. Sometimes the human eyeball catches things that the best automation can't.

Keeping Tests Around for Later

A large company wanted to test 400 Unix servers just after midnight of Y2K to ensure that the core functionality of the operating system and associated infrastructure were working correctly. A series of noninvasive tests was created, each with a PASS/FAIL response: Is the box up, can we log in, can it see the NIS servers, is the time correct, can it resolve DNS, can it mount from the NFS servers and read a file, is the automounter working, and so on. Using a central administration point, the tests could be fired off on multiple boxes at a time and the results collected centrally. All 400 boxes were tested within 20 minutes, and the team was able to report their PASS to the Y2K tracking-management team well in advance of other, smaller, units.

The tests found other uses. The site found that an obscure bug in some automounters could be triggered on a few random machines after a major network outage. By running this test suite, the affected machines could be quickly identified after any outage. So popular did the tests become with the SA team that they became part of the daily monitoring of the environment.

33.5 Step 4: Choose an Upgrade Strategy

There are two main strategies for performing server upgrades. Other approaches boil down to some combination of these approaches. Which one is appropriate for a particular upgrade depends on the circumstances.

The first strategy is an in-place upgrade. An in-place upgrade typically involves backing up the host, then upgrading the OS and software. The backout plan does a complete restore of the old host OS and software. Variations on this approach include cloning the machine, upgrading the clone, and then swapping the machines—backing out involves swapping the machines back.

The second strategy is to migrate services and customers to a new machine with the new OS and software. For each service or customer migration, the backout plan reverts to using the original machine. The migration of a service may happen all at once by stopping the service, copying the data to the new machine, starting the service on the new machine, and pointing the DNS alias there. Alternatively, the migration may be done gradually by migrating groups of users on different dates. A gradual migration may be performed by applying the “one, some, many” technique, described in [Section 7.5](#), or by grouping users by department, region, or risk tolerance.

A hybrid approach is to build a new machine with the new OS and software, test it, and then migrate by switching off the old machine and bringing the new machine up with the name and IP address of the old machine. Backing out the change involves switching off the new machine, and switching on the old one. This approach acknowledges that sometimes it is much better to do a fresh install than an upgrade. Doing upgrade after upgrade can lead to a system with a lot of damage. It can result in files left over from old patches, fragmented file systems, and a lot of “history” from years of entropy.

When upgrading a group of servers that all provide the same service, each machine will be upgraded using one of these techniques. In addition, the SAs can decide whether it is best to upgrade all servers in the group simultaneously, one at time, or in groups. The groups may be defined based on a one, some, many technique, or based on the end users if different servers serve different groups or regions.

When deciding which approach to take, the SA needs to consider the implications of each approach. The main differences between the approaches described here are in the areas of speed, risk, end-user disruption, and effort.

33.5.1 Speed

An in-place upgrade, or a hybrid that involves swapping machines, gives a fast upgrade of all users. A fast upgrade can be useful when resolving a serious problem—for example one that is impacting production, or a serious security vulnerability. It is also appropriate for services that cannot easily be split across several machines: If all users must be migrated at once, the faster you can do it, the better.

A migration of users from one server to another is inevitably a slower migration. This technique enables the SAs to do a proof of concept, and a pilot of the new service with a small user group, and work out any bugs before rolling the service out to production. If training is required for the new service, it also allows the SAs to migrate each user group right after they have received training. The users can then make the best use of their training—when the time between the training and the actual use of the product is too long, the training is likely to be forgotten. Finally, this approach allows the SAs to learn what needs to be improved in the training materials and to incorporate those changes for the next group.

33.5.2 Risk

A fast in-place upgrade is usually riskier than a slow migration upgrade because any problems will impact more end users. A slow migration impacts fewer people and allows bugs to be worked out with the early adopters. Organizing a slow migration so that the less risk-averse groups are migrated first, and the most critical users are migrated last, also reduces risk.

In some circumstances there may be so many dependencies on a particular machine or service that trying to pull everything apart to do a slow migration is riskier. These situations are quite rare, however.

Risk can be mitigated by thorough testing. For in-place migrations, testing is especially critical, as the risk is greater.

33.5.3 End-User Disruption

The normal goal in system administration is to cause as little disruption to the end users as possible. If the upgrade requires a change in the working practices of the end users, such as using a new client application, then it is inherently disruptive. In this case, try to minimize the disruption to the end users by concentrating all the disruptive tasks, such as migrating them to the new server, into a single disruption.

For upgrades that do not require a client change, doing a slow migration may

cause a disruption when migrating end users to the new system, whereas an in-place migration would not. This is another factor that should be taken into account when choosing a migration strategy.

When performing such a migration, look for ways to reduce, or eliminate, the disruption for future upgrades. For example, if the migration is being performed by region, and the disruption involves changing the name of the server that the client software uses, change the client to use an alias for the new server that is specific to that region. The slow migration, moving a group at a time to the new service, can be implemented by changing the region-specific aliases on different dates, with no disruption to the end users.

33.5.4 Effort

An upgrade that does not require end-user retraining requires less time investment from the SAs than an upgrade that does necessitate such retraining. Also, an in-place upgrade requires less time investment than an upgrade that involves a migration.

A good example of an upgrade that requires much less effort in-place than by migration is an upgrade of a system that runs services that are accessed by IP address, rather than by name. Two such services are DNS and DHCP. In the case of DNS, it should be possible to move the majority of clients by changing the DHCP settings. Of course, there are always some systems with statically configured DNS servers, which need to be painstakingly tracked down and changed. In a large organization, this can be a very time-consuming process. In the case of DHCP, all of the routers that act as DHCP relays need to be changed, which also involves considerable effort. Upgrading in-place, or re-addressing so that the new server is given the old server's IP addresses, eliminates the need for this additional effort.

The effort required of the SAs in each method is another factor that should be considered. SAs' time is a limited resource. A slow migration involving a lot of work for the SAs may delay other critical projects.

33.6 Step 5: Write a Detailed Implementation Plan

Write a detailed, step-by-step implementation plan that anyone could follow. While this process may seem like a lot of work, it can help you find potential problems in advance and resolve them ahead of the upgrade. It also makes subsequent upgrades easier, and opens the way to automating the process or delegating it to others.

The implementation plan should include not only the OS upgrade, but also what, if anything, needs to be done for each of the software packages, and all the testing scripts, along with time estimates for each step. For each test, write a list of things to check if the test fails, and all the steps you can think of to debug what caused the failure. This list can prove invaluable during the change when tests fail and you are too tired or stressed to think straight. You can keep adding to it as you think of other possibilities in the run-up to the upgrade.

33.6.1 Adding Services During the Upgrade

You must sometimes add or remove services during an upgrade. This complicates matters because more than one change is being made at a time. Debugging a system with two changes is much more difficult because it affects the tests that are being executed. Adding services has all the same problems as bringing up a new service on a new host, but you are now in a new and possibly unstable environment and it is difficult to prepare for this event by creating appropriate tests. However, if the new service is also available on a different host, tests can be developed and run against that host.

33.6.2 Removing Services During the Upgrade

Removing a service can be both easy and difficult at the same time. It can be easy for the same reason that it is easier to tear down a building than to build one. However, you must make sure that all the residents are out of the building first. Sometimes we set up a network sniffer to watch for packets indicating that someone is trying to receive that service from the host. That information can be useful to find stragglers.

It's best to disable a service in a way that makes it easy to reenable quickly if forgotten dependencies are discovered later. For example, the service can be halted without removing the software. It is usually safe to assume that if no forgotten dependencies are discovered in the next month or year, it is safe to remove the software. Note that some services may be used only once a quarter or once a year, especially certain financial reports.

Don't forget to come back to clean up! Create a ticket in your helpdesk system, send yourself email, or create an `at` job that emails you a reminder sometime in the future. If multiple SA groups or privileged customers have access to the box, it can be a good idea to add a comment to the configuration file, or rename it to include "OFF" or "DISABLED." Otherwise, another SA might assume the service is supposed to be up and turn it back on.

33.6.3 Old and New Versions on the Same Machine

Sometimes, you might be upgrading just a single service on a machine, not the entire OS. In that situation, it is helpful if the vendor permits the old versions of the software to remain on the machine in a dormant state while the new software is installed and certified.

The web server Apache on Unix is one such product. We usually install it in *optapache**x.y.z*, where *x.y.z* is the version number, but place a symbolic link from *optapache* to the release we want to use in the future. All configurations and scripts refer to *optapache* exclusively. When the new version is loaded, the *optapache* link is changed to point to the new version. If we find problems with the new release, we revert the symbolic link and restart the daemon. It is a very simple backout plan.

In some situations, the old and new software can run simultaneously. If a lot of debugging is required, we can run the new version of Apache on a different port while retaining the old version.

33.6.4 Performing a Dress Rehearsal

Take a lesson from the theater world: Practice makes perfect. Why not perform a dress rehearsal on a different machine before you perform the upgrade? Doing so might reveal unexpected roadblocks, as well as give you an indication of how long the process will take. It results in a much better implementation plan. A dress rehearsal requires a lot of resources. However, if you are about to perform the first upgrade of many, this can be a valuable tool to estimate the amount of time the upgrades will require. An absolutely complete dress rehearsal results in a new machine that can simply replace the old machine. If you have those resources, why not do just that?

Theater productions also hold what's referred to as the *tech rehearsal*—a rehearsal for the lighting and sound people more than for the actors. The actors run through their lines with the right blocking as the lighting and sound directions are put through their paces. The SA equivalent is to have all the involved parties walk through the tasks.

We also borrow from theater the fine art of pantomime. Sometimes a major system change involves a lot of physical cables to be changed. Why not walk through all the steps, looking for such problem areas as cable lengths, crossover/straight-through mismatches, male/female connector mismatches, incorrect connectors, and conflicting plans? Pantomime the change exactly how it will be done. It can be helpful to have someone else with you and explain the

tasks as you act them out. Verify to the other person that each connector is correct, and so on. It may seem silly and embarrassing at first, but the problems you prevent will be worth it.

33.7 Step 6: Write a Backout Plan

If something goes wrong during the upgrade, how will you revert to the former state? How will you “undo”? How long will that take? Obviously, if something small goes wrong, you should try to debug it and fix it within your allotted time. However, you can use up the entire maintenance window—the time allocated for the outage—trying just one more thing to make an upgrade work. It is therefore important to have a particular time at which the backout plan will be activated. Take the agreed-on end time and subtract the backout time, as well as the time it would take to test that the backout is complete. When you reach that time, you must either declare success or begin your backout plan. It is useful to have the clock watcher be someone outside the group directly performing the upgrade, such as a manager. The backout plan might also be triggered by one or more key tests failing, or by unexpected behavior related to the upgrade.

Test the backout plan to confirm that it works, and to make sure that you know how long implementing it will take. Make sure that every step is documented in detail.

33.8 Step 7: Select a Maintenance Window

The next step is a test of your technical and nontechnical skills. You must come to agreement with your customers on a maintenance window—that is, when the upgrade will happen. To do that, you must know how long the process will take and have a plan if the upgrade fails. That is more of a technical issue.

Scotty Always Exaggerated

In the *Star Trek: The Next Generation* episode “Relics,” James Doohan made a cameo appearance as Scotty from the original series. Among Scotty’s interesting revelations was that he always exaggerated when giving estimates to Captain James T. Kirk. Thus, he always looked like a miracle worker when problems were solved more quickly than expected. Now we know why the warp drive was always working sooner than predicted and the environmental systems lasted longer than were indicated. Follow Scotty’s advice! Exaggerate your estimates! But also follow Scotty’s practice of letting people know as soon as the work is tested and complete.

The following characteristics should always be communicated as part of a maintenance window:

- **When?** Your SLA should include provisions for when maintenance can be done. Customers usually have a good idea of when they can withstand an outage. Most business systems are not needed at night or on the weekend. However, SAs might not want to work those hours, and the vendor support might not be available at certain times. A balance must be found. Sites that are required to be up 24/7 have a maintenance plan engineered into the entire operation, perhaps including fall-back systems.
- **How long?** The length of the maintenance window equals the time the upgrade should take, plus the time testing should take, plus the time it will take to fix problems, plus the time it takes to execute the backout plan, plus the time it takes to ensure that the backout worked. Initially, it is best to double or triple your estimates to adjust for hubris. As time goes on, your estimates will become more accurate.

Whatever length of time you have calculated, announce the window to be much longer. Sometimes, you may get started late. Sometimes, things take longer than you expect for technical reasons (hardware, software, or unrelated or unexpected events) or nontechnical reasons (weather or car problems). The flip side to calling for a longer time window is that if you complete the upgrade and testing early, you should always notify the customers.

- **What time?** It is a good idea to clearly document the exact time that the backout plan will be initiated for the reasons described in step 5.

Case Study: The Monday Night Carte Blanche

When Tom worked at a division of Mentor Graphics, the SA staff had the luxury of a weekly maintenance window. Monday night was SA “Carte Blanche Night.” Users were expected to be logged out at 6 PM, and the SA staff could use that evening to perform any kind of major upgrades that would require bringing down services. Every Monday by 4 PM, the customers were informed of which changes would be happening and when the systems should be usable again. Customers eventually developed a habit of planning non-work-related activities on Monday nights. Rumor has it that some spent the time with their families.

Although it required a big political investment to get the practice

approved through management, the Monday night maintenance window was an important factor in creating high reliability in the division's network. There was rarely a reason to put off timely system upgrades. Problems during the week could be taken care of with quick fixes, but long-term fixes were done efficiently on Monday night. Unlike in some company environments in which the long-term fixes are never implemented, at this company long-term fixes were always put in relatively soon.

When there wasn't much to be done, one supervisor believed it was important to reboot some critical servers at 6 PM to "encourage" users to go home for the night. He believed that this helped the users maintain their habit of not planning anything critical for Monday night. Of course, the SAs were flexible. When the customers were up against a critical deadline and would be working around the clock, the SAs would cancel the Monday night maintenance or collaborate with the customers to determine which outages could happen without interfering with their work.

33.9 Step 8: Announce the Upgrade

Now announce the upgrade to the customers. Use the same format for all announcements so that customers get used to them. Depending on the culture of your environment, the message may best be distributed by email, voicemail, desk-to-desk paper memo, newsgroup posting, web page, note on door, or smoke signals. No matter which format is used, the message should be brief and to the point. Many people read only the Subject line, so make it a good one, as shown in [Figure 33.1](#).

To: all-users
Subject: SERVER REBOOT: 6 PM TODAY
From: System Administration Group <help@example.com>
Reply-To: tom@example.com
Date: Thu, 16 Jun 2001 10:32:13 -0500

WHO IS AFFECTED:

All hosts on DEVELOPER-NET, TOWNVILLE-NET, and BROCCOLI-NET.

WHAT WILL HAPPEN:

All servers will be rebooted.

WHEN?

Today between 6-8 PM (should take 1 hour)

WHY?

We are in the process of rolling out new kernel tuning parameters to all servers. This requires a reboot. The risk is minimal. For more information please visit:
<http://portal.example.com/sa/news0005>

I OBJECT!

Send mail to "help" and we will try to reschedule. Please name the server you want us to keep up today.

Figure 33.1: Sample upgrade message

It is better to have a blank template that is filled out each time than to edit previous announcements to include new information. This prevents the form from mutating over time. It also prevents the common problem of forgetting to change some parts. For example, when creating [Figure 33.1](#), we initially used a real announcement that referred to a router reboot. We changed it to be about

servers instead but forgot to change the Subject line. The example went through four rounds of proofreading before anyone noticed this. This wouldn't have happened if we had started with a blank template instead.

33.10 Step 9: Execute the Tests

Right before the upgrade begins, perform the tests. This last-minute check ensures that after the upgrade you won't be chasing problems that existed before the upgrade. Imagine the horror of executing the backout plan, only to discover that the failing test is still failing.

If the tests are properly scripted, they can be integrated into a real-time monitoring system. In fact, if your monitoring system is already doing all the right tests, you shouldn't need anything else during your upgrade. (See [Chapter 38](#), “[Service Monitoring](#),” for more discussion about service monitoring.)

Repository for Tests

It is rare that all tests can be automated and added to the monitoring system. For example, load testing—determining how the system performs under simulated amounts of work—often cannot be done on a live system. However, all tests should be available for anyone to check out and use. Store them in a well-known source code repository. Being able to run these tests during otherwise low-usage hours or on demand when debugging a problem can make it easy to track down problems.

33.11 Step 10: Lock Out Customers

It is generally better to let customers log out gracefully than to kick them out by a reboot or disconnection of service. Different services have different ways to do this. Use the facilities available in the OS to prevent new logins from occurring during the maintenance window. Many customers use an attempt to log in or to access a resource as their own test of an upgrade. If the attempt succeeds, the customer believes that the system is available for normal use, even if no announcement has been made. Thus it is important to lock out customers during a maintenance window.

Ignoring Big Red Signs

Customers tend to ignore messages from SAs. Josh Simon reports that at one client site, he tried leaving notes—black text on bright red paper taped to the monitors—saying “DO NOT LOG IN—CONTACT YOUR

SYSTEM ADMINISTRATOR AT [phone number] FIRST!” in huge type. More than 75 percent of the customers ripped the paper off and proceeded to log in rather than call the phone number. The lesson to be learned here is that it is often better to actually disable a service than to ask customers not to use it.

33.12 Step 11: Do the Upgrade with Someone

This is where most SA books begin. Aren't you glad you bought this book instead?

Now, the moment you've been waiting for: Perform the upgrade as your local procedures dictate. Insert the DVD, reboot, whatever. However, system upgrades are too critical to do alone, so have someone watch to make sure you do it correctly, for the following reasons.

First, we all make mistakes, and a second set of eyes is always useful. Upgrades aren't done every day, so everyone is always a little out of practice. Second, a unique kind of mentoring goes on when two people do a system upgrade together. System upgrades often involve extremes of our technical knowledge. We use commands, knowledge, and possibly parts of our brains that aren't used at other times. You can learn a lot by watching and understanding the techniques that someone else uses at these times. The increasingly popular practice of co-development or so-called peer programming has developers working in pairs and taking turns being the one typing. This is another development practice that SAs can benefit from using.

If the upgrade isn't going well, it is rarely too early to escalate to a colleague or senior member of your team. A second set of eyes often does wonders, and no one should feel ashamed about asking for help.

33.13 Step 12: Test Your Work

Now repeat all the tests developed earlier. Follow the usual debugging process if they fail. The tests can be repeated time and time again as the problem is debugged. It is natural to run a failing test over again each time a fix is attempted. However, since many server processes are interrelated, be sure to run the full suite before declaring the upgrade a success. The fix for the test that failed may have broken a previously successful test!

Customers should be involved here. As with the helpdesk model in [Chapter 28](#), “[Handling an Incident Report](#),” the job isn't done until customers have verified that everything is complete. This may mean calling the customer at a

prearranged time, or the customer agreeing to report back the next day, after the maintenance window has elapsed. In that case, getting the automated tests right is even more critical.

33.14 Step 13: If All Else Fails, Back Out

If the clock watcher announces that it is time to begin the backout plan, you have to begin the backout plan. This may happen if the upgrade is taking longer than expected or if it is complete but the tests continue to fail. The decision is driven entirely by the clock—it is not about you or the team. It can be disappointing and frustrating to back out of a complex upgrade, but maintaining the integrity of the server is the priority.

Reverting the system back to its previous state should not be the only component of the backout plan. Customers might agree that if only certain tests fail, they may be able to survive without that service for a day or two while it is repaired. Decide in advance the action plan for each potential failure.

After the backout plan is executed, test the services again. At this point it is important to record in your checklist the results of your changes. This documentation is useful in reporting status back to management, record-keeping for improving the process next time, or recalling what happened during a postmortem. Record specifics such as “implemented according to plan,” “implemented but exceeded change window,” “partial implementation; more work to be done,” “failed; change backed out,” or “failed; service unusable; end of world predicted.” If possible, capture the output of the test suite and archive it along with the status information. This will help immensely in trying to remember what happened when you try again next week, month, or year.

33.15 Step 14: Restore Access to Customers

Now it is safe to let customers start using the system again. Different services have different ways to permit this. However, it is often difficult to do testing without letting all users in.

There are some ways to do this, however. For example, when upgrading an email server, you can configure other email servers to not relay email to the server being upgraded. While those servers are holding email, you can manually test the upgraded server and then reenable the surrounding servers one at a time, keeping a mindful eye on the newly upgraded server.

33.16 Step 15: Communicate Completion/Backout

At this point, the customers are notified that the upgrade is complete or, if the backout plan was initiated, what was accomplished, what didn't get accomplished, and the fact that the systems are usable again. This communication has three goals. First, it tells people that the services to which they have been denied access are now usable. Second, it reminds the customers what has changed. Finally, if they find problems that were not discovered during your own testing, it lets them know how to report problems they have found. If the backout plan was initiated, customers should be informed that the system should be operating as it had before the upgrade attempt.

Just as there are many ways to announce the maintenance window, so there are many ways to communicate the completion. Of course, there is a Catch-22 here. Customers cannot read an email announcement if the email service is affected by the outage. However, if you keep to your maintenance window, then email, for example, will be working and customers can read the email announcement. If customers hear nothing, they will assume that at the end of the announced maintenance window, everything is complete.

Announcements should be short. Simply list which systems or services are functioning again, and provide a URL that people can refer to for more information and a phone number to call if a failed return to service might prevent them from sending email. One or two sentences should be fine.

The easiest way to keep the message short is to forward the original email that said the services were going down, and add a sentence to the top, saying that services are reenabled and explaining how to report problems. This gives people the context for what is being announced in a very efficient way.

Case Study: Upgrading a Critical DNS Server

This case study combines many of the techniques discussed in this chapter. During the rush to fix Y2K bugs before January 1, 2000, Tom found a critical DNS server that was running on non-Y2K-compliant hardware, which the vendor had announced would not be fixed. Also, the OS was not Y2K compliant. This was an excellent opportunity to perform a fresh load of the OS on entirely new hardware.

Tom developed a service checklist. Although he thought that the host provided only two services, by using `netstat -a` and listing all the running processes he found many other services running on the machine. He discovered that some of those extra services were no longer in use and found one service that no one could identify!

People knew that most of the software packages involved would work on the new OS because they were already in use on other machines with the newer OS. However, many of the services were home-grown, and panic erupted when it was thought that the author of a home-grown package was no longer at the company and the source code couldn't be found immediately. Luckily, the code was found.

Tom built the new machine and replicated all the services onto it. The original host had many configuration files that were edited on a regular basis. He needed to copy these data files to the new system to verify that the scripts that processed them worked properly on the new machine. However, because the upgrade was going to take a couple of weeks, those files would be modified many times before the new host would be ready. Once the tests worked on the old data, the files were recopied daily to make sure the tests still worked. When the new system was cut in, Tom stopped all changes on the old host, recopied the files to the new system one last time, and verified that the new system accepted the new files.

The tests that were developed were not run just once before the cutover, but rather were run over and over as various services on the new system became usable. However, Tom did leave most services disabled when they weren't being tested because of concern that the old and new machines might conflict with each other.

The cutover worked as follows: The old machine was disconnected from the network but left running. The new machine's IP address was changed to that of the old one. After five minutes, the ARP caches on the local network timed out, and the new host was recognized. If problems appeared, Tom could unplug the new machine from the network and reconnect the network cable of the legacy machine. The legacy machine was left running, so that not even a reboot would be required to bring it back into service: Just halt the new server and plug in the old server's network cable.

The actual maintenance window could have been quite short—a minimum of 5 minutes if everything went right and the machine could be reconnected instantly. However, a 30-minute window was announced.

Tom decided to have two people looking over his shoulder during the upgrade, because he wasn't as familiar with this version of Unix as he was with others and didn't get much sleep the night before. It turned out that having an extra pair of hands helped with unplugging and plugging wires.

The group pantomimed the upgrade hours before the maintenance window. Without changing anything, they walked through exactly what was planned. They made sure that every cable would be long enough and that all the connectors were the right type. This process cleared up any confusion that anyone on the team might have had.

The upgrade went well. Some tests failed, but the group was soon able to fix the problems. One unexpected problem resulted in certain database updates not happening until a script could be fixed. The customers who depended on that data being updated were willing to live with slightly stale data until the script could be rewritten the next day.

33.17 Summary

We have described a fairly complete process for upgrading the OS of a computer, yet we have not mentioned a particular vendor's OS, particular commands to type, or buttons to click. The important parts of this process are not the technology specifics, which can be obtained by reading manuals, but rather communication, attention to detail, and testing.

The basic tool we use in this process is a checklist. We begin by developing the checklist, which we then use to determine which services require upgrading, how long the upgrade will take, and when we can do it. The checklist drives which tests we develop, and those tests are used over and over again. We use the tests before and after the upgrade to ensure quality. If the upgrade fails, we activate the backout plans included in the checklist. When the process is complete, we announce this accomplishment to the list of concerned customers on the checklist.

A checklist is a simple tool. It is a single place where all the information is maintained. Whether you use paper, a spreadsheet, or a web page, the checklist is the focal point. It keeps the team on the same page (figuratively speaking), keeps the individuals focused, lets the customers understand the process, helps management understand the status, and brings new team members up to speed quickly.

Like many SA processes, upgrading an OS requires communication skills. Negotiation is a communication process, and we use it to determine when the upgrade will happen, what needs to happen, and what the priorities are if things go wrong. We give the customers a feeling of closure by communicating to them when we are finished. This helps the customer/SA relationship. We cannot stress enough the importance of putting the checklist on a web page. The more eyes

that can review the information, the better.

When the tests are automated, we can repeat them with accuracy and ensure completeness. These tests should be general enough that they can be reused not only for future upgrades on the same host but also on other similar hosts. In fact, the tests should be integrated into your real-time monitoring system. Why perform these tests solely after upgrades?

This simple process can be easily understood and practiced. It is one of the basic processes that an SA must master before moving on to more complicated upgrades. The real-world examples we used all required some kind of deviation from the basic process, yet still encompassed the essential points.

Some OS distributions make upgrading almost risk-free and painless, whereas others are much riskier. Although there are no guarantees, it is much better when an operating system has a way to do upgrades reliably, repeatably, and with the ability to easily revert. Having a minimal number of commands or mouse clicks reduces the possibility of human error. Being able to upgrade many machines in a repeatable way has many benefits; especially important is the fact that it helps maintain consistent systems. Any ability to revert to a previous state gives a level of undo that is like an insurance policy: You hope you never need it, but are glad it exists when you do.

Exercises

1. Select a server in your environment and figure out which services it provides. If you maintain a documented list of services, which system commands would you use to cross-check the list? If you do not have the services documented, what are all the resources you might use to build a complete list?
2. In your environment, how do you know who depends on which services?
3. Select a location that should be easy to walk to from your machine room or office, such as a nearby store, bank, or someplace at the other end of your building. Have three or four fellow students, co-workers, or friends estimate how long it will take to walk there and back. Now, all of you should walk there and back as a group, recording how long it takes. Calculate how close each of you was to being accurate, the average of these differences, and the standard deviation. Relate what you learned to the process of planning a maintenance window. Some points to consider: How long did it take? Did you start walking right away, or were you delayed? How many unexpected events along the way—running into

customers, people who wanted to know what you were doing, and so on—extended your trip's time? What did you learn from this exercise? If you repeat it, how much better do you think your estimate will be if you select the same location? A different location? Would bringing more people have affected the time?

4. In [Section 33.4](#), the claim is made that the tests that are developed will be executed at least three times, and even more times if there are problems. What are the three minimum times? What are some additional times when the tests may be run?
5. [Section 33.16](#) includes a case study in which the source code for a home-grown service almost couldn't be found. What would you do in that situation—if the source code code you were looking for couldn't be found?
6. How do you announce planned outages and maintenance windows in your environment? What are the benefits and problems with this method? What percentage of your customers ignore these announcements?
7. Customers often ignore announcements from SAs. What can be done to improve this situation?
8. Select a host in your environment and upgrade it. (Ask permission first!)
9. Which steps would you take if you had to replace the only restroom in your building? This is obviously a critical service that cannot have downtime, so take that factor into account.

Chapter 34. Maintenance Windows

If you had to power off an entire datacenter, do a lot of maintenance, and then bring it all back up, would you know how to manage the event? Computer networks and datacenters sometimes need massive, disruptive maintenance. Cooling systems must be powered off, drained, cleaned, and refilled. Large volumes of data must be moved between file servers to optimize performance for users or simply to provide room for growth. Also, improvements that involve many changes sometimes can be done much more efficiently if all users agree to a large window of downtime.

Sometimes a company allocates a specific time slot for major and risky changes to consolidate downtime to a specific time when customers will be least affected. Other times we are forced to schedule a significant maintenance window because of physical maintenance such as construction, power or cooling upgrades, or office moves. Sometimes we need to do this for emergency reasons, such as a failing cooling system.

Some companies are unable to have a large outage for business reasons. E-commerce sites and ISPs fall into this category. Other companies, such as banks, may have some core customer-facing systems that must remain up, and some less critical internal systems that can have disruptive maintenance scheduled. These companies need to provide high availability to their customers, who typically are off-site and not easily contacted. These companies do, however, still need maintenance windows.

This chapter describes a technique for managing major planned maintenance. Along the way, we provide tips useful in less dramatic settings. Projects like this require more planning, more orderly execution, and considerably more testing. We call this the **flight director** technique, named after the role of the flight director in NASA space launches. The originator of this chapter's techniques and terminology was Paul Evans, an avid observer of the space program.

The techniques presented in this chapter are useful not just for periodic maintenance windows, but also for single, major, planned systems work, such as moving the company to a new building. The end of this chapter looks at how the principles learned in this chapter apply in a high-availability site.

34.1 Process Overview

A **maintenance window** is by definition a short period in which a lot of systems

work must be performed. It is often disruptive to the rest of the company, so the scheduling must be done in cooperation with the customers. Some companies are willing to schedule regular maintenance windows for major systems and networking devices in return for better availability during normal operations. Depending on the size of the site, this could be one evening and night per month or perhaps from Friday evening to Monday morning once a quarter. These maintenance windows are necessarily very intense, so consider the capacity and well-being of the system administration staff, as well as the impact on the company, when scheduling them.

During the maintenance window, a group of SAs must perform various tasks, some of which will conflict with each other, and that work must be coordinated by the flight director. The flight director technique guides the activities before the window, during execution, and after execution (see [Table 34.1](#)).

Stage	Activity
Preparation	• Schedule the window. • Pick a flight director. • Prepare change proposals. • Build a master plan.
Execution	• Disable access. • Determine shutdown sequence. • Execute plan. • Perform testing.
Resolution	• Announce completion. • Enable access. • Have a visible presence. • Be prepared for problems.

Table 34.1: Three Stages of a Maintenance Window

In this chapter, we discuss the role and activities of the flight director and the mechanics of running a maintenance window as it relates to these stages.

34.2 Getting Management Buy-In

Before you can schedule a maintenance window, you will need to get approval from management. Whether the planned maintenance window will cause a disruption of services, or just reduce redundancy or capacity, management needs

to be aware of the plan and sign off on it.

SAs often like to have a maintenance window during which they can take down any and all systems and stop all services, because that approach reduces complexity and makes testing easier. For example, in cutting email services over to a new system, you need to transfer existing mailboxes, as well as switch the incoming mail feed to the new system. Trying to transfer the existing mailboxes while new email arrives and still ensure consistency is a very tricky problem. If you can bring email services down while you do the transfer, the cutover becomes a lot easier.

However, to get management buy-in, you will have to sell the concept in terms of a benefit to the company, not in terms of making the SA's life easier. You need to be able to promise better service availability the rest of the time. To deliver on that promise requires sophisticated planning in advance.

Some companies may not agree to a large scheduled outage for maintenance. In that case, an alternative plan must be presented, explaining what would be entailed if the request for maintenance windows is not granted, and demonstrating that customers—not the SAs—are the real beneficiaries. A single large outage can be much less annoying to customers than many little outages.

When a schedule for maintenance windows is agreed upon with management, all members of the team must commit to high availability for their systems. Be prepared to provide metrics from before and after you have succeeded in getting scheduled maintenance windows to back up your claims of higher availability. (Monitoring to verify availability levels is covered in [Chapter 38](#), “[Service Monitoring](#).”)

If you are involved with the startup of a new company, make a regularly scheduled maintenance window a part of the new company's culture.

Lumeta's Weekly Maintenance Windows

It can be difficult to get permission for periodic scheduled downtime. Therefore, as one of the first employees at Lumeta, Tom felt it was important to start such a tradition immediately, rather than try to fight for it later.

He sold the idea by explaining that while the company was young, the churn and growth of the infrastructure would be extreme. Rather than annoy everyone with constant requests for downtime, he promised to restrict all planned outages to Wednesday evening after 5 PM. Explained that way, the reaction was extremely positive. Because he used phrases

such as “while the company is young” rather than a specific time limit, he was able to continue this Wednesday night tradition for years.

For the first few months Tom made sure there was downtime every Wednesday night so that it would become part of the corporate culture. Often a single server was rebooted. Departments got used to planning their schedule around Wednesday night, knowing it was not a good time for late-night crunches or deadlines. Yet Tom also established a reputation for flexibility by postponing the maintenance window at the tiniest request.

Once the infrastructure was stable, the need for such maintenance windows became rare. People complained mostly when an announcement of “no maintenance this week” came late on Wednesday. Tom established a policy that any maintenance that would have a visible outage had to be announced by Monday evening and that no announcement meant no outage. While not required, he also would send an email to announce when there would be no user-visible outage. This prevented his team from becoming invisible and kept the notion of potential outages on Wednesday nights alive in people’s minds. Formatting these announcements this way trained people to pay attention when there was an actual outage.

34.3 Scheduling Maintenance Windows

In scheduling periodic maintenance windows, you must work with the rest of the company to coordinate dates when maintenance will occur. In particular, you will almost certainly need to avoid the end-of-month, end-of-quarter, and end-of-fiscal-year dates so that the sales team can enter rush orders and the accounting group can produce financial reports for that period. You also will need to avoid product release dates, if that is relevant to your business. Universities have different constraints around the academic year. Some businesses, such as toy and greeting card manufacturers, may have seasonal constraints. You must set and publicize the schedule months in advance, so that the rest of the company can plan around those times.

Case Study: Maintenance Window Scheduling

In a midsize software development company, the quarterly maintenance windows had to avoid various dates immediately before and after scheduled release dates, which typically occurred three times a year, as the engineering and operations divisions required the systems to be

operational to make the release. Dates leading up to and during the major trade show for the company's products had to be avoided because engineering typically produced new alpha versions for the show, and demos at the trade show might rely on equipment at the office. End-of-month, end-of-quarter, and end-of-year dates, when the sales support and finance departments relied on full availability to enter figures, had to be avoided. Events likely to cause a spike in customer-support calls, such as a special product promotion, needed to be coordinated with outages, although they were typically scheduled after the maintenance windows were set.

As you can see, finding empty windows is a tricky business. However, maintenance schedules were set at least a year in advance and were well advertised so that the rest of the company could plan around them.

Once the dates were set, weekly reminders were posted beginning 6 weeks in advance of each window, with additional notices being given in the final week. At the end of each notice, the schedule for all the following maintenance windows was attached, as far ahead as they had been scheduled.

The maintenance notices each highlighted a different major benefit to the company, such as bringing a new datacenter online or upgrading the mail infrastructure. This helped the customers understand the benefit they received in return for the interruption of service.

Unfortunately for the SA group, the rest of the company saw the maintenance weekends as the perfect times to schedule company picnics and other events, because no one would feel compelled to work—except for the SAs, of course.

That's life.

34.4 Planning Maintenance Tasks

As with all scheduled maintenance on important systems, the tasks involved need to be planned by the individuals performing them, so that no one tries anything unexpected. Original thought and problem solving are good things at other times, but should not be involved in performing the task during the window. There should be no unforeseen events—only planned contingencies.

Planning for a maintenance window also has another dimension, however. The rationale for a maintenance window is that no other scheduled outages are

necessary. This means that you need to plan ahead: If you have one maintenance window per quarter, you need to make sure that the work you do this quarter will hold you through the end of the next quarter, so that you won't need to bring the system down again.

Because maintenance windows occur only occasionally, the SAs need to plan far enough in advance to allow time to get quotes, submit purchase orders and get them approved, and have any new equipment arrive a week or so before the maintenance window. The lead time on some equipment can be six weeks or more, so this means starting to plan for the next maintenance window almost immediately after the preceding one has ended.

34.5 Selecting a Flight Director

The flight director is responsible for crafting the announcement notices and making sure that they go out on time, scheduling the submitted work proposals based on the interactions between them and the staff required, deciding on any cuts for that maintenance window, monitoring the progress of the tasks during the maintenance window, ensuring that the testing occurs correctly, and communicating status to the rest of the company at the end of the maintenance window.

A large multinational company that has periodic global maintenance windows, spanning datacenters in several different time zones, needs to have a team of flight directors, one in each geographic region, who hand off to each other in a “follow the sun” manner. Depending on the size of the company and the amount of work involved, the flight directors may come from a dedicated change-management team, or they may be chosen from the SA team for smaller sites.

For a company with a single campus, or several campuses in one time zone, the person who fills the role of flight director should be a senior SA who is capable of assessing work proposals from other members of the SA team and spotting dependencies and effects that may have been overlooked. The flight director also must be capable of making judgment calls on the level of risk versus need for some of the more critical tasks that affect the infrastructure. This person must have a good overview of the site and understand the implications of all the work—and look good in a vest.

In addition, the flight director cannot perform any technical work during that maintenance window. Typically, the flight director is a member of a multi-person team, and the other members of the team take on the work that would normally have been the responsibility of that individual. Because of the technical skills

required, the flight director is not usually a manager, unless the manager was recently promoted from a senior SA position.

Depending on the structure of the SA group, there may be an obvious group of people from which the flight director is selected each time. In the midsize software company discussed earlier, most of the 60 SAs took care of a division of the company. Approximately 10 SAs formed the core services unit and were responsible for central services and infrastructure that were shared by the whole company, such as security, networking, email, printing, and naming services. The SAs in this unit provided services to each of the other business units, so they had a good overview of the corporate infrastructure and how the business units relied on it. The flight director was typically a member of that unit and had been with the company for a while.

Other factors also should be taken into account, such as how the person interacts with the rest of the SAs, whether he or she will be reasonably strict about the deadlines but show good judgment where an exception should be made, and how the person will react under pressure and when tired. In our experience with this technique, we found that some excellent senior SAs performed flight director duties once and never wanted to do so again. Be careful to select a flight director who is a willing victim.

34.6 Managing Change Proposals

One week before the maintenance window, all change proposals should have been submitted. A good way of managing this process is to have all the change proposals online in a revision-controlled area. Each SA edits documents in a directory with his or her name on it. The documents supply all the required information. One week before the change, this revision-controlled area is frozen, and all subsequent requests to make changes to the documents have to be made through the flight director. A change proposal form should answer at least the following questions:

- What change are you going to make?
- Why are you making the change?
- Which machines will you be working on?
- What are the pre-maintenance window dependencies and due dates?
- What needs to be up for the change to happen?
- Which services or applications will be affected by the change?
- Will there be a disruption in services during the change? For how long?

- Who is performing the work?
- How many additional helpers are required? Anyone specific?
- How long will the change take in active time and elapsed time, including testing?
- What are the implementation steps?
- What are the test procedures? Which equipment do they require?
- What is the backout procedure, and how long will it take?

Here are some examples of change proposals that answer these questions. You can see the style and level of details that are helpful to include. These proposals are based on real-world scenarios.

34.6.1 Sample Change Proposal: SecurID Server Upgrade

Here is an example change proposal for upgrading a SecurID authentication server. Notice that the reason for the upgrade is given and special attention is paid to the dependencies and due dates.

- *What change are you going to make?*

Upgrade the SecurID authentication server software from v6.3 to v7.1.

- *Why are you making the change?*

New SecurID tokens have a longer ID number, which is supported only in the newer version. The next batch of tokens that we receive will have this longer ID number. We expect to need to start using these new tokens in two months' time, based on the current rate of new joiners and expiration dates of the tokens currently in use.

- *Which machines will you be working on?*

tsunayoshi and shingen.

- *What are the pre-maintenance window dependencies and due dates?*

The v7.1 software and license keys are to be delivered by the vendor and should arrive on September 14. The new version needs to be lab-certified, and pass an Operational Readiness Review, which is scheduled for September 25. Perform backups the night before the window.

- *What needs to be up for the change to happen?*

The network, console service, and internal authentication services.

- *Which services or applications will be affected by the change?*

All remote access and access to secured areas that require token

authentication.

- *Will there be a disruption in services during the change? For how long?*

Yes. Services should be treated as unavailable until the change is declared complete. Services may respond intermittently during the change window, but should not be relied upon.

- *Who is performing the work?*

Jane Smith, Senior Systems Associate, Core Services Department.

- *How many additional helpers are required? Anyone specific?*

None.

- *How long will the change take in active time and elapsed time, including testing?*

Three hours active; three hours elapsed.

- *What are the implementation steps?*

Install the software in a parallel directory and copy the database into the new location. Do not delete the old software or database. Stop the authentication server. Change symbolic link to point at the new software rather than the old version. Start the authentication server.

- *What are the test procedures? Which equipment do they require?*

Try to establish a VPN in, and to access each secured area. Test creating a new user, deleting a user, and modifying a user's attributes; check that each change has taken effect. Requires a laptop with VPN software and a connection to a network outside the firewall, and a machine inside the corporate network with a browser.

- *What is the backout procedure, and how long will it take?*

The old software and database are not deleted until after a week of successful running. To back out (takes five minutes, plus testing), stop the authentication server, change links to point back to the old software, and restart the authentication server.

34.6.2 Sample Change Proposal: Storage Migration

Here is another sample, this one with more dependencies and a more complicated set of implementation steps. What differences do you notice between the two versions?

- *What change are you going to make?*

Move *homede105* and *dbgene237* from *anaconda* to

anachronism.

- *Why are you making the change?*

We are running out of space on **anaconda** and we need to move some data onto another file server that has more room to grow.

- *Which machines will you be working on?*

anaconda, **anachronism**, and **shingen**.

- *What are the pre-maintenance window dependencies and due dates?*

Extra disk shelves for **anachronism** need to be delivered and installed; due to arrive September 17 and installed by September 21. Perform backups the night before the window.

- *What needs to be up for the change to happen?*

The network, console service, and internal authentication services.

- *Which services or applications will be affected by the change?*

Network traffic on 172.29.100.x network, all accounts with home directories on **homede105**—a list of affected users is attached—and database access to **Gene237**.

- *Will there be a disruption in services during the change? For how long?*

Yes. For the duration of the change window, the affected users will not be able to log in, and the **Gene237** database will be unavailable.

- *Who is performing the work?*

Greg Jones, Systems Associate, Storage and Services Division.

- *How many additional helpers are required? Anyone specific?*

A junior systems staffer with hardware experience will be required to rack and stack the additional disk shelves in the datacenter and perform routine backups.

- *How long will the change take in active time and elapsed time, including testing?*

One hour active; 12 hours elapsed.

- *What are the implementation steps?*

Disable the accounts of the affected users. Quiesce the **Gene237** database and shut it down. Take fresh backups of the data. Attach and configure the new disk shelves on **anachronism**. Copy the data to the new location. When the copy is complete, change the configuration to use the new disks rather than the original ones. Perform testing. After successful testing,

reenable user and database access. After one week without problems, delete the old data and free the disks for other use.

- *What are the test procedures? Which equipment do they require?*

Try to mount those directories from some appropriate hosts; log in to a desktop account with a home directory on *homed105*, check that it is working; start the gene database, check for errors, run the test database access script in *usrlocal/tests/gene/access-test*. Access to a non-SA workstation is required.

- *What is the backout procedure, and how long will it take?*

Shut down the database, if it has been started. Change the advertised locations of directories back to the old ones and rebuild tables. Takes 10 minutes to back out.

34.7 Developing the Master Plan

One week before the maintenance window, the flight director freezes the change proposals and starts working on a master plan, which takes into account all the dependencies and elapsed and active times for the change proposals. The result is a series of tables, one for each person, showing which task each person will perform during which time interval and identifying the coordinator for that task.

The individual tables are collated and redacted into a master chart that shows all the tasks that are being performed over the entire time, who is performing them, the team lead, and what the dependencies are. The master plan also takes into account complete systemwide testing after all work has been completed.

If there are too many change proposals, the flight director will find that scheduling all of them produces too many conflicts, in terms of either machine availability or the people required. The schedule also needs some slack for when things do go wrong. The difficult decisions about which projects should go ahead and which ones must wait should be made beforehand, rather than in the heat of the moment when something is taking too long and running over schedule, and everyone is tired and stressed. The flight director makes the call on when some change proposals must be cut and assists the parties involved to choose the best course for the company.

Case Study: Template for a Master Plan

Once our systems team had run a few maintenance windows, we discovered a formula that worked well for us. Changes to the systems on which most people were dependent for their work were made on Friday

evening. The first thing to be upgraded or changed was the network. Next on the list was the console service, then the authentication servers. While these were in progress, all the other SAs helped out with hardware tasks, such as memory, disk, or CPU upgrades; replacing broken hardware; or moving equipment within or between datacenters. Last thing on Friday night, large data moves were started so that they could run overnight.

The remaining tasks were then scheduled into Saturday, with some people being scheduled to help others in between their own tasks. Sunday was reserved for comprehensive systemwide testing and debugging, because of the high importance placed on testing.

34.8 Disabling Access

The very first task during a maintenance window is to disable or discourage system access and provide reminders about the maintenance window. Depending on what the site looks like and which facilities are available, this process may involve the following steps:

- Placing on all doors into the campus buildings notices with the maintenance window times clearly visible.
- Disabling all remote access to the site, whether by VPN, dedicated lines, or wireless.
- Making an announcement over the public address system in the campus buildings to remind everyone that systems are about to go down.
- Changing the helpdesk voicemail message to announce the ongoing maintenance window and state when normal service should be restored.

These steps reduce the chance that people will try to use the systems during the maintenance window, which could cause inconsistencies in, damage to, or accidental loss of their work. It also reduces the chance that the person carrying the oncall pager will have to respond to urgent helpdesk voicemails saying that the network is down.

34.9 Ensuring Mechanics and Coordination

Some key pieces of technology enable the maintenance window process described here to proceed smoothly. These aspects are not just useful for maintenance windows; they are critical to their success.

Before the maintenance window opens, test console servers and other tools that will be used during the maintenance window. Some of these facilities are

used infrequently enough that they may become nonfunctional without anyone noticing. Make sure to give yourself enough time to fix anything that is nonfunctional before the maintenance is due to start.

34.9.1 Shutdown/Boot Sequence

In most sites, some systems or sets of systems must be available for other systems to shut down or to boot cleanly. A machine that tries to boot when machines and services that it relies on are not available will fail to boot properly. Typically, the machine will boot but fail to run some of the programs that it usually runs on startup. These programs might be services that others rely on or programs that run locally on the desktop. In either case, the machine will not work properly, and it may not be apparent why. When shutting down a machine, it may need to contact file servers, license servers, or database servers that are in use to properly terminate the link. If the machine cannot contact those servers, it may hang for a long time or indefinitely, trying to contact those servers before completing the shutdown process. It is important to understand and track machine dependencies during boot-up and shutdown. You do not want to have to figure out why machines are hanging on boot-up after a machine room unexpectedly lost power.

The most critical systems, such as console servers, name-service machines, authentication servers, license servers, application servers, and data servers, typically need to be booted before compute servers and desktops. There also will be dependencies between the critical servers. It is vital to maintain a boot-sequence list for all datacenter machines, with one or more machines being included at each stage, as appropriate. Typically, the first couple of stages will have few machines, and maybe only one machine in them, but later stages will have many machines. All datacenter machines should be booted before any non-datacenter machines, because no machine in a datacenter should rely on any machine outside that datacenter (see [Section 17.4](#)).

An example shutdown/boot list is shown in [Table 34.2](#). The shutdown sequence is typically very close to, if not exactly the same as, the reverse of the boot sequence. There may be one or two minor differences.

Stage	Function	Reason
1	Console server	So that SAs can monitor other servers during boot.
2	Master authentication server	Secondary authentication servers contact the master on boot.
	Master name server	Secondary name servers contact the master on boot.
3	Secondary authentication server	So that SAs can log in to other servers as they boot. Unix hosts contact NIS servers when they boot. Rely on nothing but the master authentication server.
	Secondary and caching name servers	Almost all services rely on name service. Rely on nothing but the master name server.
4	Data servers	Applications and home directories are here. Most other machines rely on data servers. Rely on name service.
	Network config servers	Rely on name service.
	Log servers	Rely on name service.
	Directory servers	Rely on name service.
5	Print servers	Rely on name service and log servers.
	License servers	Rely on name service, data servers, and log servers.
	Firewalls	Rely on log servers.
	Remote access	Relies on authentication, name service, and logging.
	Email service	Relies on name, log, and directory services and data servers.
6	All other servers	Rely on servers previously booted and not each other.
7	Desktops	Rely on servers.

Table 34.2: Template for a Boot Sequence

The shutdown sequence is a vital component at the beginning of the maintenance window. The machines operated on at the beginning of the maintenance window typically have the most dependencies on them, so any machine that needs to be shut down for hardware maintenance/upgrades or

moving has to be shut down before the work on the critical machines starts. It is important to shut down the machines in the right order, to avoid wasting time bringing machines back up so that other machines can be shut down cleanly. The boot sequence is also critical to the comprehensive system testing performed at the end of the maintenance window.

The shutdown sequence can be used as part of a larger emergency power off (EPO) procedure. An EPO is a decision and action plan for emergency issues that require fast action. In particular, the plan includes situations that require action that cannot wait on management approval. Think of it as precompiling decisions for later execution. An EPO should include which situations require its activation—fire, flood, overheating conditions with no response from facilities—and instructions on how to verify these issues. A decision tree is the best way to record this information. The EPO should then give instructions on how to migrate services to other datacenters, whom to notify, and so on. Document a process for situations where there is time to copy critical data out of the datacenter and a process for when there is not. Finally, the EPO should use the shutdown sequence to power off machines. In the case of overheating, one might document ways to shut down some machines or put machines into low-power mode so they generate less heat by running more slowly but still provide services. The steps should be documented so that they can be performed by any SA on the team. Having such a plan can save hardware, services, and revenue.

Emergency Use of the Shutdown Sequence

One company found that its shutdown sequence was helpful even for an unplanned outage. The datacenter had a raised floor, with the usual mess of air conditioning conduits, power distribution points, and network cables hiding out of sight. One Friday, one of the SAs was installing a new machine and needed to run cable under the floor. He lifted a few tiles and discovered water under the floor, surrounding some of the power distribution points. He notified his management—over the radio—and after a quick discussion, he sent radio notification to the SA staff and a quick companywide broadcast. Out came the shutdown list, and the flight director for the upcoming maintenance window did a live rehearsal of shutting everything in the machine room down.

This operation went flawlessly because the shutdown list was up-to-date. In fact, management chose an orderly shutdown over tripping the emergency power cutoff to the room, knowing that there was an up-to-date shutdown list and having an assessment of how long it would take for

water and electricity to meet. Without the list, management would have had to cut power to the room, with potentially disastrous consequences.

34.9.2 KVM, Console Service, and LOM

KVM (keyboard, video, and mouse) switches, serial console servers, and lights-out management (LOM) are tools that make management of datacenter devices easier. They can be instrumental in making maintenance windows easier to run by providing remote access to the console or other functions of a machine when it is not fully up and available on the network.

A KVM switch permits multiple computers to all share the same keyboard, video display, and mouse. A KVM switch saves space in a datacenter—monitors and keyboards take up a lot of space—and makes access more convenient; more sophisticated console access systems can be accessed from anywhere in the network.

A serial console server connects devices with serial consoles—systems without video output, such as network routers, switches, and many Unix servers—to one central device with many serial inputs. By connecting to the console server, a user can then connect to the serial console of the other devices. All the computer room equipment that is capable of supporting a serial console should have its serial console connected to some kind of console concentrator, such as a networked terminal server.

LOM provides a management interface that allows an SA to remotely power-cycle the machine, in addition to providing KVM over IP and serial console access. LOM typically is integrated into the motherboard, although sometimes it can be added to a machine later. It functions even when the machine is powered off, as long as there is power connected to the machine. LOM is accessed over an additional network port that should be connected to a protected management network.

Much work during a maintenance window requires direct console access. Using console access devices permits people to work from their own desks rather than having to share the very limited number of monitors in the computer room or having to waste computer room space, power, and cooling on more monitors. It is also more convenient and efficient for the individual SAs to work in their own workspace with their preparatory notes and reference materials around them.

34.9.3 Communications

Because of the tightly scheduled maintenance window, the high number of dependencies, and the occasional unpredictability of system administration work, all the SAs have to let the flight director know when they are finished with a task, and before they start a new task, to make sure that the prerequisite tasks have all been completed.

For maintenance windows that extend beyond a single campus, the best way to communicate is typically to have one or more conference bridges that remain open during the entire maintenance window, and one or more internal instant messaging channels. The conference bridge should be provided by an external company, particularly if your company uses Voice over IP (VoIP), since the maintenance work may disrupt the internal phone systems. The chat channels become useful after SAs have finished work on the core systems—networks, firewalls, DNS, authentication, and any other core systems that are required for the messaging service. For large maintenance windows with multiple locations it is useful to have three conference bridges and three chat channels. One bridge and channel are reserved for status updates to the flight director(s)—for example, “Starting change ID 1234,” “Change ID 2345 completed successfully,” “Change ID 3456 backed out and verified.” The second bridge and channel are reserved for issues that have arisen and need to be tracked for impact on the schedule, and for ensuring service restoration before the end of the maintenance window. The third bridge and channel are for general communication between the SAs—for example, “Could I have an extra pair of hands in datacenter X, please?” and “What’s the status on racking my new disks?”

Within a single campus, consider using handheld radios to communicate within the group. Rather than seeking out the flight director, an SA can simply call over the radio. Likewise, the flight director can radio the SAs for a status update, and team members and team leaders can find one another and coordinate. If SAs need extra help, they can also ask for it over the radio. There are multiple radio channels, and long conversations can move to another channel to keep the primary one free. The radios are also essential for systemwide testing at the end of the maintenance window (see [Section 34.11](#)). We recommend radios because they are not billed by the minute and typically work better in datacenter environments than do cellphones. Remember, anything transmitted on the airwaves can be overheard by others, so sensitive information, such as passwords, should not be communicated over radios, cellphones, or pagers.

Several options exist for selecting the radios, and what you choose depends on the coverage area that you need, the type of terrain in that area, product availability, and your skill level. It is useful to have multiple channels or

frequencies available on handheld radios, so that long conversations can switch to another channel and leave the primary hailing channel open for others. [Table 34.3](#) compares options.

Type	Requirements	Advantages	Disadvantages
Line-of-sight	Frequency license Transmits through walls	Simple	Limited range Blocked by obstacles
Repeater	Frequency license Radio operator license	Better range Overcomes obstacles	More complex Skill qualifications
Cellular	Service availability	Simple Wide range Unaffected by terrain Less to carry	Higher cost Available only in cellphone providers' coverage area Company contracts may limit options Multiple channels may not be available

Table 34.3: Comparison of Radio Technologies

Line-of-sight radio communications are the most common and typically have a range of approximately 15 miles, depending on the surrounding terrain and buildings. A retailer should be able to set you up with one or more frequencies and a set of radios that use those frequencies. Make sure that the retailer knows that you need the radios to work through buildings and understands the kind of coverage you need.

Repeaters can be used to extend the range of a radio signal and are particularly useful if a mountain between campus buildings would block line-of-sight communication. It can be useful to have a repeater and an antenna on top of one of the campus buildings in any case for additional range, with at least the primary hailing channel using the repeater. This configuration usually requires that someone with a ham radio license set up and operate the equipment. Check your local laws.

Some cellphone companies offer push-to-talk features on cellphones so that phones work more like walkie-talkies. This option will work wherever the telephones operate. The provider should be able to provide maps of the coverage

areas. The company can supply all SAs with a cellphone with this service. This has the advantage that the SAs have to carry only the phone, not a phone and radio. This can be a quick and convenient way to get a new group established with radios, but may not be feasible if it requires everyone to change to the same cellphone provider.

If radios won't work or work badly in your datacenter because of radio frequency (RF) shielding, put an internal phone extension with a long cord at the end of every row. That way, SAs in the datacenter can still communicate with other SAs while working in the datacenter. At worst, they can go outside the datacenter, contact someone on the radio, and arrange to talk to that person on a specific telephone inside the datacenter.

Setting up a conference call bridge for everyone to dial in to can have the benefits of radio communication and people can dial in globally to participate. Having a permanent bridge number assigned to the group can save critical minutes during emergencies.

34.10 Change Completion Deadlines

A critical role of the flight director is tracking how the various tasks are progressing and deciding when a particular change should be aborted and the backout plan for that change executed. For a task with no other dependencies and for which those SAs involved had no other remaining tasks, in a weekend maintenance window that time would be 11 PM on Saturday evening, minus the time required to implement the backout plan. The flight director should also consider the performance level of the SA team. If the members are exhausted and frustrated, the flight director may decide to tell them to take a break or to start the backout process early if they won't be able to implement it as efficiently as they would when they were fresh.

If other tasks depend on that system or service being operational, it is particularly critical to predefine a cut-off point for task completion. For example, if a console server upgrade is going badly, it can run into the time regularly allotted for moving large data files. Once you have overrun one time boundary, the dependencies can cascade into a full catastrophe, which can be fixed only at the next scheduled downtime, which might be another three months away. In project management terms, tasks that are on the critical path *must* complete on time or early. Completion is defined as either successful implementation or successful roll-back.

34.11 Comprehensive System Testing

The final stage of a maintenance window is comprehensive system testing. If the window has been short, you may need to test only the few components that you worked on. However, if you have spent your weekend-long maintenance window taking apart various complicated pieces of machinery and then putting them back together, and done all this work under a time constraint, you should plan on spending all day Sunday doing system testing.

In a single-campus environment, Sunday system testing begins with shutting down all of the machines in the datacenter, so that you can then step through your ordered boot sequence. Assign an individual to each machine on the reboot list. The flight director announces the stages of the shutdown sequence over the radio, and each individual responds when the machine under his or her responsibility has completely shut down. When all the machines at the current stage have shut down, the flight director announces the next stage. When everything is down, the order is reversed, and the flight director steps everyone through the boot stages. If any problems occur with any machine at any stage, the entire sequence is halted until they are debugged and fixed. Each person assigned to a machine is responsible for ensuring that it shut down completely before responding and that all services have started correctly before calling it in as booted and operational.

Finally, when all the machines in the datacenter have been successfully booted in the correct order, the flight director splits the SA team into groups. Each group has a team leader and is assigned an area in one of the campus buildings. The teams are given instructions about which machines they are responsible for and which tests to perform on them. The instructions always include rebooting every desktop machine to make sure that it turns on cleanly. The tests could also include logging in, checking for a particular service, or trying to run a particular application, for example. Each person in the group has a stack of colored sticky tabs used for marking offices and cubicles that have been completed and verified as working. The SAs also have a stack of sticky tabs of a different color to mark cubicles that have a problem. When SAs run across a problem, they spend a short time trying to fix it before calling it in to the central core of people assigned to stay in the main building to help debug problems. As it finishes its area, a team is assigned to a new area or to help another team to complete an area, until the whole campus has been covered.

Meanwhile, the flight director and the senior SA troubleshooters keep track of problems on a whiteboard and decide who should tackle each problem, based on the likely cause and who is available. By the end of testing, all offices and cubicles should have tags, preferably all indicating success. If any offices or

cubicles still have tags indicating a problem, a note should be left for that customer, explaining the problem; someone should be assigned to meet with that person to try to resolve it first thing in the morning.

This systematic approach helps to find problems before people come in to work the next day. If there is a bad network segment connection, a failed software push, or problems with a service, you'll have a good chance to fix it before anyone else is inconvenienced. Be aware, however, that some machines may not have been working in the first place. The reboot teams should always make sure to note when a machine did not look operational before they rebooted it. They can still take time to try to fix it, but that effort is lower on the priority list and does not have to happen before the end of the maintenance window.

Ideally, the system testing and site-wide rebooting should be completed sometime on Sunday afternoon. This gives the SA team time to rest after a stressful weekend before coming into work the next day.

In larger companies, with redundant datacenters, the systemwide testing looks a little different. For example, some companies may be able to do a complete shutdown and restart on only a single datacenter during a maintenance weekend, because of availability requirements. Some large companies involve application support teams in the system testing—each application team has its own test suite, and certifies its own applications as working after the maintenance window. Since these teams know their applications far better than the SAs do, their testing can be much more thorough.

Companies often have a wide variety of end-user devices: workstation models, IP phones, printers, and so on. Each of these devices should be tested after the maintenance window. Verify that each can successfully reboot and operate. This is easier to do if there is a test lab with one of each device used in the company.

34.12 Post-maintenance Communication

Once the maintenance work and system testing have been completed, the flight director sends out a message to the company, informing everyone that service should now be fully restored. The message briefly outlines the main successes of the maintenance window, lists any services that are known not to be functioning, and indicates when the nonfunctioning services will be fixed.

This message should be in a fixed format and written largely in advance, because the flight director will be too tired to write a coherent message at the end of a long weekend. There is little chance that anyone who proofreads the

message at that point will be able to help, either.

Hidden Infrastructure

Sometimes you must find creative ways to deal with hidden infrastructure that is outside of your control.

A site had a planned outage, and all power to the building was shut off. Servers were taken down in an orderly manner and brought back successfully. However, there were some unofficial servers that the IT staff didn't know about.

The next morning, the following email exchange took place:

[Click here to view code image](#)

```
From: IT
To: Everyone in the company
All servers in the Burlington office are up and running. Should
you
have any issues accessing servers, please open a helpweb ticket.
```

```
From: A Developer
To: IT
Devwin8 is down.
```

```
From: IT
To: Everyone in the company
Whoever has devwin8 under their desk, turn it on, please.
```

34.13 Reenabling Remote Access

The final act before leaving the building should be to reenable remote access and restore the voicemail on the helpdesk phone. Make sure that this step appears on the master plan and the individual plans of those responsible. It can be easily forgotten after an exhausting weekend, but remote access is a very visible, inconvenient, and embarrassing thing to forget—especially because it can't be fixed remotely if all remote access was turned off successfully.

34.14 Be Visible the Next Morning

It is very important for the entire SA group to be in early and to be visible to the company the morning after a maintenance window, no matter how hard they have worked during the maintenance window. If everyone has company or group shirts, coordinate in advance of the maintenance window so that all the SAs wear those shirts on the day after the outage. Have the people who look after

particular departments roam the corridors of those departments, keeping eyes and ears open for problems.

Have the flight director and some of the senior SAs from the central core-services group, if there is one, sit in the helpdesk area to monitor incoming calls and listen for problems that may be related to the maintenance window. These people should be able to detect and fix these issues sooner than the regular helpdesk staff, who won't have such an extensive overview of what has happened.

A large visible presence when the company returns to work sends the message: "We care, and we are here to make sure that nothing we did disrupts your work hours." It also means that any undetected problems can be handled quickly and efficiently, with all the relevant staff on-site and not having to be paged out of their beds. Both of these factors are important in the overall satisfaction of the company with the maintenance window. If the company is not satisfied with how the maintenance windows are handled, the windows will be discontinued, which will make preventive maintenance more difficult.

34.15 Postmortem

By about lunchtime of the day after the maintenance window, most of the remaining problems should have been found. At that point, if it is sufficiently quiet, the flight director and some of the senior SAs should sit down and talk about what went right, and why; what went wrong, and why; and which improvements and optimizations can be made. That should all be noted and discussed with the whole group later in the week. Over time, with the postmortem process, the maintenance windows will become smoother and easier to carry out. Common mistakes early on tend to include trying to do too much in the allotted time, not doing enough work ahead of time, and underestimating how long something will take.

34.16 Mentoring a New Flight Director

Eventually you will need to train a new flight director. This person should shadow the current flight director during at least one maintenance window, which means that the new person should be selected far in advance.

The trainee flight director can produce the first draft of the master plan by using the change requests that were submitted, adding in any dependencies that are missing, and tagging those additions. The flight director then goes over the plan with the trainee, adds or subtracts dependencies, and reorganizes the tasks

and personnel assignments as appropriate, explaining why. Alternatively, the flight director can create the first draft along with the trainee, explaining the process while doing so.

Prior to the downtime, the trainee can handle tasks such as reaching out to SAs when clarifications are needed, ensuring that the prerequisites listed in the change proposal are met, and so on.

During the maintenance window, the trainee can gain experience by being in charge of tracking and allocating resources for a few designated subprojects.

34.17 Trending of Historical Data

It is useful to track how long particular tasks take and then analyze the data later so as to improve on the estimates in the task submission and planning process. For example, if you find that moving a certain amount of data between two machines took eight hours and you are planning a large data move between two similar machines on similar networks at another time, you can more accurately predict how long it will take. If a particular software package is always difficult to upgrade and takes far longer than anticipated, that issue will be tracked, anticipated, allowed for in the schedule, and watched closely during the maintenance interval.

Trending is particularly useful in passing along historical knowledge. When someone who used to perform a particular function has left the group, the person who takes over that function can look back at data from previous maintenance windows to see which sorts of tasks are typically performed in this area and how long they take. This data can give people new both to the group and to planning a maintenance window a valuable head start so that they don't waste a maintenance opportunity and fall behind.

For each change request, record the actual time to completion for use when calculating time estimates the next time around. Also record any other notes that will help improve the process next time.

34.18 Providing Limited Availability

It is highly likely that at some point, you will be asked to keep service available for a particular group during a maintenance window. The issue at hand may be something unforeseen, such as a newly discovered bug that engineering needs to work on all weekend, or it may be a new mode of operation for a division, such as customer support switching to 24/7 service and needing continuous access to its systems to meet its contracts. Internet services, remote access, global

networks, and new-business pressure reduce the likelihood that a full and complete outage will be permitted.

Planning for this requirement could involve re-architecting some services or introducing added layers of redundancy to the system. It may involve making groups more autonomous or distinct from one another. Making these changes to your network can be significant tasks by themselves, likely requiring their own maintenance window; it is best to be ready for these requests before they arrive, or you may be left without time to prepare.

To approach this task, find out what the customers will need to be able to do during the maintenance window. Ask a lot of questions, and use your knowledge of the systems to translate these needs into a set of service-availability requirements. For example, customers will almost certainly need name service and authentication service. They may need to be able to print to specific printers and to exchange email within the company or with customers. They may have VoIP phones that need to remain up. They may require access to services across wide-area connections or across the Internet. They may need to use particular databases; find out what those machines, in turn, depend on.

Look at ways to make the database machines redundant so that they can also be properly maintained without loss of service. Make sure that the services they depend on are redundant. Identify which pieces of the network must be available for the services to work. Look at ways to reduce the number of networks that must be available by reducing the number of networks that the group uses and locating redundant name servers, authentication servers, and print servers on the group's networks. Find out whether small outages are acceptable, such as a couple of ten-minute outages for reloading network equipment. If not, the company needs to invest in redundant network equipment.

Devise a detailed availability plan that describes exactly which services and components must be available to that group. Try to simplify it by consolidating the network topology and introducing redundant systems for those networks. Incorporate availability planning into the master plan by ensuring that redundant servers are not down simultaneously.

34.19 High-Availability Sites

By the very nature of their business, high-availability sites cannot afford to have large planned outages. This also means that they cannot afford *not* to make the large investment necessary to provide high availability. Sites that have high-availability requirements need to have lots of hot redundant systems that

continue providing service when any one component fails. The higher the availability requirement, the more redundant systems that are required to achieve it. (Recall that $N + 1$ redundancy is used for services such that any one component can fail without bringing the service down, $N + 2$ means any two components can fail, and so on.)

These sites still need to perform maintenance on the systems in service. Although the availability guarantees that these sites make to their customers typically exclude maintenance windows, they will lose customers if they have large planned outages.

34.19.1 The Similarities

Most of the principles described previously for maintenance windows at a corporate site apply to high-availability sites:

- High-availability sites need to schedule the maintenance window so that it has the least impact on their customers. For example, ISPs often choose 2 AM (local time) midweek; e-commerce sites need to choose a time when they do the least business. These windows will typically be quite frequent, such as once a week, and they are shorter, perhaps four to six hours in duration.
- These sites need to let their customers know when maintenance windows are scheduled. For ISPs, this means sending an email to their customers. For an e-commerce site, it means having a banner on the site. In both cases, the message should be sent only to those customers who may be affected, and should contain a warning that small outages or degraded service may occur during the maintenance window and give the times of that window. There should be only a single message about the window.
- Planning and doing as much as possible beforehand is critical because the maintenance windows should be as short as possible.
- There must be a flight director who coordinates the scheduling and tracks the progress of the tasks. If the windows occur on a weekly basis, this may be a quarter-time or half-time job.
- Each item should have a change proposal. The change proposal should list the redundant systems and include a test to verify that the redundant systems have kicked in and that service is still available.
- The site needs to tightly plan the maintenance window. Maintenance windows are typically smaller in scope and shorter in time. Items scheduled by different people for a given window should not have

dependencies on each other. There must be a small master plan that shows who has which tasks and their completion times.

- The flight director must be very strict about the deadlines for change completion.
- Everything must be fully tested before it is declared complete.
- LOM, remote KVM, and console access benefit all sites.
- The SAs need to have a strong presence when the site approaches and enters its busy time. They need to be prepared to deal quickly with any problems that may arise as a result of the maintenance.
- A brief postmortem the next day to discuss any remaining problems or issues that arose during the window is useful.

34.19.2 The Differences

Maintenance windows for high-availability sites differ in the following ways:

- A prerequisite is that the site must have redundancy if it is to have high availability.
- It is not necessary to disable access. Services should remain available.
- Because ISPs and e-commerce sites do not have on-site customers, being physically visible the morning after the window is irrelevant. However, being available and responsive is still important. SAs facing this situation must find ways to increase their visibility and ensure excellent responsiveness. They should advertise what the change was, how to report problems, and so on. They might maintain a blog, have a corporate social media presence, or put banner advertisements on the internal web sites advertising the newest features.
- A post-maintenance communication is usually not required, unless customers must be informed about remaining problems. Customers don't want to be bombarded with email from their service providers.
- The most important difference is that the redundant architecture of the high-availability site must be taken into account during the maintenance window planning. The flight director needs to ensure that none of the scheduled work can take the service down. The SAs need to ensure that they know how long failover takes to happen. For example, how long does the routing system take to reach convergence when one of the routers goes down or comes back up? If redundancy is implemented within a single machine, the SA needs to know how to work on one part of the machine

while keeping the system operating normally.

- Availability of the service as a whole must be closely monitored during the maintenance window. There should be a plan for how to deal with any failure that causes an outage as a result of a temporary lack of redundancy.

34.20 Summary

The basics for successfully executing a planned maintenance window fall into three categories: preparation, execution, and post-maintenance customer care. The advance preparation for a maintenance window has the greatest effect on whether it will run smoothly. Planning and doing as much as possible in advance are key. The SA group needs to appoint an appropriate flight director for each maintenance window. Change proposals should be submitted to the flight director, who then uses them to build a master plan and set completion deadlines for each task.

During the maintenance window, remote access should be disabled, and infrastructure, such as console servers and radios, should be in place. The plan needs to be executed with as few hiccups as possible. The timetable must be adhered to rigidly; it must finish with complete system testing.

Good customer care after the maintenance window is important to its success. Communication about the window and a visible presence the morning after the window ends are key.

Integrating a mentoring process, saving historical data and doing trend analysis to improve estimates, providing continuity, and providing limited availability to groups that request it are elements that can be incorporated into the process at a later date. Proper planning, good backout plans, strict adherence to deadlines for change completion, and comprehensive testing should avert all but some minor disasters. Some tasks may not be completed, and those changes will need to be backed out. A well-planned, properly executed maintenance window never leads to a complete disaster. A badly planned or poorly executed one could, however. These kinds of massive outages are difficult to manage and risky for the organization.

Exercises

1. Read the paper on how the AT&T/Lucent network was split ([Limoncelli, Reingold, Narayan & Loura 1997](#)), and consider how having a weekend maintenance window would have changed the process. Which parts of that project would have been performed in advance as preparatory work, which

parts would have been easier, and which parts would have been more difficult? Evaluate the risks in your approach.

2. A case study in [Section 34.3](#) describes the scheduling process for a particular software company. What are the dates and events that you would need to avoid for a maintenance window in your company? Try to develop a list of dates, approximately three months apart, that would work for your company.
3. Consider the SAs in your company. Who do you think would make good flight directors and why?
4. Which tasks or projects at your site would be appropriate for a maintenance window? Create and fill in a change-request form. What could you do to prepare for this change in advance of the maintenance window?
5. [Section 34.8](#) discusses disabling access to the site. Which specific tasks would need to be performed at your site, and how would you reenable that access?
6. [Section 34.9.1](#) discusses the shutdown and reboot sequence. Build an appropriate list for your site. If you have permission, test it.
7. [Section 34.18](#) discusses providing limited availability for some people to be able to continue working during a maintenance window. Which groups are likely to require 24/7 availability? Which changes would you need to make to your network and services infrastructure to keep services available to each of those groups?
8. Research the flight operations methodologies used at NASA. Relate what you learned to the practice of system administration.

Chapter 35. Centralization Overview

This chapter is about the differences between centralized and decentralized services, the benefits of each, and when to change a service's structure from one to the other. The level of centralization influences the cost and quality of the service. The chapters that follow will discuss our recommendations for which services and organizational functions should be centralized, and how to transition a service from one to the other.

Services may be organized in ways that are centralized, decentralized, or distributed. An example of a **centralized model** would be an entire company using one large email platform located in its headquarters. Remote sites connect to the platform over the company's network. A **decentralized model** would be where each division designs, creates, and operates its own email system. Each might use an entirely different set of products. Each would have different policies; for example, one division might have higher disk space limits for its users. A **distributed model** would be where one centralized organization creates a single standard and deploys it once in each division and then centrally manages and operates them all.

An organization can be structured similarly. A large company might have a centralized IT model where a single IT organization provides services to the entire company; the division you are from would be inconsequential to which services you receive in this case. Alternatively, it may have a decentralized model where each division has its own independent IT department, each with its own policies and procedures, each implementing a different set of services, and each using different technologies. In yet another alternative, in a distributed model there would be many teams, one for each division or location. They all work as one team, uniformly offering the same services the same way, but with different people in each division or location.

Often we find an existing organization or service follows one model but would benefit from following another model. The resulting change process is called **centralization**, if reorganizing into a centralized model, or **decentralization**, if moving into a decentralized model.

35.1 Rationale for Reorganizing

IT services represent a significant cost of doing business for most organizations. Reducing that cost through increased efficiency is one of the main drivers behind

centralization. However, end-user satisfaction is typically linked to having on-site support and fast response times, which can lead to a decentralized model. Organizations need to find a balance between these two models to realize the benefits of both.

35.1.1 Rationale for Centralization

Centralization brings efficiency by reducing duplication of effort. For example, new versions of software are tested, certified, and deployed by a single central team, rather than having a team in each department or region going through the same process. Vendor evaluations and selection are also performed by a single team, which not only reduces duplication of effort in the vendor selection process, but also allows the company to negotiate better discounts through larger volume purchasing.

Centralization should also improve the quality of the service. Centralization encourages specialization, with SAs who are dedicated to maintaining and improving each component, rather than spending a little bit of time in many different areas. When an SA has to support many different services, he or she is constantly pulled in many directions at once, and has the time to do only the bare minimum to keep each service running. When an SA is focused on a single service, he or she can make continuous improvements to all aspects, such as reliability, provisioning, monitoring, performance, security, capacity, and documentation. Through specialization, centralization enables automation, which leads to a more reliable service and faster turnaround times. Centralization forces SAs to build systems in a way that they can be remotely managed.

35.1.2 Rationale for Decentralization

If centralization has so many benefits, why do so many companies have decentralized IT organizations? At many companies, this model is a historical legacy. Companies may have merged with or acquired other companies but never merged their IT departments, leaving each division with its own IT organization. Some companies simply have no unifying IT strategy, such that small IT groups spring up out of demand.

Sometimes decentralization takes the form of a division or customer group that decides to break away from a centralized service or organization. The motivation is usually one of the following:

- **Low quality:** The group may feel they can provide a better-quality service, higher availability, or faster response to outages if they do it

themselves.

- **Needs not met:** The central service does not meet certain needs, and no amount of negotiation has resulted in a commitment for the features to be added. The central organization may not have the staff to address the issues.
- **Lack of customizability:** To be efficient, a centralized service may be provided in a way that restricts customization. People may be unhappy with the onsize-fits-all service if it impedes their work.
- **Cost:** Groups may feel they can perform a service at a lower cost (though often this simply means the cost is hidden elsewhere).
- **Desire for new features:** A group may want to stay on the cutting edge of technology and be dissatisfied with a central IT organization that lags behind. Central services often become calcified and lose the will to support newer features.

Going it alone should be a last resort. If one is to break away from a centralized service, that step should be taken only after careful consideration. After a single major outage, there may be calls to break off from the central service and go it alone. Writing a business case that takes into account all the costs not only provides a way to make sure all issues are thought out, but also gives time for cooler heads to prevail.

Quality Drives Adoption

A university introduced a centralized “managed PC” service. For a certain monthly payment per PC or laptop, the machine would be provided, the OS and applications would be upgraded automatically, and the device would be replaced every three years. Over the three-year period the department would be billed a total that was much more than the initial purchase price of a PC, but the value was in the support. Departments balked at the offering because they believed they were already doing it themselves for less money. They were using graduate students to manage their PCs, and that labor was considered “free.”

What convinced departments to use the managed service was that the early adopters reported how much better things were now that they had trained professionals maintaining their PCs instead of graduate students, and the realization that using the service freed up the graduate students to have more time for research. The dean hadn’t realized that for many graduate students, their IT responsibilities had essentially become a full-

35.2 Approaches and Hybrids

Centralized means having one focus of control. This definition applies to the technology: a unified architecture, and centralized systems control and management. It also applies to the management of the IT team.

Defaulting to a centralized approach creates a solid, efficient environment. Decentralization of any aspect of the environment should be a well-justified exception, where there are good reasons why a centralized service cannot meet the requirements. Be judicious and cautious about approving exceptions, and set down clear guidelines, with periodic review where continuing an exception needs re-justification on an annual basis. Be aware that one exception is always used to justify the next, which can quickly lead down the path to anarchy.

When defining the strategy for your organization, your goal should be to deliver the best possible service for the lowest possible price. Examine each function to evaluate the benefits and potential pitfalls of centralization.

Hybrid strategies include the following:

- **Strictly centralized service:** The service is run by a central group, there are no others, and the service is self-contained. People interact with it only as a user, or else the service is invisible to most people. One example is a payroll system. Outside of the group, people interact with this system via a web interface. Another example is core infrastructure, such as email, Internet access, authentication, and voice services.
- **Centralized platform:** The service is centralized but individual teams or departments can subscribe to it and use it as necessary. One example of a centralized platform is a backup service that individual departments use for very different purposes—for example, home directories versus databases. The interface gives them flexibility to meet their needs. Another example might be internal web services, with a central team managing the service, including web servers, search capabilities, a knowledge base, and documentation repository. Different departments subscribe to various parts of the service, but manage their own content.
- **Centralized with decentralized delegation:** There is a central team that is responsible for the service's architecture, maintenance, oncall responsibility, and so on. However, some support elements are delegated to decentralized units. For example, in large companies with geographically

diverse offices, it is typical that the WAN team is in the headquarters and rarely, if ever, visits the individual offices. If a router needs to be swapped out, local helpdesk people can be walked through the process. If an office is too small to have any technical staff, hardware vendors provide on-site part-swapping services. Sometimes small offices are rented from facilities management companies that provide security and cleaning services, and that can also provide on-site technicians to replace broken hardware, plug cables into switch ports, and so on.

- **Centralized with well-defined exceptions:** The service is centralized but independent teams can set up their own services in parallel. Setting up separate services introduces duplication of effort, and therefore increased costs, so approval is required from both the management chain that runs the centralized service and the management chain of those seeking to set up their own service. The approvals should come from a high level, such as the chief information officer (CIO) on the IT side, and a similar-level manager in other parts of the organization.
- **Centralized with leaky exceptions:** This is similar to the previous strategy but the approval process has “no teeth” and all exception requests are approved. This may occur when the exception approvals need to come from a low-level manager who is easily overruled, or when the CIO does not support the centralized model. This model is a disaster because it leads to, essentially, no centralization. We recommend against this model but it’s common enough that it should be mentioned.

35.3 Summary

Services can be centralized, decentralized, or distributed. Centralized means the service is provided by one team for the benefit of the entire company.

Decentralized means each team or division provides the service for itself.

Distributed is a hybrid where one centralized organization creates a single service but deploys it where needed, with centralized management.

The IT organization itself may be organized similarly. Some companies have one IT organization, in others each division has an IT organization, and so on.

Centralization is the process of moving toward a centralized model. Centralization emphasizes reduced cost via mass-production. The goal is to save money by reducing duplicate effort. Decentralization emphasizes specialization. When a service is decentralized, it is usually because customers are dissatisfied and break away from the centralized authority to achieve custom results.

We believe that the ideal situation is a centralized or distributed service that judiciously permits the features (and customizability) that customers need. Ideally services are self-service, empowering the users to be self-sufficient. The centralized aspect saves money by reducing duplication of effort. Some of those savings should be put toward building a service that is higher quality than the individual customers could achieve on their own.

Exercises

1. What does it mean for a service to be centralized, decentralized, or distributed?
2. What are the motivations for changing from a decentralized to a centralized model?
3. Which model is best for IT desktop support services?
4. Which model is best for a corporate email service?
5. What are the main drivers toward centralization?
6. What are the main drivers toward decentralization?
7. How can centralization work in a global company?
8. How centralized or decentralized is your current environment? Give several examples.
9. When are regional teams appropriate?
10. Give an example of a service or an aspect of your organization that should be centralized, and explain why.
11. Give an example of a service or an aspect of your organization that should be decentralized, and explain why.
12. Describe a small centralization project that would improve your current system.

Chapter 36. Centralization Recommendations

In this chapter we take a number of IT responsibilities and make general recommendations about using centralized or decentralized structures for each. The examples will focus on areas related to architecture, security, infrastructure, IT support, and other services.

36.1 Architecture

For site architecture we recommend the strictly centralized service model. Site architecture involves determining how services should be deployed within a company, how various systems should interact with each other, how the network and datacenters should be designed, and so on.

With the centralized approach the company has a unified architecture, and all services are built the same way in all regions and all divisions of the company. Each service can be designed within the context of the architecture of the site as a whole, and interactions between different services and systems are considered. A unified architecture function makes the site easier to support. It makes it easier to document how things work, and to bring new staff up to speed. It also makes it easier and more cost-effective to automate repetitive tasks, as the automation applies across the whole company, with limited or no customizations required.

36.2 Security

For the security of the site we recommend the strictly centralized service model. Site security involves defining global policies governing access to corporate assets, and introducing technological and process-based solutions to implement those policies.

A strictly centralized approach is the best because when security is not treated holistically, the discrepancies in policies result in an inconsistent security model for the company. Inconsistent security and insufficient access controls for sensitive data are major causes of security breaches. To be effective, security controls need to be holistic and well coordinated across the company.

Some companies may need regional additions to the security policy, due to local regulations. For example, in the banking industry there are several countries that have special rules in relation to client confidentiality, service availability, and administration of services. With a strictly centralized service model, these locations can have local security policies and measures in addition

to the global ones, without creating inconsistencies that can be leveraged to compromise the site’s security.

Inconsistent Security Policies

Many years ago one company had two divisions, A and B, as a result of a merger. In division A, the security model was based on controlling access to the network through firewalls, and trusting anything that was on the network. In division B, the model was to carefully control access to each device. Division B had open connections to customer sites, to allow customers carefully controlled access to some specialized hardware.

Since the two divisions were part of the same company and shared a network, the overall security of the company was flawed. The open connections in division B could be (and were) used to compromise the open hosts in division A. Once the machines in division A were compromised, the attacker could leverage that foothold to gain access to hosts in division B through people who had access to both systems.

Within the security area, there are various components for which we would recommend a different approach, as shown in [Table 36.1](#). The following sections describe those components in more detail.

Security Component	Recommended Approach
Authentication	Strictly centralized service
Authorization	Centralized platform
Internet access	Strictly centralized service
Remote access	Strictly centralized service
Extranet connections	Centralized platform
New acquisitions	Strictly centralized service
Malware detection	Strictly centralized service
Data leakage prevention	Strictly centralized service or Centralized with decentralized delegation

Table 36.1: Centralization for Security Services

36.2.1 Authorization

Authorization means defining resource access entitlements. Authentication is how someone identifies himself to the system. Together, authentication and

authorization are used by access control systems, which permit or deny access based on who is accessing a system and what that person is authorized to access.

Strictly centralized authentication means users have a single set of credentials that are used for everything at the company, rather than having to remember which set of credentials to use for each service.

Providing a centralized platform for authorization enables resource owners and internal application developers to easily define their own resource access policies. It provides a central place where end users can apply for access, but allows the resource owners to define their own approval process and approvers. It is also a central place where a user's access entitlements can be reviewed, and it can be hooked into the HR systems to provide automated provisioning and revocation of access rights based on joiner, leaver, and role changes.

Such centralization permits a single sign-on (SSO) experience for users. They can log in once and have access to all permitted services.

36.2.2 Extranet Connections

Extranet connections are physical or virtual private network links to third parties, such as other companies. Usually additional services are provided by or to the third party across these links beyond what is generally available on the Internet. Sometimes the same services are offered, but the connection is in place for increased speed, reliability, or privacy.

We recommend the centralized platform model because extranet links are a component of site security where there normally are several different access models, with different requirements. The central team needs to provide a set of standard extranet access models that other groups can subscribe to for their extranet needs.

Design of new standard models to meet new requirements is performed by a central team that makes sure that the models comply with all security policies, and are defined in such a way that infrastructure costs can be shared across subscribers and future instances will be faster to set up. To promote cost sharing, extranet connections should terminate in one or more geographic hub sites, rather than in the local office that first wants the connection.

36.2.3 Data Leakage Prevention

Many companies implement **data leakage prevention** (DLP) controls to prevent accidental or deliberate sharing of confidential data or intellectual property outside the company. For most companies these security measures should be

implemented as a strictly centralized service. However, in companies where local regulations dictate that certain information may not leave the country, for example, it may be necessary to set up this service as a centralized service with decentralized delegation to local teams who manage and review their local set of rules and detection events.

36.3 Infrastructure

Infrastructure is used to provide service to the whole company. It is not specific to one particular group of end users. Infrastructure services should be unified, giving the same level of reliability and service across the whole company. Centralizing infrastructure services ensures universal compatibility, simplifies documentation and support, reduces costs, fosters automation, and improves services.

Within the infrastructure area, there are various components for which we would recommend different approaches, as shown in [Table 36.2](#).

Infrastructure Component	Recommended Approach
Datacenters	Centralized with decentralized delegation
Networking	Centralized with decentralized delegation
IP address management	Centralized with decentralized delegation
Inventory management	Strictly centralized service
Software depot	Strictly centralized service
Configuration management	Strictly centralized service
Namespace management	Strictly centralized service
Communications	Centralized platform
Data management	Centralized platform
Monitoring	Centralized platform
Logging	Centralized platform
Automated systems	Strictly centralized service
Private cloud	Strictly centralized service

Table 36.2: Centralization for Infrastructure Services

36.3.1 Datacenters

For global companies it is important to make sure that all datacenters are designed and run the same way. Datacenter design, processes, and procedures should be centrally defined and managed. Teams that are providing global

services—whether they be infrastructure services, HR systems, or a core business service—need to be able to deploy servers and services in the same way everywhere, not through custom solutions for each datacenter. A large company may have different classes of datacenters with a specific list of infrastructure available in each. That requires global datacenter standards that specify what is required: which core infrastructure is in each datacenter; how much power and cabling, and which network equipment, are in each rack; and so on.

These sorts of standards are easy to specify and adhere to when building out a new datacenter. However, certifying and deploying new technology to existing datacenters requires a central team to ensure that the datacenters stay in sync.

Datacenter operations are delegated to local teams, reporting to a single global head of datacenter operations. These regional teams are the only ones who have physical access to the datacenters to deploy new equipment, remove retired equipment, perform hardware maintenance tasks, and bring the datacenter up-to-date with the latest standards as they evolve.

Problems with Nonstandardized Datacenters

At one large global company that had many datacenters in each country, new infrastructure had been deployed haphazardly. When an SA wanted to put a machine into a datacenter, it would take weeks before he or she could track down a datacenter room that had the required security zone, SAN connections, and sufficient capacity for the device. Sometimes it was necessary to run new fiber connections to a room that had capacity and was in the right security zone, but lacked the required SAN.

The difficulties associated with putting a machine into a datacenter delayed many projects. Because there was no single central team responsible for the global datacenter design and deployment, the datacenters had drifted apart, with significant adverse impact on the company.

36.3.2 Networking

The most basic building block of any IT infrastructure is the network. The network includes everything from the LAN to the WAN. We recommend centralized network design and tooling, with decentralized delegation to regional teams. The network should have a single coherent architecture, using standard components and solutions deployed globally. It should be easy to understand and maintain. The regional teams follow the global standards, and use the same

standard set of tools.

There should also be a single central network operations center (NOC) that has the full picture of everything that is going on in the network. The central NOC escalates problems to the regional teams, and the regional teams inform the NOC when they are making changes. This allows the NOC to identify when a reported issue is related to a scheduled change, and it can delay or cancel the change if appropriate.

The NOC has an overall view of what is happening in the network, and coordinates with the regional operations centers (ROCs). The regional teams are the ones who are primarily responsible for the local networks. The NOC has primary responsibility for the WAN. The NOC does triage for network incidents, and acts as the coordinator. The NOC and the ROCs report to the same global network operations manager.

For companies with many small branch offices, it is common to use a facilities company that takes care of everything from the security systems and building maintenance to cleaning and caring for plants. Such facilities companies can often also provide a remote hands service, doing cabling or replacing broken hardware. The personnel performing these tasks work under the direction of the ROCs.

36.3.3 IP Address Space Management

IP address management (IPAM) is closely tied to network management. In large companies, large blocks of address space are allocated to regional teams, who then allocate subnets for specific office, datacenter, or backbone usage.

Central management of the IP address space ensures that networks are not used in more than one location. It also makes it easier to define clear interfaces between regions to reduce route flapping across a global network. In addition, it makes it possible to define standards, such as allocating particular blocks of private addresses to be used for backup networks that are not routed between regions.

36.3.4 Namespace Management

Centralizing DNS services and management makes support easier and often enables better automation. It reduces the complexity that can make debugging difficult, especially when problems have a DNS or DHCP component. When DNS is not centralized, DNS zones need to be delegated to independent systems that are managed by different teams. These delegations are a potential source of

failure, as updating them may be forgotten when machines are replaced.

Outage Due to Decentralized DNS

At one site, the recursive caching DNS servers that all clients used for queries were managed by team A. The parent domain for the company was managed by team B, and a particular Microsoft ActiveDirectory (AD) DNS zone was managed by team C on the AD domain controllers (DCs).

Team C needed to upgrade all of the DCs, so it did a rolling upgrade. Every second weekend, this team brought a new DC online with a new name and IP address, and decommissioned an old one. The cutover ran smoothly, with the new servers taking over services as expected.

The name server records for the new DCs were automatically added to the appropriate DNS zones in AD, and the caching DNS servers learned about any new name servers as the caches expired. However, as DNS was not the core competency of team C, they forgot to tell team B to update the zone delegation.

The system continued to work as long as one old-style DC existed. Finally the last old DC was replaced. The next day, people started noticing that some of team A's DNS servers were answering queries for this zone, but others were not. Team A needed to figure out what was going on. The difference in behavior was due to the cached list of NS records expiring from some caches, which then retrieved the list of delegated (retired) name servers from the parent domain. Team A had to get teams B and C together to fix the delegations.

In an environment with a single central DNS management system, this outage would never have happened. A centralized team that has DNS as a core competency would not have forgotten to update the delegations.

With the right centralized management system, there are no zone delegations outside the system, and the management system ensures that it is self-consistent. Errors due to one team forgetting about a dependency on an additional configuration in another team's service are less likely to happen.

36.3.5 Communications

Communications systems include email, chat, voice, and video conferencing. With a centralized platform, different individuals and departments may subscribe to different services, and be charged accordingly. For example, at some

companies perhaps not everyone needs desktop video conferencing, or some people may have soft phones while others have hard phones, or use of telepresence rooms may be charged back to the department.

Customers have high expectations about communication systems. They want to be able to communicate with anyone, at any time, and they expect the system to always work. It is much easier and cheaper to achieve and maintain such universal communication when those services are managed and maintained by one team. The same products, versions, and configurations should be used in all locations, ensuring compatibility. Interactions among this limited set of standard products can be thoroughly tested and managed. For services that require special network configuration, such as quality of service or power over Ethernet, the central communications team can work with the central networking team to engineer a solution that works for the whole company, rather than having to repeat the exercise multiple times and perhaps end up with incompatible solutions.

36.3.6 Data Management

Data management includes storage and backups. These services are used by many other global services. The centralized platform approach enables customers to select the amount, type, speed, and location of their data storage; it also provides an SLA.

Storage and backups are complex, technical services. Expecting every team to have a high degree of expertise in these areas is unrealistic. Providing the service on behalf of others not only amortizes expertise across more groups, but also empowers smaller teams to back up data that previously was not, or was backed up badly.

As we mention in [Chapter 44](#), “[Backup and Restore](#),” it is important to ensure that you can restore data. Enforcing the discipline of regularly testing the ability to restore data is easier in a centralized service. There is no point in doing backups if you do not know that you can do restores. It is easier to enforce such policies at a single choke point.

Case Study: Bidirectional SLA for Backups

At Google the centralized backup system has a bidirectional SLA. Customers of the backup solution have to prove that they can do a restore on the live system while the service is still running. They must do this at least once every six months, or those systems are no longer backed up.

The basis of this bidirectional SLA is that backups are of no use unless you can do a restore.

36.3.7 Monitoring

Centralized monitoring has many benefits. Monitoring is difficult. One team can focus on it and do it well. Otherwise, every team will do it with varying degrees of quality. Centralization also reduces the chance that multiple teams will poll the same device or service. It enables correlation between events so that the root cause of a problem can be identified more quickly.

Providing monitoring as a centralized platform means that anyone can subscribe to the monitoring service. Subscribers need to be able to define

- What to monitor
- How to monitor it
- Their own monitoring plug-ins
- How frequently something is monitored
- Which data retention and consolidation policy should be applied
- Which events, if any, should trigger alerts
- During which time frames each alert should be triggered
- Which alerting mechanism to use
- Whom to alert
- The severity of the alert

The centralized monitoring platform must be sufficiently accessible and customizable so that all systems and services can be properly monitored. The service owners configure the monitoring and alerting in a way that makes sense for their systems and services. For example, the DNS team would want to ensure that its DNS servers are responding correctly to queries. A team with a web-based service might want to monitor the load time for various pages.

36.3.8 Logging

Centralized logging enables correlation between events so that the root cause of a problem can be identified more easily. A centralized logging service should have dedicated high-end storage with lots of capacity and an automated archiving and retrieval system for older data. It needs to provide an interface that enables SAs to retrieve and scan the logs for their systems to debug problems. It should enable data correlation to find trends or problems that extend beyond a

single system.

A centralized logging facility requires compartmentalized security so that access to the logs (or individual fields) can be controlled. Otherwise, the privacy implications will prevent the service from being adopted.

36.4 Support

The centralization of support services is an issue that is as important as the infrastructure and architecture issues discussed so far. Our support infrastructure is often the only way we are visible to our customers. Therefore, these decisions affect how our customers see us.

Centralization does not mean that the entire IT team needs to be in a single building, country, or region. The support staff for a geographically diverse company needs to be geographically diverse as well. However, regional staff should report into a unified global IT organization under the CIO.

If the organization consists of several different business units, perhaps as a result of acquisitions, there should be criteria for how and when the different units are connected. For companies that frequently acquire other companies, it is important to have a process for integrating both the network and the support staff into the central structure.

No matter which level of centralization is in effect, customers expect to receive the same treatment: Everyone should live under the same policies, enforced to the same degree, and everyone should receive the same quality of support. If one division has demonstrably better support, or receives special treatment, or has IT policies that are different, it will be perceived (rightfully so) as unfair.

Our recommendations for specific services are summarized in [Table 36.3](#).

Organizational Component	Recommended Approach
Helpdesk	Strictly centralized service
End-user support	Centralized with decentralized delegation
Server support	Centralized with decentralized delegation
Infrastructure support	Strictly centralized service
Network operations	Centralized with decentralized delegation
Security operations	Strictly centralized service

Table 36.3: Centralization for Organizational Support

36.4.1 Helpdesk

All companies should have a single centralized helpdesk that all end users contact when they need support. A central helpdesk is a single point of contact for the end user whether he or she is working in the office, at home, or on the road. The central helpdesk can spot problems that span across multiple locations.

The helpdesk is also a single point of contact for other SA groups as they roll out new versions of their OS, software, or service. SAs can notify the helpdesk and ask its staff to watch out for problems and escalate those problems more quickly after an upgrade.

The central helpdesk also builds up a single knowledge base that is more comprehensive than multiple knowledge bases developed by diverse teams, ensuring consistent support across locations and business units. The central helpdesk is also a single point that needs to be updated when reorganizations cause the escalation points of contact to change.

36.4.2 End-User Support

End-user satisfaction surveys typically show that customer satisfaction is strongly linked to having a local support presence. Customers like to be able to ask a quick question and get an answer, without having to go through the helpdesk for everything. For example, someone might ask a passing SA, “Do you think my disk sounds sick? I think it is noisier than it used to be.” Customers may not want to open a ticket for something that they are not sure is a problem. However, having on-site staff at every location can be expensive for companies with many small offices, and it is harder to keep isolated support staff in the loop.

Companies need to find a balance between these competing influences. Some common models include having someone from end-user support on-site on particular days. For example, someone might be in the main office with the rest of the team on Mondays, then at one remote office on Tuesdays and Thursdays, and at another remote office on Wednesdays and Fridays. Smaller offices may get a visit once every two weeks, or just on an as-needed basis.

From an organizational point of view, the end-user support teams need to report into a central end-user support organization that is a part of the central IT support team. They need to be kept in the loop about changes that may affect their customers, and they need to be able to feed their customers’ needs back to the other teams.

36.5 Purchasing

While it often is not strictly part of the IT department, the purchasing department plays a key role in the IT infrastructure. Purchasing should be a strictly centralized service. Centralizing purchasing allows a company to negotiate better deals with vendors, thereby saving money on both the initial purchase and the maintenance contracts. It also enables a company to develop hardware standards, by purchasing only a limited number of standard hardware configurations from each vendor. This enables the company to stock spare parts in several locations, which further reduces the price of hardware maintenance and eliminates long turnaround times with the vendor. In places where the vendor does not have a depot, stocking spare parts can also significantly reduce repair times.

Even for relatively low-volume purchases from a vendor, the purchasing department may be able to get a better deal by going through a third-party reseller that does a lot of business with both that vendor and your company. However, if purchasing is done by smaller independent groups, then these savings opportunities will be missed. This department is sometimes outsourced to another company that specializes in purchasing.

36.6 Lab Environments

For lab environments, we recommend a strictly centralized lab for integration testing, and smaller, decentralized labs for individual teams. The purpose of the centralized lab is to test and confirm that new hardware, software, and configurations will work as expected in production, with all device types. Each lab environment should be firewalled off from the production environment so that it can't inadvertently interfere with production services.

The centralized lab should have some of everything that is in the production environment, including LAN, WAN, wireless, remote access, all security devices, authentication services, DNS, DHCP, LDAP, voice, multimedia, workstations, different server types, an AD domain, a Kerberos domain, storage, and so on. It should be set up as a mini-production network.

One of the steps for each group, before deploying something into production, is to deploy it into this lab, test against the other devices there, and have all the other teams perform testing against it. For potentially disruptive testing, it is important to have a scheduling system in place, so that the disruptive testing does not ruin someone else's nondisruptive tests. Each team is responsible for keeping its lab configuration current—that is, in sync with production.

36.7 Summary

In this chapter we identified many common IT functions and discussed the benefits of operating them in a centralized fashion.

Security-related services such as policies, Internet connectivity, authorization, and data leakage protection are best done in a centralized manner because they require a high degree of specialized knowledge, and practices must be uniformly enforced to be effective. A chain is only as strong as its weakest link.

Infrastructure components such as networking, datacenters, and IP address management can benefit from centralization due to their size. Volume discounts and standardized operational procedures save money and increase quality. Effort directed toward improving the system is amortized over all of its uses.

Commodity services such as email, end-user desktop support, and purchasing are best centralized because they are needed by everyone in the company and must interoperate across the company. A company shouldn't have one division with a better email system than the other divisions.

Backups, monitoring, and in-person helpdesk support are best organized in a distributed model because of the geographic limits that are inherent to these services.

Exercises

1. Which services are recommended for centralization?
2. Which services are recommended for decentralization?
3. Which services are recommended to be operated in a distributed model?
4. List four services or roles in your organization and categorize them by their service models.
5. Which services or roles in your organization would benefit from a change from one model to another?

Chapter 37. Centralizing a Service

This chapter is about the process of taking an existing decentralized service and centralizing it.

We are proponents of centralization. However, most large sites, and many midsize sites, have a number of services that are decentralized. In some cases, this is historical: There was no strong policy for centralization, and nobody was allocated to providing and supporting some of these central services. In other cases, this model arose because the people who wanted to build the decentralized service convinced management that they had a real business need to do so.

In any case, we often find ourselves in an environment where there are some decentralized services. If they are infrastructure services, or any of the services that we described in the previous chapter, they are candidates for centralization. If several different groups have built similar services, they are also candidates for centralization. Conversely, if the service is really specific to one particular group and is unlikely to ever be needed by other groups, it is not a candidate for centralization.

When you have identified a service that is a good candidate for centralization, take the following steps:

1. Understand the current solution.
2. Make a detailed plan for building and migrating to a centralized service.
3. Get management support and, if necessary, a budget.
4. Fix the problems that led to decentralization.
5. Provide an excellent centralized service.
6. Start slowly and ramp up.
7. Look for low-hanging fruit.

37.1 Understand the Current Solution

The first phase of centralizing a decentralized service involves understanding the current solution: its features, its strong and weak points, its associated change processes, its users and how they use it, and the reasons the service is currently decentralized. It is important to understand the existing situation thoroughly before making the change, because otherwise you run the risk of making the service worse. You will get only one chance at this. If you fail, that failure will be used as a reason to not move to a centralized service forevermore.

It is also important to understand the reason that the service is decentralized before you centralize it. It may be due to some technical, procedural, or SLA problems with the central service, which you need to fix, or it may be historical, such as a merger or the lack of a central service when the group acquired the service. The reasons for decentralization may vary between the different groups running the decentralized services. Their customers and management may have their own reasons for decentralization. It is important to understand all these views.

Devise a short questionnaire that is used for interviewing all the stakeholders. That way, you can stay on topic and make sure that you don't forget to ask a key question because the conversation went in an unexpected direction. The questions that you ask should be neutral, not judgmental. The purpose of the questionnaire is to gather the facts that you need to make the central service better than the decentralized one. Use the following list of questions as a starting point for your questionnaire:

- How do you request configuration changes to the service? Is there an approval process?
- How long does it take before configuration changes are implemented? Is there an SLA for how long requests should take?
- Which automation interacts with this service? Who is the best person to talk to about the automation?
- Which manual processes use this service? Who is the best contact for understanding those processes?
- How is this service accessed by end users? Do they need to authenticate? Is end-user access generally available or limited in some way (how and why)?
- Which levels of administrative access are available for this service (e.g., reader, writer, superuser), and which access control features are used?
- What is the process for requesting administrative access? Which approvals are required?
- How are administrative users authenticated?
- How do you request feature changes for the service?
- What is the process for accepting, rejecting, and prioritizing feature change requests?
- What is the typical turnaround time for feature change requests?
- How are feature changes tested and released into production? How

frequently does this happen?

- Which features of the service are important to you?
- What do you like about the service?
- What improvements would you like to see in this service?
- Who are the customers of this service?
- How is the service supported?
- How is the service monitored?
- Which metrics do you use to measure service availability?
- How reliable is the service? What is the SLA?
- What are the most common problems or causes of failure?
- What is the history that led to this service being run by this group?

Do not be judgmental or defensive when you hear the answers: Ask the questions, listen to the answers, write them down, and thank the person. The aim in this phase is simply to gather the data, not to defend your group or to point out better solutions or errors in the respondent's reasoning.

37.2 Make a Detailed Plan

Once you understand the current solution(s), the next step is to define what, if anything, needs to change in the centralized service to meet the needs and expectations of the customers of the decentralized services. Then you need to build a detailed plan of all the work that needs to be done, who needs to be involved, and approximately how much of each group's time it will take. Consider not just the work on the centralized service itself, but any automation or manual processes that need to be changed to work with the centralized service.

Consider also the effort and teams who need to be involved in migrating the decentralized services to the centralized service. Migrations may involve teams other than those who are running the respective services. For example, DHCP migrations would also involve the network teams who need to make DHCP-relay changes on the routers, and perhaps the desktop support teams who might want to confirm that everything is still working on the office networks. Teams who support automated systems that interact with this service may need to make configuration or software changes in tandem with the migrations. Change-request tools may need to be switched on migration day, and outstanding tickets migrated from one queue to another. Use the responses to the questionnaires to figure out who you need to talk to and what else you need to consider.

You also need to look at what needs to be communicated to end users. Do they need training on the new system or processes? Do they need to be informed of a change in the support model? Which changes need to be made to monitoring systems, to documentation, and to incident escalation processes?

37.3 Get Management Support

At most companies, before you can centralize services, you will need management support. The managers you talk to will want to know the costs involved and how long it will take. They will want to know that you have thought through the proposal and considered all aspects of the migration, not simply the technical aspects. Therefore it is best to first make a detailed plan before you look for management support.

Management support is essential for pushing through a potentially unpopular change. Groups that have their own decentralized services often do not want to migrate to a centralized one. You will also need management support and a budget for building or improving the centralized service, and for devoting some resources to maintaining it afterward.

Explain the benefits of centralization in terms of how this change will benefit the business and the users. Present your detailed plan, timeline, and cost and resource estimates. In particular, estimate the cost savings and the anticipated service improvements.

If managers buy into the centralization plan, it is likely that they will need to take it higher up the management chain to get the support of a senior manager, who can then direct the other groups to support and participate in the migration efforts. The more ammunition that you can give your management to use in these conversations, the better your chances are for success.

37.4 Fix the Problems

Once you have management support to proceed with the centralization project, the first thing that you need to do is fix any problems that have been identified with the current service. These may be known problems for a service that is in active use. These problems may be why other groups have built their own decentralized services.

Look at your existing list of known issues and anything that you learned during the questionnaire phase to identify and prioritize problem areas. There is no point in trying to migrate people to a centralized service when you know that key features are missing, or stability or scaling problems are present.

37.5 Provide an Excellent Service

It is important that the new service is significantly better than the service it replaces in ways that are tangible to the direct users. Management may already prefer the new solution because of costs or efficiency objectives. But if it isn't significantly better to the direct users, we as SAs haven't done our job.

Make the centralized service fast, highly reliable, full-featured, automated, and customizable. Load test it before migrating anyone to it. Monitor resource usage, understand how it scales, and know which resources are your limiting factors. Be sure that it will scale to the level that you need in the time frame that you need, and still perform well.

A centralized service should be cheaper to run and better supported than a series of decentralized ones. If the centralized service is really good, and in particular, better than the decentralized services, then most people will want to migrate to it, making the job of centralizing significantly easier.

Also, remember that you have only one chance to make a good first impression. This rule applies whether this is a brand-new service that no one has used, or you are migrating people from an existing decentralized service to a centralized one. People's opinions are always colored by their first impressions, so make it a good one.

Don't move anyone onto the new platform until it has been extensively tested. Write automated test suites to test every feature of the new service, and to load test it. The rewards are worth the effort. Aside from helping you make that all-important good first impression, you will be able to reuse these tests to validate new releases before upgrading the system in the future.

If the new service involves some software or automation systems that are written in-house, use test-driven development (TDD), which was discussed in [Section 8.3](#) and is described in more detail in Volume 2 of this book series.

If the new service is really good, then others will hear about it, and be more inclined to use it. If it is not up to expectations, then the service will get bad press, and it will be much harder to persuade people to start using it, even after you have fixed all the problems.

37.6 Start Slowly

Start migrating users onto the new service slowly. This is particularly important for a brand-new service. You do not want to have a big, visible outage on the first day. Start with those users who will be most tolerant of failures. Your own and other SA teams are a good place to start.

Take some time at the beginning to get experience with the new service with a friendly audience. Find and fix any bugs or performance issues. Continue to migrate groups, gradually increasing the size of the groups as you build confidence in the process and the service. Pay attention to utilization and performance data as you add load, so that you can have confidence that the service is scaling as expected.

37.7 Look for Low-Hanging Fruit

When migrating people to using a centralized service, start by migrating the people who want to migrate, or those who don't currently have this service but want it. Those who are resistant to being migrated can be late adopters (see the anecdote in [Section 11.2](#)).

When migrating people from a decentralized service to a centralized one, start with the simplest case: the users who do not have complicated requirements. Find out what their biggest complaint about the old system is, and make sure the new system excels in that area.

Look for other opportunities to align your migrations with the customers' needs. For example, if the machines being used by decentralized services are nearing their end of life (EOL), then migrating that group to the centralized service will yield a clear savings over having to replace the EOL equipment.

Centralizing DNS and DHCP

One large company had a variety of decentralized DNS and DHCP systems, but wanted to centralize these services. Some of the decentralization was along business-unit lines, and some was along technology lines—DNS for Windows ActiveDirectory (AD) domains was provided by the Windows AD domain controllers (DCs).

The team working to centralize DNS/DHCP started with the business unit whose solution was reaching its EOL first, and migrated each of the large business units in turn as their equipment neared its EOL. At that point the team had about 80 percent of the services on the central platform. The more complicated migrations, and those where there was the most resistance, were left for last. At that point, the new centralized service was well established as the strategic direction, and other groups were more willing to work with the central team to ensure that all their needs were met.

37.8 When to Decentralize

For all that we advocate a centralized approach, we sometimes find ourselves in a situation where we end up building a decentralized service for ourselves, because the centralized one does not meet our needs.

Whenever you find yourself in a situation where you are seriously considering building your own decentralized service, first try really hard to work with the group providing the centralized service. Provide them with a clear list of high-level requirements, and justification for those requirements. Give them some time to come back to you and let you know if your needs will be met in the short, mid or long term, or not at all. Then decide what to do.

Decentralized Monitoring

Christine once worked in a very large company where there was a centralized monitoring service, which was linked into the trouble-ticket system. What was monitored and how were opaque to the service owners, and the only output that they saw was in the form of incident tickets.

The monitoring team had no mandate, and no funding, to perform any kind of historical monitoring. Therefore there was no data to use for capacity planning. Teams using this monitoring system could only be reactive, not proactive, which is far from ideal when it can take months to get additional capacity funded, ordered, and delivered.

In addition, the monitoring system used by the centralized team had very limited methods it could use, and was not extensible. Christine wanted the monitoring system to not just ping and SNMP query her systems, but also monitor the response time of queries to the service itself.

She built a list of her requirements, and shared it with the team leader. Unfortunately, she was told that the centralized team could not meet her needs, as they had a limited budget and insufficient staff.

As a result, Christine ended up building her own monitoring system that other teams in her area also used. It was not a production-quality system, but it was a lot better than nothing, and proved invaluable for early detection of problems, outage prevention, capacity planning, and debugging complex incidents.

Christine stayed in touch with the people responsible for the monitoring tools, and was the first to try out the new tool in the lab, when they finally were able to get one.

37.9 Managing Decentralized Services

Sometimes we find ourselves in a situation where we need to work with decentralized services. In this situation it is important to define clear boundaries between the services that are run by different groups, so that problems in one service cannot propagate into others. The most obvious reason for doing so is damage limitation. If there are not explicit interfaces between the different administrative domains, then they can all share the same fate when a problem in one area affects the others. Perhaps less obvious is the point that when there is an outage in one part of the service, it needs to be clear who owns the problem, so that the correct support and escalation paths can be followed. Otherwise, the outage will last a lot longer than it should, while people try to figure out who needs to be involved.

Use existing technologies and protocols to define clear boundaries and interfaces. For example, if different parts of the network are managed by different groups, define clear boundaries with Border Gateway Protocol (BGP), where you each specify which networks you will advertise to the others, and which network advertisements you will accept from the others. Then, if one group accidentally starts advertising another network in its area, that network advertisement will not leak into other areas. The damage will be limited to one area, and ownership of the problem and resolution will be clear.

Decentralized Network Management

At one large multinational company, the router configurations were managed by regional teams. They did not use a routing protocol, such as BGP, to limit propagation of routes between regions.

One weekend, the network team in South America made a change in one of their routers, putting in a static route to a firewall for a particular network. However, that network was already in use in Europe. The network that was defined in South America was smaller than the one in Europe, and was propagated all over the world. The routers treated this route as preferred, because it was “more specific.”

The computers in Europe that were on this portion of the network became completely nonfunctional, because all traffic that was intended for them—even from within the country—was routed to the firewall in South America. The situation was even more confusing for the local network team, because other computers on the same network were operating normally. It took quite some time to find this more specific route, and then

track down its origin and get it removed.

If the boundaries between the administrative domains had been codified with BGP, the effect on the end users would have been very limited, and it would have been easier to track down the problem as being related to something in South America.

Security is another example. If different parts of the company have different security policies and solutions, then it is in the best interest of both parts to isolate themselves from the other part and treat the interconnections like any other link to a business partner. Each side should use its own approach to third-party connectivity to secure the connection. Then each part of the company can be sure that its environment is protected according to the security design and policies, and the company as a whole is better protected.

One example where this kind of a segregation may or may not be possible is with namespaces. Namespaces include machine and domain naming, user and group identities, asset tags, software package naming, and directory names. For anything that is used as a key in the inventory system, it is possible to have a non-centralized namespace only if there are two completely separate inventory systems that are used in different business units, and there are no inventory items that are shared between these business units or used in both locations.

If you find yourself in a situation where there are decentralized services and it is not possible to leverage technologies to define clear boundaries, you should push hard for centralization. Explain the problems with the decentralized model in terms of the operational risks, and ideally the potential costs to the company due to outages and difficulty in diagnosing and debugging problems. Each time there is an incident where debugging and resolving the problem is hampered by a decentralized service with no clear boundaries, document the details in the post-incident report, and provide these details when you recommend that the service be centralized.

37.10 Summary

Centralizing a previously decentralized service is an important but difficult process.

The first step is to understand the current service. What does it offer, what do people like about it, and what are the requirements for the new service? Build a detailed plan for the service and how the transition will be performed. Most importantly, identify what the new system will fix. This is the key to getting buy-

in from management and users alike.

You have only one chance to make a good first impression. So make sure the new service is significantly better than the old one, and assure that the service people receive during the first few weeks is particularly good.

Roll out the new solution a little at a time. Do not do a flash-cut or overnight change for everyone. A slow roll-out permits you to find and fix problems before they impact all users.

Exercises

1. What are the steps in centralizing a service?
2. Which steps are key to a centralization project being a success?
3. Pick a service provided in your organization and identify how you would centralize it using the advice in this chapter.
4. Previous chapters discussed resistance to change. Combine the advice from [Section 11.1](#) and this chapter to propose how you might reduce customer resistance when centralizing a service.

Part VIII: Service Recommendations

Chapter 38. Service Monitoring

This chapter is about monitoring the services and networks in your environment. Monitoring is the primary way we gain visibility into the systems we run. It is the process of observing information about the state of things for use in decision making. The operational goal of monitoring is to detect the precursors of outages so they can be fixed before they become actual outages, to collect information that aids decision making in the future, and, of course, to detect actual outages. The ideal monitoring system makes the operations team omniscient and omnipresent.

There is an axiom in the business world that if you can't measure it, you can't manage it. This holds true in system administration, too, and we use monitoring to do the measurement. In fact, the goal of monitoring is to make our systems observable. You can't manage what you can't see.

Monitoring is an important component of providing a reliable, professional service. As discussed in [Section 17.6.1](#), monitoring is a basic component of building a service and meeting its expected service levels. It isn't a service if it isn't monitored. If there is no monitoring then you're just running software.

This chapter covers the following topics:

- **The two types of monitoring:** Real-time and historical
- **Building a monitoring system:** Some of the pitfalls to watch out for
- **Historical monitoring:** Gathering, storing, and viewing the data
- **Real-time monitoring:** Problem detection and alerting
- **Scaling challenges:** Prioritization, cascading alerts, and coordination
- **Centralization:** Building a shared platform
- **Advanced monitoring:** End-to-end tests, application response times, compliance monitoring, and meta-monitoring

Good monitoring helps us foresee trouble before it turns into an outage. Rather than monitoring whether a service is up or down, we can detect when it becomes sick and fix it before the problem escalates into a customer-visible issue. For example, rather than detecting that a disk is full, we can monitor that a disk is 80 percent full and increasing. Now we have time to avert the problem rather than just respond to it.

Monitoring is even more critical where high uptime is required, such as with e-commerce web sites and emergency services. However, it is also important in

everyday enterprise applications that are expected to be available just Monday through Friday during business hours. Without monitoring, if a service crashes over the weekend, we may arrive on Monday morning to find that our co-workers are upset that they haven't been able to get any work done. With monitoring, we can fix the problem on Sunday and instead receive kudos for being proactive.

38.1 Types of Monitoring

Systems monitoring can be used to detect and fix problems, identify the source of problems, predict and avoid future problems, and provide data on SAs' achievements. The two primary ways to monitor systems are to gather historical data related to availability and usage and to perform real-time monitoring to ensure that SAs are notified of failures that aren't fixed automatically.

Historical monitoring is used for recording long-term uptime, usage, and performance statistics. It has two components: collecting the data and viewing the data. The results of historical monitoring are conclusions: "The web service was up 99.99 percent of the time last year, an increase from the previous year's 99.9 percent statistic." Utilization data is used for capacity planning. For example, you might view a graph of bandwidth utilization gathered for the past year for an Internet connection. The graph might visually depict a growth rate indicating that the pipe will be full in four months. Bosun, Prometheus, Munin, and Ganglia are commonly used historical monitoring tools.

Real-time monitoring alerts the SA team of a failure as soon as it happens. It has two basic components: a monitoring component that notices failures and an alerting component that alerts someone to the failure. More advanced monitoring systems have a third component, which fixes some of the detected problems. There is no point in a system's knowing that something has gone down unless it alerts someone to the problem. Even if the system fixes the problem, that fact should be tracked, as it may indicate a deeper issue, such as a software bug that needs to be fixed. The goal is for the SA team to notice outages before customers do. This results in shorter outages and problems being fixed before customers notice, along with building the team's reputation for maintaining high-quality service. Nagios and Big Brother are commonly used real-time monitoring systems.

38.2 Building a Monitoring System

Typically, historical and real-time monitoring are performed by different systems. The tasks involved in each type of monitoring are very different. After

reading this chapter, you should have a good idea of how they differ and know what to look for in the software that you choose for each task.

But first a few words of warning. Monitoring uses network bandwidth, so make sure that it doesn't use too much. Monitoring also uses CPU and memory resources, and you don't want your monitoring to make your service perform poorly. Security is important for monitoring systems because they generally have more access to other systems than most other servers do. For example, there may be firewall rules to let the monitoring system reach devices in other security zones, and they may contain credentials that need to be used for some of the monitoring, such as SNMP queries. Greater access can be leveraged by an attacker to compromise other systems or to initiate a denial-of-service attack. Strong authentication between the server and the client is best. Older monitoring protocols, such as SNMPv1, have weak authentication.

Within a local area network, the network bandwidth used by monitoring is not usually a significant percentage of the total available network bandwidth. However, over low-bandwidth—usually long-distance—connections, monitoring can choke links, causing performance to suffer for other applications. Make sure that you know how much bandwidth your monitoring system is using. A rule of thumb is that it should not exceed 1 percent of the available bandwidth. Try to optimize your monitoring system so that it goes easy on low-bandwidth connections. Consider putting monitoring stations at the remote locations with a small amount of communication back to the main site.

Another option is to use primarily an SNMP trap-based system, whereby the devices notify the monitoring system when there is a failure, rather than a polling-based system, whereby the monitoring system checks status periodically. Since correctly configured traps occur much less often than the polling frequency, this will reduce your bandwidth consumption. However, an SNMP trap is a single UDP packet, so it is not guaranteed to be delivered, and could be dropped due to network congestion or a brief network glitch.

Under normal circumstances, a reasonable monitoring system will not consume enough CPU and memory to be noticed. However, you should test for failure modes. What happens if the monitoring server (the machine that all the servers report to) is down or nonfunctional? What happens if a network outage occurs?

38.3 Historical Monitoring

Polling systems at predefined intervals can be used to gather utilization or other

statistical data from various components of the system and to check how well system-provided services are working. The information gathered through such historical data collection is stored and typically used to produce graphs of the system's performance over time or to detect or isolate minor problems that occurred in the past. In an environment with written SLA policies, historical monitoring is the method used to monitor SLA conformance.

Historical data collection is often introduced at a site because the SAs want to know if they need to upgrade a network, add more memory to a server, or get more CPU power. They might be wondering when they will need to order more disks for a group that consumes space rapidly or when they will need to add capacity to the backup system. To answer these questions, the SAs realize that they need to monitor the systems in question and gather utilization data over a period of time to see the trends and the peaks in usage. There are many other uses for historical data, such as usage-based billing, anomaly detection, and presenting data to the customer base or management (see [Chapter 49](#), "[Perception and Visibility](#)").

38.3.1 Gathering the Data

How you intend to use the data that you gather from the historical monitoring will help to determine which level of detail you need to keep and for how long. For example, if you are using the data for usage-based billing and you bill monthly, you will want to keep complete details for a few years, in case there is a customer complaint. You may then archive the data and expire the online detailed data but save the graphs to provide online access for your customers to reference. Alternatively, if you are simply using the graphs internally for observing trends and predicting capacity needs, you might want a system that keeps complete data for the past 48 hours, reasonably detailed information for the past 2 weeks, somewhat less detailed information for the past 2 months, and very condensed data for the previous 2 years, with everything older than 2 years being discarded. Consider how you will use the data and how much space you can use when deciding on how much to condense the data. Ideally, the amount of condensing that the system does and the expiration time of the data should be configurable.

You also need to consider how the monitoring system gathers its data. Typically, a system that performs historical data collection will want to poll the systems that it monitors at regular intervals. Ideally, the polling interval should be configurable. The polling mechanism should be able to use many standard forms of communication. For example, it should be able to use a standard query

mechanism, such as SNMPv2, that many systems provide for exactly this purpose. It should be able to do basic checks using ping (ICMP) packets. It should be able to perform more specific application checks by connecting to the service and simulating a client. It is also useful to have a monitoring system that records latency information, or how long a transaction took. The latency correlates well to the end users' experiences. Having a service that responds very slowly is practically the same as having one that doesn't respond at all.

The monitoring system should support as many other polling mechanisms as possible, preferably incorporating a mechanism to feed in data from any source and parse the results from that query. The ability to add your own tests is important, especially in highly customized environments. At the same time, having a multitude of predefined tests is valuable, so that you do not need to write everything from scratch.

Many services include a URL that can be queried to receive a list of statistics and monitoring-related information. Typically this data is provided in JSON format. A monitoring system can query this URL, record the items it wants, and discard the rest. This is becoming the universal format for monitoring web-based application services.

38.3.2 Storing the Data

Historical data can consume a lot of disk space. This potential problem can be mitigated by data summarization or expiration.

Summarization means reducing the granularity of data. For example, you might collect bandwidth utilization data for a link every five minutes. By comparison, retaining only hourly averages, minima, and maxima might require 75 percent less storage. It is common to store the full detail for the past week but to reduce the information down to key hourly data points for older data.

Expiring data simply means deleting older data. You, for example, might decide that data older than two years does not need to be retained at all, or can be archived to removable media—DVD or tape—in case it is ever needed.

Limiting disk space consumption by condensing the data or expiring it affects the level of detail or historical perspective you can provide. Bear this tradeoff in mind as you look for a system for your historical data collection.

In the 1990s, such decisions were much more important because disk space was so much more expensive. With data storage being relatively inexpensive now, many sites simply design their monitoring systems to retain data at full granularity forever, which in business is often three to five years.

38.3.3 Viewing the Data

The output that you generally want from this type of monitoring system is graphs that have clear units along each axis. You can use the graphs to spot trends. Sudden, unexpected peaks or drops are often an indication of trouble. You can use the graphs to predict when you need to add capacity of any sort and as an aid in the budget process. A graph is also a convenient form of documentation to pass up the management chain. A graph clearly illustrates your point, and your managers will appreciate your having solid data to support your request for more bandwidth, memory, disk space, or whatever it is that you need.

38.4 Real-Time Monitoring

A real-time monitoring system tells you when a host is down, a service is not responding, or some other problem has arisen. A real-time monitoring system should be able to monitor everything you can think of that can indicate a problem. It should be able both to poll systems and applications for status and to receive alerts directly from those systems if they detect a problem at any time. As with historical monitoring, the system should be able to use standard mechanisms, such as SNMP polling, SNMP traps, ICMP pings, UDP, and TCP, as well as provide a mechanism for incorporating other forms of monitoring. A real-time monitoring system should also be capable of receiving and processing log messages from any system that has a remote logging capability, and generating alerts based on those logs.

In addition, such a system also should be capable of sending alerts to multiple recipients, using a variety of mechanisms, such as email, paging, telephone, and opening trouble tickets. Alerts should go to multiple recipients, because an alert sent to one person could fail if that person's pager or phone has died or if the person is busy or distracted with something else.

The storage requirements of a real-time monitoring system are minimal. Usually, it stores the previous result of each query and the length of time since the last status change. Sometimes, the system stores running averages or high and low watermarks, but it rarely stores more than that. Unlike historical monitoring, which is used for proactive system administration, real-time monitoring is used to improve reactive system administration.

When evaluating a monitoring system, look at the things that it can monitor natively to see how well it matches your needs. You should consider monitoring both availability and capacity. **Availability monitoring** means detecting failures of hosts, applications, network devices, other devices, network interfaces, or

connections of any kind. **Capacity monitoring** means detecting when some component of your infrastructure becomes, or is about to become, overloaded. For example, that component could be the CPU, memory, disk space, swap, backup device, network or other data connection, remote access device, number of processes, available ports, application limitations, or number of users on a system.

The monitoring system should also be able to do **flap detection**. When a monitored value frequently moves from a good state to a bad state and back again, then it is said to be flapping. This is a third state that a service can be in. Rather than registering and clearing alerts with each state change, the monitoring system should detect the continued state changes and mark the service as flapping, and generate an alert indicating that the service is in this state.

Like historical monitoring systems, the real-time monitoring system needs to be flexible and to permit the creation of your own test modules. Preferably, a system should be able to use the same modules for both real-time and historical monitoring.

The most important components of a real-time monitoring system are the notification mechanism and the processes your site puts in place for dealing with the notifications or alerts.

38.4.1 SNMP

SNMP stands for Simple Network Monitoring Protocol. No one is really sure whether the *simple* refers to *network* or to *protocol*. Problems with SNMP make it difficult to use on larger-than-simple networks. Although it attempted to be simple, the protocol itself is rather complex.

In SNMP's most basic form, a packet is sent to a network device, such as a router, with a question called a **GET**. For example, one might ask, "What is the value of **IF-MIB::ifOutOctets.1**?" That variable is in the group of interface-related (**IF**) variables, the one that records how many bytes (octets) were sent out of the interface (**ifOutOctets**), on interface number 1. The router replies with a packet containing the value.

A monitoring system can actively query a device using **SNMP GET**. It can generate alerts based on the values it receives in the response. For example, it could detect that CPU usage has exceeded a threshold value, or that network traffic on a given interface has dropped below an expected value.

A monitoring system can also be configured to receive and process **SNMP traps** that are sent by the devices that are being monitored. A system generates

SNMP traps when certain events occur, such as if it detects disk, fan, or power problems, or a threshold is exceeded—for example, 95 percent usage of IP addresses in an IPv4 DHCP range. SNMP traps have the advantage that the monitoring system is alerted immediately when an event occurs, as opposed to having to wait for the next round of active polling before noticing the problem.

SNMP traps are usually state-change events. A trap is sent when a network interface goes down, and another is sent when a network interface comes up. A monitoring system should generate alerts based only on “down” state-change events, not on “up” events. It should be able to correlate those events, and provide a configurable delay to wait for the corresponding up event before generating an alert on a down event. It should also be able to cancel alerts that have been already generated on the down event, when the corresponding up event arrives.

Note that because traps are typically sent based on a state change, a system usually sends only a single SNMP trap when the event occurs. It does not continue to send periodic traps indicating that the problem still exists. If the trap is not received for any reason, such as the UDP packet being dropped, the alerting system will not be notified of the event. To mitigate this problem, SNMP traps can be sent to multiple receivers, and monitoring should not rely on SNMP traps alone, but rather on a combination of traps and active polling. SNMP also provides the option for the SNMP trap sender to send an SNMP Inform request, which asks the trap receiver to acknowledge the receipt of the trap.

There are SNMP variables for just about everything, and every kind of technology. A group of related variables is called a MIB. There are standard MIBs for Ethernet devices, DSL devices, ATM devices, SONET devices, T1/E1 devices, and even non-network technologies—disks, printers, CPUs, processes, and so on.

SNMP’s simplicity, however, is also the source of its major disadvantage: One variable is requested in each packet. So, for example, if you want to monitor five variables for each interface, and the system has dozens of interfaces and hundreds of devices, you are soon sending a lot of packets! SNMP-based monitoring systems use complicated schemes to gather all the information they are trying to collect in a timely basis. When some devices are far away and latency is high, waiting for each reply before sending the next request drags down performance. Later versions of SMNP support requesting multiple variables in one packet, but this feature is not always supported.

SNMP is also associated with security problems. Devices that support SNMP

require a password—called a **community string** for some reason—in the packet to prevent just anyone from gathering data. GET defaults to the password `public`, and PUT defaults to the password `private`. Luckily, most vendors do not provide sensitive data in variables that GET can read, and they disable PUT completely. SNMP versions 2 and 3 encrypt the password, which is an improvement. However, having the same password for many devices is not very secure. If you had to change the SNMP password on every router in your enterprise every time an SA leaves the company, a large company would always be changing these passwords.

Most devices can be configured to permit SNMP requests to come only from specific IP addresses. It's best for sites to designate a few specific IP addresses ranges that will house any SNMP servers and to configure all SNMP-capable devices to respond only to devices in those ranges.

The data in an SNMP packet is encoded in a format called ASN.1. This format is very complicated and difficult to implement. In 2002, there was a big scare when it was discovered that many of the routers on the Internet had a security flaw in their ASN.1 decoding software. In some cases, the security flaw could be triggered as part of the password verification, so it didn't matter whether an attacker knew your SNMP community strings.

The situation has since improved greatly. However, consider the following six recommendations for all SNMP-capable equipment:

- Treat community strings the same way as passwords. Change them frequently, do not use vendor-default strings, and do not send them unencrypted over the network.
- Configure network equipment to process SNMP packets only from particular IP ranges. Two ranges, one for each redundant network monitoring center, is reasonable for many sites. Larger sites may need two IP ranges per region.
- Use SNMPv3 when the vendor supports it, and enable encryption.
- Change SNMP community strings at least once a year. Have a transition period during which the old and new community strings are accepted.
- Automate a weekly audit of all network devices. From one of the approved network ranges, attempt to use all the previous SNMP community strings that should have been eliminated. Also attempt to use vendor-default community strings such as `public` and `private`. With a few web searches, you can easily find a list of default vendor network device passwords. Try them all. Now repeat this test from outside the designated

network ranges.

- Track vendor security bulletins related to SNMP closely. New vulnerabilities appear a few times a year.

38.4.2 Log Processing

Most systems are capable of sending log messages to a remote logging server. For systems that do not generate SNMP traps, processing logs in real time is another way to detect events more quickly than a polling-based system can. This technique can also detect events that a polling-based system might miss, such as a fan failure or disk errors.

An alerting system should treat logs the same way as it does other data sources. It should store the previous state, so that it can detect state changes, and the time of the last state change, so that it can do duration-based escalations if necessary. Logs are useful for debugging but take up a lot of storage space. Logs for all systems should go to dedicated log servers, which should have sufficient storage for them.

The log servers should also process the log files from many different systems to detect related events, and generate alerts based on those events. Software such as Splunk and Logstash can be used to build this functionality.

38.4.3 Alerting Mechanism

A real-time monitoring system has an alerting mechanism to make people aware that something requires attention. There is no point in software knowing that something has failed or is overloaded unless it tells someone about it or does something about it and makes a note of it for an SA to look at later.

A monitoring system's alerting mechanism should not depend on any components of the system that is being monitored. If any given failure causes the monitoring system to be unable to report the failure, it needs to be redesigned. Email and trouble-ticket systems are popular alerting mechanisms, but should not be the only options. Email can fail and can have long delays. Trouble-ticket systems are complex and rely on other infrastructure, which could fail. Alerting for critical problems needs to happen quickly. Also consider whether the alerting mechanism can be intercepted by third parties. Wireless communication, such as pages and text messages, is susceptible to third-party spying. At a minimum, do not send sensitive information, such as passwords, over these channels, and consider whether such information as `backbone-router is down for 45 minutes` is proprietary.

Alerting Policy

Whatever monitoring system you use, you need to have a policy that describes how the alerts are handled. The policy needs to answer some fundamental questions before you can implement real-time monitoring. How many people do the alerts go to? Do alerts go to the helpdesk or to the individuals who look after the components that are having problems, or some combination of the two? How do the recipients of the alerts coordinate their work so that everyone knows who is handling each problem and they don't get in one another's way? If a problem persists beyond some predetermined length of time, do you want to escalate the problem? If so, what is the escalation path? How often do you want to be informed of a problem, and does that depend on what the problem is? How do you want to be informed of the problem? What is the severity of each problem? Can the severity be used as a way to determine the policy on how the problem is handled? Your monitoring and alerting system must be able to implement your policy.

When choosing a real-time monitoring system, look at what your policy says about how you want it to alert you of problems and how often. For example, if you implement an early-warning mechanism for capacity problems, you may want it to open a trouble ticket with an appropriate priority in your existing helpdesk system. This mechanism should not open additional tickets, once for each probe, which could be every few minutes. Optimally it should update the existing ticket when the issue has remained open for a certain amount of time. At that point it probably should actively page whoever is on call.

A good alerting system will repeat the page if there is no response. However, it is distracting to have the pager or phone continue buzzing, when it is just telling you something that you know already. Such a distraction can increase the time to repair the problem. Instead, the monitoring system should be able to stop alerting when someone acknowledges the alert, indicating that the issue is being worked on. Acknowledging the alert puts the affected service into maintenance mode. When the service is restored to a good state for a configurable period of time, or the technician marks that his or her work is complete, the monitoring system should take that service out of maintenance mode, so that it will send alerts the next time there is a problem. Maintenance mode is similar to the snooze button on an alarm clock. Without this feature, it is tempting to turn off an alarm, which all too often leads to forgetting to fix the problem or, worse, forgetting to reset the alarm. How alerting is handled in maintenance mode, and how and when something is returned to service, should be documented in the alerting policy.

Your monitoring system should be flexible in the forms of alerting that it uses, and it should allow you to use different alerting mechanisms for different types of problems.

Message Format

The error messages that the system delivers must be clear and easy to understand. If recipients of an alert need to look up information or call another person to translate the error message into something they understand, the message is not clear enough. For example, the error message `SNMP query to 10.10.10.1 for 1.2.3.4.5.6.7.8.9.10 failed` is not as clear as `Interface Hssi4/0/0 on wan-router-1 is down`. Likewise, `Connect to port 80 on 10.10.20.20 failed` is not as clear as `web server on www-20 is not responding`.

However, the message must not make assumptions that may be wrong. For example, if the message said `web server on www-20 is not running` rather than that it was not responding, an SA might check whether it was running and assume that it was a false alert, rather than checking whether it might be hung or failing to respond for some other reason.

I'm Hot! I'm Wet!

One night, very early in the morning, a couple's phone rang. The wife woke up and answered, to hear a sultry female voice saying, "I'm hot. I'm wet." Putting it down as a prank call, the wife hung up. Thirty minutes later, the same call repeated. After the fourth call, the husband finally woke up and took the call and bolted from his bed.

It was the alarm system in his machine room. The HVAC had broken down, spilling water under the floor. The alerting system was calling him. The vendor hadn't told him how the alert sounded, just that he should put his phone number in a particular configuration file. Test your alerting system, and inform those who might receive messages from it in your stead.

38.4.4 Escalation

Another component of the policy and procedures that your monitoring and alerting system should implement is the escalation policy describing how long each problem should be permitted to persist before another person is alerted. The escalation policy ensures that even if the person receiving the alert is on vacation

or doesn't respond, the issue will be passed on to someone else. The escalation policy needs to describe the different escalation paths for various categories of alerts.

Case Study: Escalation Procedure

One home-grown system had a particularly sophisticated escalation procedure. The system could be configured with a responsibility grid that mapped services to responsible entities. The entities could be a person or a group of people. The system could be configured with vacation schedules, so it knew whom not to alert. If a problem persisted, the system could walk up a responsibility chain, paging increasingly senior staff.

Each type of service—email, web, DNS, and so on—had its own configuration, and particular systems—the CEO's web proxy, the e-commerce web site, and so on—could be marked as critical, in which case the escalations happened more quickly. Each service had a default responsible entity, but this could be overridden for particular systems that had special requirements. All this led to an extremely effective alerting service.

38.4.5 Active Monitoring Systems

An active monitoring system processes the problems it detects and actively fixes the ones that it knows how to deal with. For example, an active monitoring system might restart a process that it detects is not running or not responding.

Active monitoring systems can be useful up to a point. Although they respond more quickly than a human can, they have limitations. In general, an active monitoring system can implement only a temporary fix. The system won't detect and permanently fix the root of a problem, which is what really needs to happen (see [Chapter 30](#), "[Fixing Things Once](#)"). An active monitoring system also needs to make sure that it reports what it is doing and opens a trouble ticket for the permanent fix. However, it may be more difficult for the SAs to diagnose the real source of the problem when a temporary fix has been applied. The SAs also need to make sure that they don't get lazy and not bother to permanently fix the problems that the active monitoring system has identified. If the SAs simply automate temporary fixes through an active monitoring system, entropy will set in, and the system as a whole will become less reliable.

Case Study: Active Monitoring and the Wrong "Fix"

An active monitoring system at one company monitored `/var` and rotated log files if the disk filled up too much. This deleted the oldest log file and thus reclaimed space. One day, the log files were being rotated each time the monitoring system checked the disk, and the `/var` partition was still 100 percent full. When the SAs logged in to try to resolve the problem, they found that they were unable to look at the log files for clues, because they were all zero length by then.

It turned out that a developer had turned on debugging for one of his processes that was run from cron once a minute. And so, once a minute, the cron job sent out an email. The mailbox (in `varmail`) filled up the disk!

Active monitoring systems also have a limit as to the problems they can solve, even temporarily. Some problems that the system may detect but cannot fix may be caused by a physical failure, such as a printer running out of ink or paper or being switched off or disconnected. If it doesn't accurately diagnose the source of the problem and tries to fix it through software commands, the system may cause more problems than it solves in such cases. Other problems may require complex debugging, and it is not feasible to expect an automated system to be able to correctly diagnose all such problems. In particular, problems that require debugging on multiple hosts and pieces of networking equipment, such as slow file transfers between two hosts on a quiescent network, are beyond the abilities of the automated systems on the market today.

It is a good idea to limit what an automated system can do, in any case. From a security point of view, an automated system that has privileged access to all or most machines is very vulnerable to exploitation. The focus in writing such a system is always utility rather than security, and active monitoring systems are large, complex programs that have privileged access to some machines; thus, there will be security holes to exploit. Such systems are also interesting targets because of the level of network-wide privileged access they have. An active monitoring system is not something that you want to have on an unprotected network. From a reliability perspective, the more this program is permitted to do on your network, the greater the calamity that can befall you if it goes awry.

38.5 Scaling

Once you start monitoring some things and see how useful the monitoring is, you will want to monitor more aspects of your site. Increasing the number of

things to be monitored introduces scaling problems. All monitoring systems have problems as you scale them. Simply gathering all the data that needs to be checked every five minutes is time-consuming. Matters can reach a point where the monitoring system is still trying to collect and process the data from one run when the next round of data collection should begin.

Systems doing historical monitoring usually need to do extra processing to condense the data they store. Some devices may take a while to respond to the information requested, and the system usually can handle only a limited number of open requests at a time. All this can lead to scaling problems as the system monitors more objects.

When scaling a monitoring system to thousands of entities around a large network, network links can become clogged simply from the monitoring traffic. To solve this problem, some monitoring systems have remote probes that collect data and send only summaries back to the master station. If these probes are strategically placed around the network, they can greatly reduce the amount of network traffic generated. The master station stores the data and makes it available to the SAs. The master station also holds the master configuration and distributes that to the remote monitoring stations. This model scales much more extensively than a single monitoring station or multiple unrelated monitoring stations.

38.5.1 Prioritization

Real-time monitoring systems also have other scaling problems. When such a system is watching many aspects of many devices, there will always be some things that are “red,” indicating some form of outage requiring attention. To scale the system appropriately, the SAs must be able to tell at a glance which of the red issues is the “reddest” and having the most impact. Essentially, the problem is that a monitoring system typically has only a few states to indicate the condition of the monitored item. Often, there are just three options: green, yellow, and red. Monitoring many things requires a finer granularity. For example, a very granular priority system could be built in, and the reporting system could display the problems in a priority-ordered list.

38.5.2 Cascading Alerts

Another problem often experienced is that an outage of a nonredundant network component between the monitoring system and objects being monitored can cause a huge flood of failures to show up in the monitoring system, when there is actually only one failure. The flood of alerts can hide the real cause of the

problem and cause the people receiving the alerts to panic.

A monitoring system should have a concept of dependency chains. The dependency chain for an object that the system is watching lists the other outages that will cause this object to show up as experiencing an outage as well. The monitoring system should then use the dependency chain to alter its alerting. For example, rather than sending 50 alerts, it could send one that says, “Multiple failures: root cause ...,” and list only the highest outage in the chain, rather than the outages that are downstream. On a graphical display, it should show the outages along with their dependency chains in an appropriate tree structure so that the root is clearly visible.

In large networks, dependency chains can be difficult to maintain. Some monitoring systems provide a discovery component to identify and maintain the network topology component of the dependency chain.

38.5.3 Coordination

Another problem with scaling a real-time monitoring system relates to how the problems are handled. If multiple people all try to solve the same problem in an uncoordinated way, without knowing that others are working on it, they may make the problem worse. At the very least, some time will be wasted. In addition, they will get in one another’s way and confuse one another about what is happening on the system. The monitoring system should have some way to enable an SA to “claim” a problem, so that other SAs know that someone is working on it and whom they should see if they want to help out. This may be done through the trouble-ticket system and procedures surrounding the assignment of a problem before an SA begins to work on it.

38.6 Centralization and Accessibility

Monitoring should be a central service provided to all SAs by a shared infrastructure team. The other alternative is for each SA group to build and maintain its own monitoring system, which results in wasted resources and duplication of effort. Any of the SAs should be able to add something to the list of things monitored, rather than having to submit a request to a particular SA who will be busy with other tasks. They should also be able to define views into the monitoring system, so that it is easy for them to view only the information about the systems in which they are interested.

Making the monitoring system accessible requires good documentation. Some forms of monitoring may require finding a piece of information that is not

normally used in the course of day-to-day administration, such as the SNMP MIB for a component of the system. The documentation should tell the SA which information he or she will need and how to find it, as well as how to put that information into the monitoring configuration. If there are choices, such as how often the item should be queried, which values correspond to which alert state, how a problem should be prioritized, which graphs to draw, or how to define an escalation path, those options must all be clearly documented as well. In this situation, documenting the preferred defaults is key.

It is frustrating for both the SAs who set up the system and the rest of the SAs in the group if all requests for additional monitoring have to go through one or two people. A system that is not accessible to the whole group will not be used as extensively, and the group will not benefit from the system.

38.7 Pervasive Monitoring

Ultimately, it is nice to be able to monitor everything, or at least everything beyond the desktops. This is particularly important for sites that depend on extremely high availability, such as e-commerce sites. If adding systems and services to be monitored is a task that is performed manually, it will be forgotten or omitted on occasion. To make monitoring pervasive throughout your service, it should be incorporated into the installation process.

For example, if you build many identical machines to provide a service, you should be using an automated installation process, as described in [Chapter 8](#), “[OS Installation Strategies](#).” If you are building machines this way, you could incorporate adding the machine to the monitoring system as part of that build process. Alternatively, you could install on the machine something that detects when it has been deployed in its final location—if you stage the machines through a protected build network—and provide a way for the machine to notify the monitoring system that it is alive and what it needs to have monitored. Being sure that you are, in fact, monitoring everything requires some degree of automation. If such things can’t be automated, the installation process can at least automatically generate a help ticket requesting that it be added to the monitoring system.

For devices where you do not control the software that is installed, such as appliances and network equipment, you can leverage the inventory system to populate the monitoring system. As discussed in [Section 26.1](#), the inventory system should be an integral part of your deployment process. When a device is marked as operational in the inventory system, that change should kick off an automated process that adds the device into the monitoring tools. Equally, when

a machine needs to be taken out of service for maintenance, the inventory system should be updated accordingly. The inventory system should disable monitoring for that system while it is in maintenance mode, and reenable it when the device is marked as operational again. Similarly, when a device is marked as decommissioned in inventory, the inventory system should trigger jobs to clean up all configuration associated with it in other systems, such as monitoring, DNS/DHCP, firewall rules, and so on.

It can also be useful to have a monitoring system that detects when devices are added to the network. Such a system is beneficial at sites that need pervasive monitoring, because it should detect any devices that fell through the cracks and failed to be added to the monitoring system through some other means. In addition, it can be helpful just to know that a device was added and when that happened. If the device is causing a problem on the network, for example, that knowledge can save hours of debugging time.

38.8 End-to-End Tests

End-to-end testing means testing entire transactions, with the monitoring system acting as a customer of the service and checking whether its entire transaction completes successfully. An end-to-end test might be relatively simple, such as sending email through a mail server or requesting particular web pages that cause database queries and checking the content that is returned. Alternatively, it might be a more complex test that simulates all the steps that a customer would make to purchase something on your e-commerce site.

Case Study: Mailing

At AT&T and later Lucent, John Bagley and Jim Witthoff developed the mailing facility. It relays email messages, or mailpings, off mail servers and measures the time the message takes to find its way back to the mailing machine. At one point, mailing was used to monitor more than 80 email servers, including both Unix and MS Exchange SMTP gateways. It provides several web tools to display the delivery time data for a given day or for a given server over a specified period. In addition to historical data collection, mailing includes a mechanism for generating alerts if a particular message has not been delivered after a certain threshold.

The ability to have end-to-end monitoring of the mail transport within their company permitted the team to not only respond to problems quickly, but also develop metrics that let them improve the service over time. Some commercial monitoring systems now offer this kind of

feature.

Of course, once you start monitoring extensively, you are likely to see problems that your monitoring system fails to catch. For example, if you are providing an e-commerce service over the Internet, a sequence of events has to complete successfully for your customer to be able to complete a transaction. Your monitoring system may show that everything is up and working, but an end user may still experience a problem in executing a complete transaction.

Perhaps a transaction depends on things that you haven't thought of monitoring or that are very difficult to monitor. For example, if the transaction relies on sending email, something may be wrong with your mail server configuration, preventing the message from being delivered properly, even though the email server accepted the email for delivery. Or, a bug in the application could cause it to loop, fail, or generate garbage at some point in the process. Or the database could be missing some tables. Any number of things could go wrong, yet not be detected by your monitoring system.

The best way to ensure that the service your customers want to use is up and functioning properly is to emulate a customer and check whether the transaction completes successfully. In the example of an e-commerce site, build a test case that requests pages in the order that a customer would, and check the content of the page returned at each stage to confirm that the appropriate data and links are there. Check the database to ensure that the transaction has been recorded in the appropriate place. Perform a credit card authorization check, and verify that it completes successfully. Send email and make sure that it arrives. Step through every part of the process as the customer would do, and determine whether everything is as expected at each step.

This end-to-end testing can uncover problems that might otherwise go unnoticed until a customer calls to complain. In an e-commerce environment, failing to notice a problem for a while can be very costly. There is a large and growing market of commercial monitoring systems, many of which are focused on the particular needs of e-commerce customer space.

38.9 Application Response Time Monitoring

The other sort of extended monitoring that can be very useful in both corporate and e-commerce environments is application response time. Every component of a system may be operational, but if the system is too slow, your customers will not be happy. In an e-commerce environment, this means that you will lose

business. In a corporate environment, it will lead to a loss of productivity and numerous calls about the slow network or perhaps a slow system or application. In either case, unless you are monitoring the application response time, it will probably take quite a while to figure out why your customers are unhappy, by which time your reputation may have suffered serious damage.

It is much better to devise a way to monitor the application response time as it appears to the end user of the application. It is then useful to have both a historical chart and some sort of a threshold alert for the response time. Application response time is typically an extension of the end-to-end testing discussed previously. Where appropriate, this information can be shared with the application developers to help them scale and optimize their product.

38.10 Compliance Monitoring

A monitoring system can also be used to check compliance with various policies. In some cases, it may be able to fix problems that it detects; in other cases, it may generate a report based on the results. Compliance monitoring may be performed less frequently than historical or real-time monitoring. For example, a site might check compliance with certain policies hourly or daily rather than every five minutes.

A site might have three supported BIOS versions, three supported versions of a particular piece of software, or five supported hardware configurations for a particular type of device. The compliance monitoring system queries each device to find out which hardware, firmware, and software the system has. If the machine is out of compliance for firmware or software versions, the monitoring system could be configured to upgrade it to a particular supported version. For hardware that is out of compliance with the supported hardware configurations, a compliance monitoring system would generate a report.

Case Study: Functionality Skew

One site had an audit script that ran daily, and found and fixed a number of compliance issues. The same site also had a monitoring system that ran more frequently, and noticed some of those same problems but couldn't fix them. This site ran into problems because the two different systems frequently got out of sync, as the updated list of supported versions got pushed to each of them through different processes, which were not synchronized. When they were out of sync, it was not easy to tell which system was right. To solve this problem, the site decided to abandon the auditing system and move all of its functionality into the monitoring

system.

If aspects of compliance monitoring seem too heavyweight to monitor and repair using a monitoring system, consider moving this functionality into an automated configuration-management system, which runs less frequently and can also fix the problem. When choosing this approach, consider how to monitor this automated system. It should be sufficient to check whether that automation is failing, or not completing within a certain period of time. It should not be necessary to monitor the version and configuration of each thing that the automated configuration system is supposed to take care of.

38.11 Meta-monitoring

Meta-monitoring is monitoring your monitoring system. How would you know if your monitoring system died? If you haven't received an alert in three days, is everything fine, or did someone power off the machine that processes the alerts? Sites often forget to monitor their monitoring system; when an outage occurs, they simply no longer get notified.

The simplest form of meta-monitoring is to have the monitoring system monitor a few simple things about itself, such as its own disk becoming full. Alerts should trigger long before the condition would prevent the monitoring system from working.

If you are monitoring a larger number of devices, or if uptime is extremely critical, a second monitoring system that monitors only the primary monitoring system is warranted.

Meta-monitoring should be based on an SLA, just as any other monitoring system is. For example, if you expect the primary monitoring system to do all its probes every ten minutes, meta-monitoring should generate an alert if a probe has not executed for more than ten minutes.

If a system was able to perform a full rotation of probes within the SLA for weeks and now suddenly it can't, this situation should be investigated. If this condition persists, perhaps a hard drive is starting to fail, the network is misconfigured, or a process has gone out of control. It may also indicate that someone has accidentally messed up the configuration.

Historical meta-monitoring can record how the system responds to growing numbers of probes. As more services are monitored, the system needs more time to probe them all. Collecting historical data can help predict the point at which the monitoring system will no longer be able to meet the SLA. Scaling

techniques can be invoked to prevent the monitoring system from becoming overloaded.

38.12 Summary

This chapter discussed monitoring in its two incarnations: historical data gathering and real-time monitoring and alerting. The two forms are quite different in what each involves and what each is useful for, yet you need similar feature sets in both for many aspects of monitoring.

Historical availability monitoring and data collection means tracking availability and usage of systems so as to graph and analyze the data later. It involves gathering, storing, and condensing lots of data. Historical data collection requires lots of disk space, databases, and processing. It is useful for capacity planning, budget justification, customer billing, providing an overview of what is happening at a site when a problem is detected, and anomaly detection.

Real-time monitoring involves polling systems to check their state and watching for problem notifications from built-in system monitors. Real-time monitoring is generally combined with an alerting system. This combination is used to detect problems and notify the SAs of them (almost) immediately. It is a tool for providing better service and for detecting the root of a problem. With this approach, the alerts give more precise information than a customer-oriented problem report, such as “I can’t download my mail.”

Both types of monitoring are required tools at an e-commerce site because the customers are so much more distant and fickle—they don’t really care whether you know about or fix the problem, because they can simply go to another site. Monitoring is a useful tool in both its forms in any well-run site.

Both forms of monitoring have problems when it comes to scaling, and splitting the monitoring system into several data gatherers and one central master is the best method to resolve these problems. Real-time monitoring also has scaling problems involving prioritization and responding to alerts in an appropriate, timely, and coordinated manner.

As your monitoring system becomes more sophisticated, you will want to consider implementing end-to-end testing and application response time testing so that you know exactly what the end user is experiencing and whether that is acceptable. You also may want to consider ways to ensure that everything is monitored appropriately, particularly in a service provider environment. To do so, you can consider ways to add a system to the monitoring list when it is built.

You may also look at ways to detect when devices have been added to the network but not to the monitoring system.

Monitoring is a very useful tool, but it is also a project that will increase in scope as it is implemented. Knowing the ways that the system will need to grow will help you select the right monitoring systems at the beginning of the project, scale them, and add functionality to them as needed during their lifetime.

Exercises

1. How do you monitor systems for which you are responsible? If you do not have a formal monitoring system, have you automated any of your ad hoc monitoring?
2. Have you implemented active monitoring for anything in your environment? If so, how well does it work? Does it ever prevent you from finding the root cause of a problem? Explain why or why not.
3. Which systems would you add to a monitoring system if you had it, and why? Which aspects of those systems would you monitor?
4. If you already have a monitoring system, how might you improve it?
5. Which features discussed in this chapter are most important to you in selecting a monitoring system for your site, and why?
6. Which, if any, other features not discussed are important to you, and why?
7. Investigate freely available historical data monitoring systems. Which one do you think would suit your environment best, and why?
8. Investigate commercial historical data monitoring systems. Which one do you think would suit your environment best, and why?
9. If you had to choose between the free and commercial historical data monitoring systems that you selected in Exercises 7 and 8, which would you pick for your site, and why? Which, if any, features do you feel it lacks?
10. Investigate freely available real-time monitoring systems. Which one do you think would suit your environment best, and why?
11. Investigate commercial real-time monitoring systems. Which one do you think would suit your environment best, and why?
12. If you had to choose between the free and commercial real-time monitoring systems that you selected in Exercises 10 and 11, which would you pick for your site, and why? Which, if any, features do you feel it lacks?

- 13.** How many items—machines, network devices, applications, and so on—do you think a monitoring system at your site would need to scale to in the next three years? What would you need to meet that demand?
- 14.** Are there any advantages or disadvantages to using the same package for both types of monitoring, if that is possible?

Chapter 39. Namespaces

In this chapter, we explain principles of organizing namespaces. A **namespace** is a set of names that have a common purpose. Examples of namespaces are account names, printer names, fully qualified domain names, Ethernet addresses, service name/port number lists, and home directory location maps.

A **nameservice** associates attributes with each name in a given namespace. For example, a computer user directory (nameservice) uses the account names namespace as a set of unique keys and associates unique identifiers (UIDs), home directories, owners, and so on with each account name. A configuration management database (CMDB) uses the hostnames namespace as a unique key and associates with it asset numbers, hardware serial numbers, customer (owner) information, base and optional hardware components, operating system (OS) version and patches, installed software, and so on.

39.1 What Is a Namespace?

The term “namespace” can be confusing because it can be used to refer to either an abstract concept or a concrete data set. For example, usernames are a namespace. Every multiuser operating system has a namespace that is the list of identifiers for users. Therefore, a typical company has the abstract concept of a username namespace. However, the namespace used at my company is different from the one at your company (unless you’ve just hired all of my co-workers!). In that sense, each company has different username namespaces: your set of users and my set of users. Most of this chapter uses the term “namespace” to refer to a particular (concrete) namespace database. The text is specific when it’s worthwhile to differentiate the two.

Namespaces come in many shapes. Some namespaces are flat; that is, there is no hierarchy. Often it is the implementation of the nameservice that uses the particular namespace that determines the shape. For example, in Windows, a WINS directory is a flat namespace. In Unix, the set of UIDs is a flat namespace. Other namespaces are hierarchical, such as a directory tree. No two files in a particular directory can have the same name, but two different files called `example.txt` can be in different subdirectories.

The larger and more sophisticated an environment becomes, the more important it is that namespaces be managed formally. On the one hand, smaller environments tend to need very little formality with their namespaces. They all

might be controlled by one person who keeps certain information in her head. On the other hand, mega-corporations must divide and delegate the responsibility among many divisions and people.

It is very powerful to simply acknowledge that your environment has namespaces in the first place. Until that happens, you just have piles of data. When each namespace is thought of independently, we lose the potential benefits that can be gained by linking them. New SAs tend to see each namespace as a separate item, but over time they learn to see a more global picture. In other words, they learn to see the whole forest rather than getting caught up with each tree. Seeing a particular namespace in the context of a larger vision opens up new options.

39.2 Basic Rules of Namespaces

The basic rules of namespaces are very simple:

- Namespaces need policies: naming, longevity, locality, and exposure.
- Namespaces need procedures: adding, changing, and deleting.
- Namespaces need centralized management.

Namespaces should be controlled by general policies. The larger your SA team is, the more important it is for policies to be actual documents, rather than oral traditions. As a team grows, these written policies become tools for communicating with other SAs or training new SAs. You can't reprimand an SA for permitting a customer to name a new PC outside the permitted name rules if those rules are not in written form. Written policies should be the basis for the requirements specified when you create automation to maintain namespaces. Written policies also set expectations with customers. The policies are used as the basis for defining procedures for making changes to the namespace and related processes.

39.3 Defining Names

The namespace's naming policy should answer these questions: Which names are permitted in this namespace? Which names are not permitted in this namespace? How are names selected? How are collisions resolved? When is renaming allowed?

One needs rules for which names can be part of a namespace. Some rules are dictated by the technology. For example, Unix login IDs can consist of only alphanumeric characters plus a limited number of symbols. Other rules may emerge from corporate policy, such as restrictions on "offensive" login IDs.

External standards bodies set rules, too: Before RFC 1123 ([Braden 1989](#)) changed the rules in 1989, DNS names couldn't begin with a digit, making it difficult for 3Com, Inc., to register its domain.

Different naming methods can be used for different namespaces, and the choice of a method will often be influenced by whether the namespace is flat or hierarchical. Some companies may choose to treat a hierarchical namespace, such as fully qualified domain names, as a flat one—for example, by making all computer names unique before the domain is appended.

The methods for choosing names fall into a number of general categories:

- **Thematic:** All names fit a theme, such as naming all servers after planets, and using planet names from science fiction when you run out of real planet names. This approach is possible for small namespaces but does not scale well. It can add a sense of fun in a start-up or other small company.
- **Functional:** Names have functions—for example, account names for a specific role (`admin`, `secretary`, `guest`), hostnames that reflect the duties of the machine (`dns1`, `cpuserver22`, `web01`), or names for permission groups (`web-masters`, `marketing`). This approach can be useful for creating functional groupings, such as email lists, shared mailboxes, or aliases that are used to access a service independent of the machine(s) that provide it. However, functional identifiers do not work well for individual usernames. Imagine changing your role from SA to developer. Does your username change? What happens to your email, accounts, and access?
- **Descriptive:** Descriptive names usually communicate factual information, rather than rules. Good examples are shared storage mount labels that describe the customers or data for which the partition is intended (`/develop`, `\Departments\Finance`), and printer names that indicate which type of printer or driver is used (`laserjet`, `photo`, `11x14`) or the printer's location (`testlab`, `breakroom`, `CEO-desktop`). Geographical names, usually indicated by city or airport code, are heavily used by large organizations for permissions groups and email lists (`sjc-all`, `chicago-execs`, `reston-eng`). Descriptive names work well for small companies and for abstractions, like mailing lists.
- **Formulaic:** Names fit a strict formula. For example, some sites name all desktop workstations as `pc -` followed by a six-digit number. Login names might be strictly the first initial followed by the first six letters of the last name, followed by a random series of digits to make a unique name. The advantage of formulaic names is that they scale well and can be generated

automatically. However, they can be difficult to remember, and it can be easy to confuse one name with another, or to mistype it. In some cases, the formula used needs careful consideration. For example, people find usernames impersonal when they are a random combination of letters and digits, with little resemblance to their names. This naming practice makes people feel like they are but a small cog in a large machine.

- **Hybrid:** Often several naming elements are combined. The most common hybrid is functional naming with descriptive naming when an organization has multiple geographical locations—for example, `nyc-marketing` or `sjc-web-03`. This is especially true for networking gear, where often names are used to provide maximum debugging information, such as abbreviations for the network provider, the colocation facility, or even the rack location. Formulaic naming can be combined with functional and descriptive components. For example, mailing lists might always begin with `list-` and be followed by a descriptive and/or functional name. Or a formula for computer names might require functional and descriptive components, such as the kind of machine it is (workstation, server, or router), whether it is physical or virtual, its location, the operational environment (production, test, or development), and its function. The advantage of these sorts of names is that they can convey a wealth of useful information to those who understand the formula.
- **No method:** Sometimes, the formula is no formula. Everyone picks something. Conflicts and collisions are resolved by first-come, first-served policies.

There is tension among these methods. Once you use one scheme for a particular namespace, it is difficult to change. Many organizations avoid making a specific choice by combining more than one method, but will usually pick one as primary and another as secondary in the combination.

Also, when choosing a formula, be aware that if too much information is encoded into a name, it can have knock-on effects. For example, if machine names contain location information, what do you do when the machine moves to another location? If you have to change the name, do you know all the dependencies on that name for people, processes, applications, licenses, certificates, and documentation? In a large company that is likely to have such a naming scheme, datacenter consolidation or expansion can cause mass machine moves to happen, even into other states or countries.

Functional names can make software configuration easier: Helpdesks can

support software more easily if the mail server is called `mail` and the calendar server is called `calendar`. However, if such services ever move to different hosts, a confusing situation can arise. It is better to have functional aliases that point to such hosts. Aliases, such as DNS CNAME records, are great for non-interactive service machines such as mail, web, and database servers, or distributed file system (DFS) names for Windows file servers. Aliases don't work nearly as well for interactive compute servers, because users will notice the hostname and start to use it. It can be confusing when logs are generated with the real hostname rather than the functional alias, but it would be more confusing if they referred to the alias with no way to determine which specific machine was intended.

Sequential Names

Formulaic names give a false sense of completeness. A customer submitted a panicked trouble ticket when she noticed that some of the hosts in the sequence `software-build-1` through `software-build-9` weren't reachable. There was no problem. Those missing machines had been recycled for other purposes.

Theme names can be cute—sometimes too cute. One part of Bell Labs where Tom worked used coffee-related names for public printers: `latte`, `decaf`, `grande`, `froth`. Another part of Bell Labs named its printers after the room they were in, or which office they were nearest, plus `-color` if it was a color printer: `2c-310`, `3f-413`, `3f-413-color`. While the first group's names were cute, they were confusing. The latter configuration was less fun, but new users never had to suffer the embarrassment of wandering around the hallway asking every stranger they met if they knew where to find “decaf.”

The method used for naming reflects the corporate culture and the organization's size. In a company with hundreds of thousands of individual workstations, it is hard to come up with unique, fun names for each of them, and formulaic naming standards can aid the helpdesk to prioritize issues with servers ahead of issues with workstations. In such a large company, the corporate culture is often more serious and business focused. Smaller companies are more likely to want to create a progressive, fun atmosphere, focusing on the individuals and encouraging their creativity. In such a culture, `latte` is a great name for a printer.

Names have security implications. Intruders might find `sourcecodedb` a

more interesting target than `server05`. Alternatively, intruders have long known that SAs usually break the naming rules for their own systems. Intruders may target these machines as the best sources of privileged credentials. If they discover a network where all hostnames are strictly formulaic except for a few random machines named after *Star Trek* characters, the intruders will assume that the *Star Trek* hosts are the SAs' machines. It's even reasonable to assume that the lead SA's host is the one named `picard` and that the person in charge of security sits in front of `worf`. Although we don't encourage anyone to rely on security through obscurity, there are benefits in camouflaging the machines with unremarkable names. For further reading on the topic of hostname selection, we recommend RFC 1178 ([Libes 1990](#)).

Difficult-to-Type Names

At one site, a group decided to name its machines after famous mathematicians, using short names such as `boole` for the machines that people should log into and longer ones such as `ramanujan` for those they shouldn't. It's much easier to type `boole`!

39.4 Merging Namespaces

When organizations merge operations, they often need to merge namespaces as well. The first thing to address is the namespace policies. The companies can adopt one of the existing policies and naming standards, and migrate the other organization to the new policies and standards. Alternatively, they can create a set of new policies and standards and then migrate both organizations to comply with those new policies and standards.

Once the policies and standards for the new organization have been determined, the other aspect of merging namespaces to consider is dealing with namespace conflicts. A namespace conflict occurs when both namespaces contain the same name. For example, a conflict occurs if both user account databases have usernames `srchalup`, `chogan`, or `tal`.

Resolving namespace conflicts can be a political issue as much as a technical one. Sometimes a merger is really an acquisition but is called a merger to make the other group feel better about the changes that will be coming. Mutually deciding on a policy for handling namespace conflicts before they arise will help prevent bad feelings, especially if the policy is perceived as fair by both sides.

Namespace Conflict Resolution

The chances for name collision are greater if both organizations are using the same method of assigning names. Whose server named `gandalf` has to be renamed? Which Susan has to change her login name from `SUE` to something else? Where will mail to the `sales` alias go? The following sample policies help resolve such conflicts:

- **Servers:** All servers with name conflicts are given a company-specific hostname alias in the directory or DNS. Thus, the two servers named `gandalf` are reachable as `gandalf-CompanyA` and `gandalf-CompanyB` until one can be retired or easily renamed.
- **Logins:** The customer who has been with his or her organization longest gets to keep the login name. In one merger that Strata assisted with, a high-ranking manager gave up his first-name login to an individual employee in the firm being acquired, even though he could have used his status in the company to make himself an exception to the policy. Word of this got around quickly and made a positive difference in moving forward.
- **Email:** `First.last` conflicts in email addresses can be very irksome. The policy that may meet with most success is that both individuals change their email addresses. The mail gateways are then configured to route mail to these customers based on the domain to which it was addressed. Mail to `susan.jones@companyA` will be resent to `susan.a.jones@Merged`, and mail to `susan.jones@companyB` will be resent to `susan.b.jones@Merged`. Most firms choose to reroute mail rather than bounce it, but the same strategy can be used to bounce mail with a customized message: “Susan.Jones@companyA: Address has changed to Susan.A.Jones@companyA. Please update your address books.” When implementing an email routing policy such as this one, it is important to specify a time limit, if any, for when the special routing will be disabled.

39.5 Life-Cycle Management

A namespace’s life-cycle management policy should answer this question: When are entries in this namespace removed? Some entries in a namespace need to expire on a certain date or after a certain amount of inactivity. You might proscribe that accounts for contractors be sponsored by a regular employee, who must renew the request once a year, and that such an account must be removed if

no renewal is performed. IP addresses can be scarce, and in a loosely controlled environment, you might expire the IP address that has been granted to a customer if a network-monitoring system shows that the IP address hasn't been used in a certain number of months.

Names, once exposed to customers, last longer than you would ever expect. Once a name is established in the minds of your customers, it is very difficult to change. You should plan for this. If an email address is printed on a business card, it is difficult to recall all those cards if you require a person to change an email address. Once a document repository is announced to be on file server *fred*, don't even think of moving it to *barney*.

Good technology lets you obscure names that shouldn't be interrelated. The Unix automounter can be used so that customers refer to the repository as *homedocs*, and the name of the file server is obscured. The location of the documents should not be tied to the name of the server. The fact that they are interrelated shouldn't matter to your customers. Aliases can be a good solution here, though some technologies, such as NFS, require clients to be rebooted for them to "see" the change. On Windows, use DFS to obscure file server names. Never name a web server *www*. Give it a generic name, and make *www* an alias to it. Announce only the *www* name, and you can move the functionality to a different host without worry. If you consistently use aliases and other technologies that obscure names as appropriate, you can move functionality from one host to another with confidence.

A Machine Named *calendar*

For many years, the calendar server that Tom used was on a host named *calendar*. It seemed like a good idea at the time, because the host was purchased expressly for the job of being the calendar server. But eventually it also became the main print server, and customers were confused that their print jobs were being spooled to a host named *calendar*.

The hostname couldn't be changed, because the software license was locked to that hostname. This problem could have been avoided if the host had been called something else and *calendar* was an alias for the machine. Tom could also have defined *printer* as an alias for the host, and made the customers happy, or at least less confused.

39.6 Reuse

The namespace's reuse policy should answer this question: How soon after a name has been deleted or expired can it be reused? Sometimes, the name can be reused immediately. For example, some sites might want to reuse a printer name immediately so that PCs do not need to be reconfigured. Other sites may use a configuration-management database (CMDB) where machine names are unique and are never reused to preserve the life-cycle management and audit trails.

You should be more concerned with email addresses, however. Reusing an email address too quickly can result in misdirected mail that could expose confidential information to someone who is not entitled to know it.

Many large companies tie user IDs and personal email addresses to an individual forever. The only time they are reused is when that person comes back to the company, at which point he or she has the same employee number, user ID, and email address as before. Restoring these attributes should not be confused with restoring all the access rights that the person had before leaving. Access rights should be freshly assigned based on the person's new role.

There may be a different policy for email addresses used for shared mailboxes and email lists. For those, you might have a policy that once such an email address has been deleted, no one else can reuse that address for six months or a year, so that one group is less likely to receive email intended for a different group.

A reuse policy can be implemented in software. However, in small, infrequently changing namespaces, the SAs should simply be sensitive to the ramifications of reuse. For example, if you are asked to give a customer's PC a name that was recently used for a popular server, you might suggest a different name so as to prevent confusion. The level of confusion will be particularly high if a lot of hardcopy documentation mentioning the server's name is still in the field.

39.7 Usage

A single namespace may be used in multiple nameservices. For example, the login ID namespace is used in the user accounts database, the HR database, the authentication service, the authorization/permissions service, the email system, and so on. For each namespace, you need to consider the following:

- **Scope:** Where will the namespace be used?
- **Consistency:** When will attributes associated with a name be kept consistent across nameservices that use the namespace?
- **Authority:** For attributes that are to be kept consistent across

nameservices, which service is the authoritative source?

39.7.1 Scope

A namespace's scope policy should answer this question: Where will this namespace be used? How global or local a particular namespace is can be measured on two axes: diameter (geographically, or how widely it is used) and thickness (how many services use it).

The **diameter** is how many systems use a particular namespace database—single host, cluster, division, enterprise-wide, and so on. Although your entire enterprise might use Microsoft ActiveDirectory, a given namespace database—say, username/passwords—might be usable only for your department's machines. Other departments may have different lists of users and passwords for their machines.

RADIUS is an authentication protocol for VPN servers and other network devices that can be implemented in such a way that devices all around a global enterprise access the same database for usernames and passwords. People can log in with the same username/password no matter which VPN servers they use when traveling around the world.

Network Information Service (NIS) makes it easy to manage namespaces that have a diameter of a single cluster of Unix/Linux hosts.

Case Study: A Simple Way to Set Naming Standards

Bell Labs Research has many different Unix environments, or clusters, but amazingly few conflicting UIDs. This fact was discovered, much to our surprise, when some of these environments were merged. How this happened is an interesting story.

Many years ago, someone invited all the SAs from all the computing centers to lunch. At that lunch, the SAs divided up the UID space, allocating large ranges to each research center. Everyone agreed to stay within their allocated space except when creating accounts for people from other computing centers, in which case their UIDs from that center would be carried forward. No policy was created, no bureaucratic allocation czar was appointed, and no penalty scheme was implemented. Everyone agreed to follow the decisions because they knew it was the right thing to do. Shortly after the meeting, someone sent out email describing the ranges.

Years later, those UID ranges are still followed. When new SAs are

hired, someone forwards them a copy of that email explaining what was agreed on many years ago. Each center has some kind of “create an account” script that embodies the guidelines in that email. This convention has worked very well all these years because it is truly the simplest solution for the problem.

The diameter of a namespace has a lot of implications. If each division has a different namespace for login IDs, what happens if a person has accounts in two namespace databases? For example, what if `tal` is Tom Limoncelli in one division and Terry Levine in another? This can be fine until Tom needs a login in Terry’s division. Should Tom be `tal2` exclusively in Terry’s division, or should Tom be required to change his account name everywhere? That would be very disruptive, especially if Tom needed an account on Terry’s network for only a short time. It is also common to run across this issue when a company wants the diameter of a namespace to encompass other completely independent and unrelated companies, organizations, and individuals. The most commonly seen solution in this scenario is to use email addresses as the unique user identifier. However, email addresses change when people change employer or Internet service provider (ISP). Federated identity is another approach and is discussed further in [Section 39.8](#).

The **thickness** of a namespace is based on how many services use it. For example, a company might allocate a unique ID, such as `tal` or `chogan`, for each employee and use that identifier for the person’s email name, login ID, ID for logging into intranet services, name on VPN services, and so on. Even though these are different databases, provided by different nameservices using different protocols—ActiveDirectory, NIS, RADIUS, and so on—the same ID is used for all of them. The namespace (the person-to-user ID mapping) is common across all the nameservices that use it, even though each one will have other parameters associated with that ID. For example, each of those uses of the ID will have different permissions associated with it, and might even have a different password.

Case Study: Lucent’s “Handles”

It can be useful to have a single, global namespace and to encourage all other namespaces to align themselves with it. Lucent gave each employee a “handle” that is a unique identifier. Employees could select their own handles, but the default was a person’s first initial followed by his or her last name. This namespace was globally unique, a challenge considering

that in 2000, Lucent had more than 160,000 employees.

Each division was encouraged to create account names that were the same as the person's handle. All services run by the central IT department (email, human resources, remote access) used handles exclusively to act as a role model. This system resulted in the best of both worlds. Divisions could adopt login IDs that matched the associate's Lucent handle, but they could deviate if they wished. Deviations meant accepting the risk that current and future collisions might cause confusion.

The benefits of using the Lucent handle rather than letting customers select unique ones were obvious, so little enforcement was required. During corporate acquisitions, the company being acquired had to deal with namespace collisions, but could retain backward-compatible names when needed.

Sometimes, a company does not want a globally flat namespace. In some cases, the technology in use provides for a hierarchical namespace instead. For example, in many companies, it is not required that hostnames be unique within an entire corporation, because DNS provides for zones and subzones. With departmental DNS zones, each department can have a `www`. A department could even name its desktop PCs `pc` followed by a number, and sites wouldn't have to coordinate with one another to ensure that no overlapping numbers were used. However, where the namespace is thicker and used by both hierarchical (e.g., DNS) and flat (e.g., WINS) technologies, uniqueness is important. Therefore understanding the current and future thickness of the namespace is an important prerequisite to defining the naming policy.

Case Study: Wide and Thick E-commerce Namespaces

E-commerce sites typically have a username namespace that is extremely wide and thick. The customer establishes and uses one username and password for all the services at the site, whether they are on one machine or hundreds. Yahoo is well known for this practice. Once a customer establishes a profile, it applies to all the services offered. The customer might have to activate those additional services, but they are all tied to one username and password for the person to remember.

In contrast, some large enterprises have multiple user IDs for a single employee, or a single ID with different passwords (low consistency). Both scenarios make it difficult for the end user to remember which credentials

to use for which service.

39.7.2 Consistency

A namespace's consistency policy should answer this question: Where the same name is used in multiple nameservices that have additional attributes in common, which of these common attributes will be kept consistent? Consistency is measured on two axes: level and strength. The level of consistency measures the percentage of nameservices that should have consistent values. The strength of the consistency measures how tightly the consistency is maintained.

High consistency means that a name used in one place will have the same values for all common attributes in all nameservices. **Low consistency** means that the values of the common attributes will be different across the various nameservices.

For example, you might establish a policy that if someone has a Unix account, the numeric UID must be the same everywhere the person has Unix accounts. You might establish a policy that when the same ID is used to access services within the company, such as account login, email, and internal authenticated web services, the password is kept in sync, but when that ID is used to connect to the company from a less secure location over a VPN, a two-factor authentication mechanism is required. You might also have a policy stating that when the user ID is used to get elevated privileges within the company, the same two-factor authentication system is used. Alternatively, if two-factor authentication is not an option, the policy might state that the password used to connect to the company's VPN server is not kept in sync with the one used elsewhere. In fact, it would be wise to *require* different passwords for both systems.

Case Study: Different User IDs

One large company was formed out of the merger of two similar-size companies, each of which had a completely different standard for allocating user IDs. There were no conflicting user IDs, and unifying all user IDs into a single consistent standard was considered too disruptive, risky, and expensive.

Each entity continued to allocate user IDs using its own standard. When someone from organization A needed access to systems in organization B, that person was allocated a new user ID that conformed to organization B's standards. The end result was that most people had two user IDs: one to access all systems and services in organization A and the other to

access all systems and services in organization B.

Email addresses were unified into a single system, however. The unified email addresses did not use either of the user IDs, but rather a third standard. However, both user IDs and the email account were linked to the person's employee number. To make things easier for the end users, the SAs provided a web-based tool for password changes that updated the passwords for the email account and both user IDs at once, thereby keeping the passwords in sync, even though the user IDs were different.

In this case, the passwords had higher consistency than the login IDs!

Strong consistency means that it is not possible for the attribute to be different across nameservices where consistency is supposed to be maintained.

Weak consistency means that it is possible for an attribute to have different values across nameservices, even when it should be consistent.

For example, consider a company where a person's individual login ID and password are supposed to be the same across all corporate systems, but Unix passwords are checked against a Unix-specific authentication service, and Windows passwords are checked against a Windows-specific authentication service. The SAs provide a tool to enable people to change their passwords in both systems at once. However, if it is still possible to use the native Unix and Windows tools to change the passwords in each of the environments separately, then the consistency is weak.

Now consider a site where all systems that map a login ID to a real name get periodic updates with that mapping from a particular nameservice, such as the HR database. Any locally made changes will be overwritten with the data from the HR database. The real name attribute in the email system, the trouble-ticket system, user account databases, the phone directory, the asset database, and so on is automatically updated by an authoritative source. Even if it is possible to make local changes, they do not endure because they are actively kept consistent with the HR database. In this case, the real name attribute would be said to have strong consistency.

One UID for an Account Everywhere

At Bell Labs, each person has his or her own login ID and password, and can use those credentials to log in to all the general-purpose Unix machines. The login ID is associated with the same UID on all those systems. However, not everyone can access the special-purpose machines.

Tom inherited ownership of a DNS server that was one of these machines. Only people who needed to make updates to DNS had accounts, and their UIDs were allocated starting at 1,000 and incrementing upward. The previous SA had justified the situation by stating that the DNS server was too secure to ever run NFS, and using NFS on the DNS server would create a circular dependency between the two services. Thus, since it would never run NFS, the UIDs didn't have to be the same as everywhere else, and it would have taken an entire 10 seconds per account to identify their regular UIDs.

That might have been a minute per year that the previous SA saved in the three years he ran the machine. However, when backups of this machine were restored to other hosts, the UIDs no longer matched. To forestall future problems, Tom set a policy that all new accounts would be created using UIDs from their home system, and the legacy UIDs would be realigned during an upgrade, when downtime was permitted.

This is an example of weak consistency. It started out as weak and low consistency. After Tom had completed his remediation work, the consistency was high, but still weak, since no automated systems or shared database maintained the consistency.

39.7.3 Authority

Strong consistency requires designating an authoritative source for each attribute, a practice called having a single source of truth. With this approach, changes can be made only at the authoritative source, with the changes then being distributed from the authoritative source to other systems.

Often, the easiest way to ensure strong consistency is to eliminate the need for it by having a wide and thick nameservice that is called directly from the other nameservices, rather than duplicating the data into the other nameservices. However, sometimes the protocols that are available in the various nameservice applications do not allow for this approach, or the resource requirements and response times make it undesirable. In these cases, a data synchronization method needs to be implemented.

Keeping attributes in various nameservices in sync requires a unique identifier from a namespace that is referenced by all the nameservices. Often this is the primary key, such as the user ID or hostname. In some companies, another unique identifier might be used, such as the employee ID number or the asset number for machines.

Case Study: Updating an Authoritative Source

At one company, the CMDB was a central point of integration for installation and upgrades of OSs and software packages, as well as configuration of location-based services. The CMDB used the asset tag as a primary key. Hostnames were also unique and had a one-to-one mapping with asset tags. Likewise, virtual machines had asset tags (and hostnames). The CMDB was integrated with the DNS, DHCP, and IPAM (DDI) nameservice, which provided IP addresses for hosts and location information for networks, to support the localization processes. The DDI service stored the asset tags as a foreign key to facilitate the data exchange.

The company decided that the DDI service was the authority for the network information, such as the IP addresses, Ethernet MAC addresses, and fully qualified domain names (FQDNs). For performance reasons, the data was duplicated between the DDI service and the CMDB, rather than requiring the CMDB to look up the information when it was required, which meant that it needed to be kept in sync. Since the DDI service was the authority, data from the DDI service would overwrite what was in the CMDB, so changes to these fields in the CMDB would not persist.

The DDI service was considered the authoritative source for all network information. However, a different process sent the MAC address for newly installed machines to the CMDB. The CMDB had a separate table for the MAC addresses sent by this process. Now the CMDB knew the correct MAC address for the asset, but the otherwise authoritative DDI service listed the MAC address as “unknown.” To resolve this conflict, when the DDI service synced its data to the CMDB, that one field was sometimes synced in the opposite direction. If the DDI service listed the MAC address as unknown and the CMDB had a MAC address for this system in another table, then the CMDB sent that MAC address to the DDI system, because the CMDB was temporarily more authoritative than the DDI system. After that, all future changes to the MAC address would need to be made in the DDI service and would subsequently be reflected in the CMDB just like changes in other data.

The astute observer will note that the DDI service is not really the authority on the MAC addresses for a machine—the machine itself is. However, for integration and maintenance purposes, it is easier to have a single central authority. The central authority can, however, use discovery

mechanisms to automatically keep its data accurate.

39.8 Federated Identity

Federated identity links a person's electronic identity and associated attributes across different identity management systems. A subset of federated identity is single sign-on (SSO), where a person signs into one service and receives a ticket, or token, that is trusted by other services, and sometimes even other organizations, so that the person does not need to sign in again, at least until the token expires. SSO is a subset of federated identity because it relates to authentication only, and not to other aspects of a person's identity. Federated identity is wide and thick. You have one electronic identity, which is used everywhere, on every service, with the same attributes.

Within your own company you can build a single employee database that is linked to all your internal directories and authentication and access control systems. You add new employees in a single, central database. Access is provisioned from a central system. When an employee moves to another organization, there is a central point to review access to see if it is still appropriate. Likewise, when an employee leaves, there is a single place to update the person's status, which revokes all of his or her access. But what happens when your company subscribes to cloud-based solutions? This is where federated identity comes into play.

As companies increasingly move toward SaaS solutions, they need a way to centrally control access to all these different services. If each SaaS solution had its own user database for your company, with its own set of passwords, then each user would need to remember the login information for each site. That's not a good user experience, especially if users have previously had single sign-on for services provided within the company. More importantly, the joiner, mover, leaver (JML) process becomes unwieldy. You need to add people in multiple vendors' databases. You can no longer review and control their application access from a central point. And you have to contact all of these vendors to have employees' access revoked when they leave the company.

A federated identity management provider gives you that single point of control. You manage your employee database centrally with this provider, and you provision who has access to each of the various SaaS solutions centrally. You can include whatever additional information you need, such as department-specific codes for billing service usage back to the various departments. Users also manage their passwords centrally. They get SSO tokens when they log into

one of the services, so that they do not need to log into the next service, even though it is hosted by a different SaaS provider.

The essence of federated identity is that one's identity is portable across multiple autonomous environments. This is achieved through the use of open standards, which provide maximum interoperability across independent companies. Technologies used for federated identity include Security Assertion Markup Language (SAML), OAuth, Secure Cross-Domain Identity Management (SCIM), and OpenID.

Ultimately, a person's identity and credentials can be stored by the provider, and then associated and unassociated with particular companies as the person changes employers. A person's identity and credentials are not tied to the current employer, but rather to the person himself or herself, who may be transiently linked to one or more employers. This individual may have access to some SaaS services privately, and others as part of his or her employment. Nevertheless, the person's identity and single sign-on capabilities are unified across all of these services through the federated identity management service.

39.9 Summary

This chapter established some rules for namespaces. First, we must acknowledge that namespaces exist. Then, it becomes obvious that all namespaces share certain qualities.

Namespaces need policies on naming, access control, longevity, scope, consistency, and reuse. These policies must be written, approved by management and technical staff, and available for reference to customers and SAs, thereby clearly establishing the rules of your namespaces. Small sites that run without such documentation and survive do so because the few SAs work closely together and have such policies as part of their culture. Unfortunately, this informality doesn't scale well. When such companies grow, they may suddenly find themselves with a larger group of SAs who have not been indoctrinated into the "standard" way of doing things.

Establishing written policies and procedures about namespaces improves communication within the SA team and communicates expectations to customers. Namespaces can greatly benefit from central management.

Namespaces are used by nameservices, which associate additional attributes with the names in the namespace. When multiple nameservices have certain attributes in common, the consistency policy determines whether those attributes are kept in sync across the different nameservices.

Federated identity management is required for companies that outsource services to cloud providers, to enable them to efficiently manage the JML process as well as access to the various services. A federated identity database is wide and thick, and it provides customers with a single sign-on experience across multiple SaaS vendors.

Exercises

1. What are the namespaces in your environment?
2. Which namespace-related policies does your organization have? Are they written or simply understood?
3. How are machine names defined?
4. How are user identities defined? How many user identities can a person have within your organization?
5. Which namespaces in your environment are hierarchical, and which are not?
6. One aspect of an environment's maturity can be measured by determining whether namespaces in the environment meet the basic requirements listed in this chapter. Evaluate your environment on this basis.
7. Suppose that you moved to a new organization and your login ID was already in use. What would you do?
8. Research federated identity management providers. Evaluate and briefly describe some of them. Explain which one you would choose for your company, and why.

Chapter 40. Nameservices

In the previous chapter, we looked at namespaces as a concept. In this chapter, we look at how those namespaces are managed, enriched with additional information, and provided as a service. A nameservice is an instantiation of a namespace that associates particular attributes with names in the namespace and makes this information available to other systems and end users using some particular protocol or set of protocols. Examples of nameservices include DNS, DHCP, LDAP, ActiveDirectory, the HR directory, the configuration-management database (CMDB), and a two-factor authentication (2FA) system. A namespace is a set of unique keys. A nameservice is a concrete implementation of a namespace that associates a set of attributes with each of those keys.

40.1 Nameservice Data

Nameservices associate additional data with the individual names in a namespace. They may also link names in one namespace to names in another namespace. For example, user identifiers are names in one namespace that a nameservice links to home directories, which are names in a different namespace.

As described in [Section 39.7.1](#), multiple nameservices may use the same namespace, each associating a different set of attributes with that name. For example, the namespace that is the list of user IDs at a company may be used by a Unix account database, a Windows account database, the corporate directory, the email system, the permissions database, and so on.

When setting up a specific nameservice, there are a number of questions that we need to answer:

- Which data will be stored in this nameservice?
- Which namespaces does this nameservice use?
- What are the consistency requirements for the data?
- How will the consistency requirements be met?
- Which nameservice is authoritative for data with a strong consistency requirement?
- How is the data stored in this nameservice accessed?
- What are the limits of those access methods and how do they scale?
- What is the data capacity limit for this nameservice?

In addition, nameservices are typically services that many other services rely on. Therefore the reliability of nameservices is key to the reliability of the whole environment and all other services.

40.1.1 Data

Nameservices are all about the data. As described in [Section 39.7.1](#), the scope policy for a namespace defines in which nameservices it will be used. The various nameservices associate additional data with the names in that namespace. When setting up a nameservice, you need to define which attributes this nameservice will associate with names in that namespace. You also need to define which attributes are mandatory and which are optional, and how the attributes will be kept up-to-date.

A nameservice that provides incomplete, incorrect, or out-of-date data is not very useful. You should monitor the completeness of the data. If there is an authoritative source for some of the data, you can monitor the accuracy, but it would be better either to automatically update the data in this nameservice from the authoritative source, or to answer queries for that data by retrieving the answer from the authoritative source.

40.1.2 Consistency

As discussed in [Section 39.7.2](#), a namespace should have a consistency policy. When two or more nameservices associate the same attribute with names in a given namespace, the policy may state that this attribute should have high consistency. When implementing these nameservices, you also need to implement mechanisms that comply with the consistency policy.

For example, if multiple authentication nameservices exist, each of which associates a password with a name from the username namespace, then you might implement a password-changing mechanism that changes the password in all those nameservices at once. Or, if the CMDB contains information about the FQDNs and IP addresses associated with a machine, but that information is mastered out of the DNS nameservice, you might have the DNS system update the CMDB through a custom interface.

The ultimate consistency comes from having a nameservice that is the single source of the data. Other nameservices can have interfaces into this master nameservice. The nameservices will be completely consistent if queries to another nameservice are answered by that nameservice querying the master nameservice. Another option is for the other nameservice to have a copy of the

relevant portions of the data in the master nameservice. With this approach, the copy then needs to be kept up-to-date, via either realtime updates or a periodic synchronization.

40.1.3 Authority

When two nameservices contain overlapping data, you need to define which one is the authority for that data. That data includes the list of names in the namespace itself.

For example, the Unix authentication database, the Windows authentication database, the email system, and the corporate directory may all contain the list of usernames in the company, with a mapping between each username and the corresponding real name. If that mapping is different in each of those nameservices, which one is right? And more importantly, if the list of usernames differs, should you add the missing usernames into the services that do not have them, or should you remove those usernames from the ones that do?

For each namespace, you must define which nameservice is the authority for that namespace. Then, for each nameservice using that namespace, define which nameservice is the master for each attribute that is associated with the names in that namespace.

For example, you might make the HR database the authority for the username namespace. It is the authority for who the employees and contractors at the company are, so it makes sense for the HR database to be the authority for most, if not all, personnel data. However, it probably does not make sense for the HR database to be the master database for home directories, passwords, or application authorizations. Different nameservices will be the authority for those data, even though the username portion of those nameservices is mastered elsewhere.

Having a clearly defined source of authority for each set of data means that you can clearly define how to handle data conflicts: The data master wins. So, for example, if the HR database indicates that a person is no longer in the company—his or her username is no longer in that namespace—then the other nameservices should remove that username from their nameservices. In other words, they should remove that person's access. Or, if the HR database indicates that someone's name has changed, then the other nameservices should be updated accordingly. Namespace data should not be treated as a democracy, but rather as a set of dictatorships.

40.1.4 Capacity and Scaling

As with any service, you need to understand how the nameservice will be accessed, and what its capacity limitations are. For example, if the nameservice is a database, and it is accessed using SQL queries, how many queries per second can it handle? Does that number depend on the type of query? How much data can it hold before it reaches system capacity limits, or the quantity of data slows down performance significantly? When you need to add capacity, either for the queries per second rate or the size of the databases, how can you scale up the service?

For a DNS service, you may specify that it can be accessed using standard DNS requests, but not zone transfers. You should benchmark the DNS servers to understand how many queries per second they can handle, and document how to scale the various components. The resilience techniques discussed in the next section can also be used for scaling, in many cases.

40.2 Reliability

The first and most important thing to consider when setting up any specific nameservice is that it needs to be highly reliable. Many other services rely on nameservices, so a failure of a nameservice can cause widespread failures that cascade through other services.

A reliable service must be resilient to failures. Different types of nameservices use different protocols, but in each case there are resilience features available. You just need to take advantage of them. This section examines the most common nameservices, and goes into some detail on the resilience features that are available and appropriate for those services. In particular, we will look at DNS, DHCP, LDAP, authentication, AAA, and, briefly, databases.

40.2.1 DNS

Domain Name Service (DNS) is used by nearly every client and service that exists. Clients and application servers make continuous streams of DNS queries, mostly unbeknownst to the end users. Any brief outage in DNS is instantly visible to a lot of people. So how do we make it resilient to failure?

The DNS service is made up of several components. In the context of this section, we use the term “client” to refer to DNS clients, whether they be workstations, servers, or any other kind of device. We use the term “server” to refer to a DNS server.

Client machines are configured with one or more **DNS resolvers**. These are

the IP addresses that the client should use to send queries to DNS servers. Clients will typically query the resolvers in the order that they are listed. They query the first one, and if they do not get a response within a certain period of time, they query the next. Having multiple resolvers configured on each client is one element of resilience for the DNS service. The DNS resolvers are set by DHCP for DHCP clients. Devices that have statically configured IP addresses can also query DHCP for the network settings, including the DNS resolvers, but frequently the DNS resolvers are also statically configured.

Anycast

One of the challenges with the DNS service is that the client configuration contains IP addresses, which can make it difficult to move the service to another machine, and difficult to scale without touching every client, or at least many clients. Set up the DNS servers to advertise anycast IP addresses, and to respond to queries on the anycast address. Then configure the clients to use the anycast IP addresses as their DNS resolvers.

An **anycast address** looks like a normal unicast IP address, but the routers will route packets to *any* one of the machines that are advertising that address, using the routing protocol's metrics to determine the best route. The DNS servers advertise the address into the routing table when they are up and the DNS service is running. Anycast is commonly used on the Internet by ISPs providing DNS resolver service to their customers, and by companies that provide large-scale DNS service, such as the DNS root and top-level domain servers. But it can also be used by enterprises on their private corporate networks.

Other than making the DNS servers' IP addresses independent of the client-side configuration, anycast allows for much faster failover for the client. Rather than waiting for the first DNS server to time out, and then trying the second DNS server, the clients' requests are automatically rerouted to the closest server that is up. [Figure 40.1](#) shows an anycast DNS server in each region, all advertising the same anycast address. Clients go to the nearest anycast server, but fail over to the others transparently when the closest one is down.

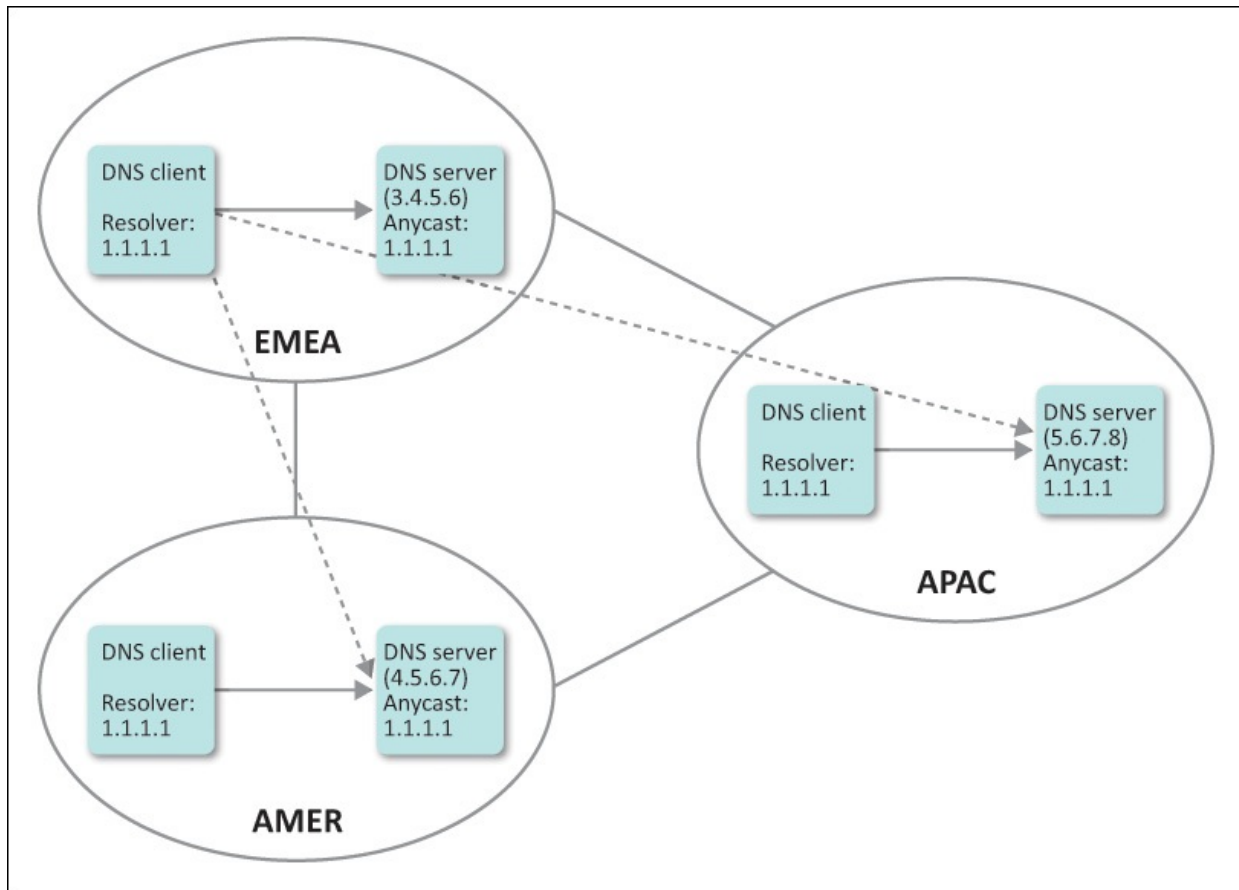


Figure 40.1: Multi-region DNS anycast

In addition, anycast makes it easier for the SAs to scale the DNS service. They do not need to get a new DNS server IP address into some set of the client configurations, but rather just need to add a new DNS server at a different point in the network, and have it join the anycast group. The new server will automatically draw the traffic from the closest clients.

Multiple Authoritative and Root Servers

The servers to which clients send DNS queries are called resolvers or **DNS caching servers**. These are the servers that respond directly to clients. They are configured with a set of **DNS root servers**, and do not need any information other than that, although they can have more complex configurations. DNS caches follow referrals in the DNS name hierarchy to find answers on behalf of clients. They learn which servers respond quickly, and which slowly, and automatically gravitate toward the faster ones. This makes the DNS service self-healing. When some of the root or authoritative servers fail, the DNS caching servers automatically switch to the ones that are still up. When they return to service, the traffic will automatically swing back when those servers are found to

be responding quickly again.

DNS Master

Authoritative DNS servers contain the DNS zone data. A DNS zone can, and should, be served by multiple authoritative DNS servers. Authoritative DNS servers can be set up in a master/slave configuration, where one server is the true authority for the zone, and the other authoritative DNS servers contain copies of the data, which are updated from the master. They can also be set up in a multi-master configuration. The difference is that one or more masters receive **dynamic DNS updates** to the zone. Dynamic DNS updates are used to automatically add and remove records—for example, for DHCP clients that are dynamically assigned IP addresses.

In the single master and multiple slaves configuration, you need to consider how to make the master resilient to failures. If the master fails, the slaves will continue to serve the DNS zone data, but dynamic DNS updates will no longer work. The resilience options available for the master depend on the server architecture. Failure of any of the slaves will be transparent to clients because of the self-healing nature of DNS.

In the multi-master configuration, you need to understand if it is a true multi-master, or if there is one real master and the others simply store and forward the updates. In the case of a true multi-master, you need to understand how the DNS database is healed after a network partition. Multi-master configurations typically cope well with the case of server failure—when the server comes back up, it resyncs with the other master and carries on. However, they typically perform poorly when all the servers stay up, but the network is partitioned so that the servers can no longer exchange traffic across the partition. In such a case, the servers on each side of the partition assume that the servers on the other side are down. All servers continue to receive and process dynamic DNS updates, so when the network is healed, the databases are out of sync. Each side has received and processed valid updates in the meantime.

You need to understand how the database gets healed. Does one side “win” and updates from the other side are lost? Or is there a more sophisticated merge? Or is manual intervention required? The CAP theorem for distributed databases is relevant here and is discussed in more detail in Volume 2 of this book series. In essence, it is often paraphrased as “Consistency, availability, and partition tolerance: pick two.” When choosing a multi-master setup, you need to understand which two were chosen, and what the implications of that choice are for you.

40.2.2 DHCP

The Dynamic Host Configuration Protocol (DHCP) is another nameservice. It is defined in Internet Engineering Task Force (IETF) Requests for Comment (RFCs) 2131 and 3074. Those RFCs define a failover protocol for a pair of DHCP servers. When setting up a DHCP service, you should always set up DHCP servers in pairs and take advantage of this failover feature.

When DHCP clients come onto a network, they broadcast their request for network configuration. On the client networks the routers pick up those requests. Each router must have both DHCP servers in a failover pair defined as DHCP relays for that network. Since the DHCP servers are defined on every subnet, it is not a trivial task to switch to another DHCP server when one fails. Using DHCP failover pairs means that the service keeps running even when one server is down.

A DHCP client receives a lease on an IP address, with an expiration date and time. DHCP clients that have a lease start trying to renew that lease after half of the lease time (technically the T1 timer) has expired, by directly contacting the DHCP server that gave it the lease. If that server is down at the time, it will not get a response. However, as the lease time gets closer to expiring, it will start broadcasting its renewal requests, allowing the failover partner, which also knows of the existing lease, to renew the lease. The client will give up the IP address when the lease expires unless it has received an acknowledgment of its renewal request.

The lease time is another factor in the reliability of the DHCP service. What matters to clients is not so much that the DHCP service is up and running, but rather that the clients have an IP address and are able to work. Short lease times mean that if the second DHCP server has an issue before you have had a chance to fix the first one, very quickly no clients will be able to function. With long lease times, new clients on the network will be unable to work, but existing clients will continue functioning for longer, resulting in lower impact. However, if you use long lease times on a network where clients remain for just a short time, this approach will not reduce the impact, because the majority of the clients will be new. In addition, you may run out of addresses on an IPv4 network if clients do not release their leases properly before leaving the network. IP address shortage is not an issue with IPv6, which has 340 undecillion (3.4×10^{38}) IP addresses, as compared to IPv4's paltry 3.7 billion.

40.2.3 LDAP

Lightweight Directory Access Protocol (LDAP) servers also provide a nameservice. It is commonly used for providing distributed directories for authentication databases; email, phone, and address directories; and distributed databases of systems, networks, services, and applications. It can be used to feed other nameservices, such as DNS. LDAP is defined in another IETF standard: RFC 4511.

Like DNS, the LDAP service can be made more resilient by configuring multiple LDAP servers into the client-side configuration. LDAP servers can also be made resilient by using anycast addresses to access the directories. Also as in DNS, the data in LDAP can be dynamically updated, and the SA setting it up needs to decide between a master/slave setup and a multi-master setup, with the same issues arising related to the network partition for the multi-master setup.

One common way to set up a resilient LDAP service is to place the master behind a load balancer, with all updates being directed to one of the servers. When that one is down, the updates are directed to a different server. When the first one comes back, it syncs with the new master. Partitioning is not an issue with this setup because the load balancer directs the updates to only one server at a time, so the database remains in a consistent state.

Microsoft's ActiveDirectory (AD) is based on LDAP in a multi-master setup, with all the domain controllers (DCs) being masters. It is also commonly used to provide DNS services, with the LDAP directory feeding the AD-integrated DNS service. For environments with a significant population of Windows devices, AD is a convenient, well-supported resilient directory service that is well integrated with most of the major services that an enterprise IT department needs to provide.

40.2.4 Authentication

Authentication is a key nameservice on which just about everything else relies. On the client side, applications should be able to work with as many different authentication mechanisms as possible, so that the SAs are able to choose one or more authentication services that work with all applications. Applications and operating systems that support pluggable authentication modules (PAM) are ideal, giving the SAs control over which authentication modules the application should use.

SAs may want more than one authentication service if there are different services that require different levels of security. For example, some internal services that do not provide access to sensitive data may require just username

and password authentication. In contrast, administrative access, or access to services that provide access to sensitive data, may require two-factor authentication. Thus the SAs would need two authentication services—one for password-based authentication and the other for two-factor authentication.

Some authentication services are built on LDAP, and resilience for LDAP was previously discussed. Kerberos is another protocol that is often used in enterprises for building secure authentication services. Sometimes Kerberos is used in combination with LDAP. Kerberos services are also integrated into Microsoft ActiveDirectory, for example. Kerberos provides for mutual authentication between the client and the server. It is a ticket-based system, in which clients and end users get tickets for accessing the various services.

Kerberos authentication services are controlled by a key distribution center (KDC). On the client side, the Kerberos domain, or realm, is configured, and this realm is strongly linked with the DNS domain. In fact, Windows won't accept the DNS domain and the Kerberos realm being different. The KDCs can be found by using DNS lookups for the Kerberos service in that domain. KDCs can be added and removed from the list of available KDCs using dynamic DNS updates. Multiple KDCs are set up using a master/slave configuration. The master, or primary, KDC is where changes to the database are made. Secondary KDCs have copies of the database, and can perform authentication checks on behalf of clients.

40.2.5 Authentication, Authorization, and Accounting

Authentication, authorization, and accounting (AAA, or Triple-A) is another core nameservice that most enterprises need to provide. The Remote Authentication Dial-In User Service (RADIUS) and the Terminal Access Controller Access-Control System Plus (TACACS+) protocols are commonly used for AAA services. Typically, AAA servers are configured to refer to other external nameservices so as to verify user identity and to obtain a set of authorization parameters.

AAA services are typically accessed by other services. For example, an enterprise might offer a service that requires a user to log in, and then grants permission to certain parts of the service based on what that user is permitted to access. This service should use a AAA service to perform the authentication against the enterprise-standard authentication systems (such as ActiveDirectory), and may be configured to query, for example, a centralized SQL database that is used for defining user authorizations for all services. That database returns the service-specific group or role that the user is in, which tells the service what the

user is authorized to do.

The clients of AAA services, therefore, are other services. Each of those other services should be configured with multiple AAA servers, for resilience. Each external nameservice that the AAA servers access should be resilient. Moreover, the AAA service, which is a client of those services, should be configured appropriately so that it takes advantage of the resilience features of each of those external nameservices, as shown in [Figure 40.2](#).

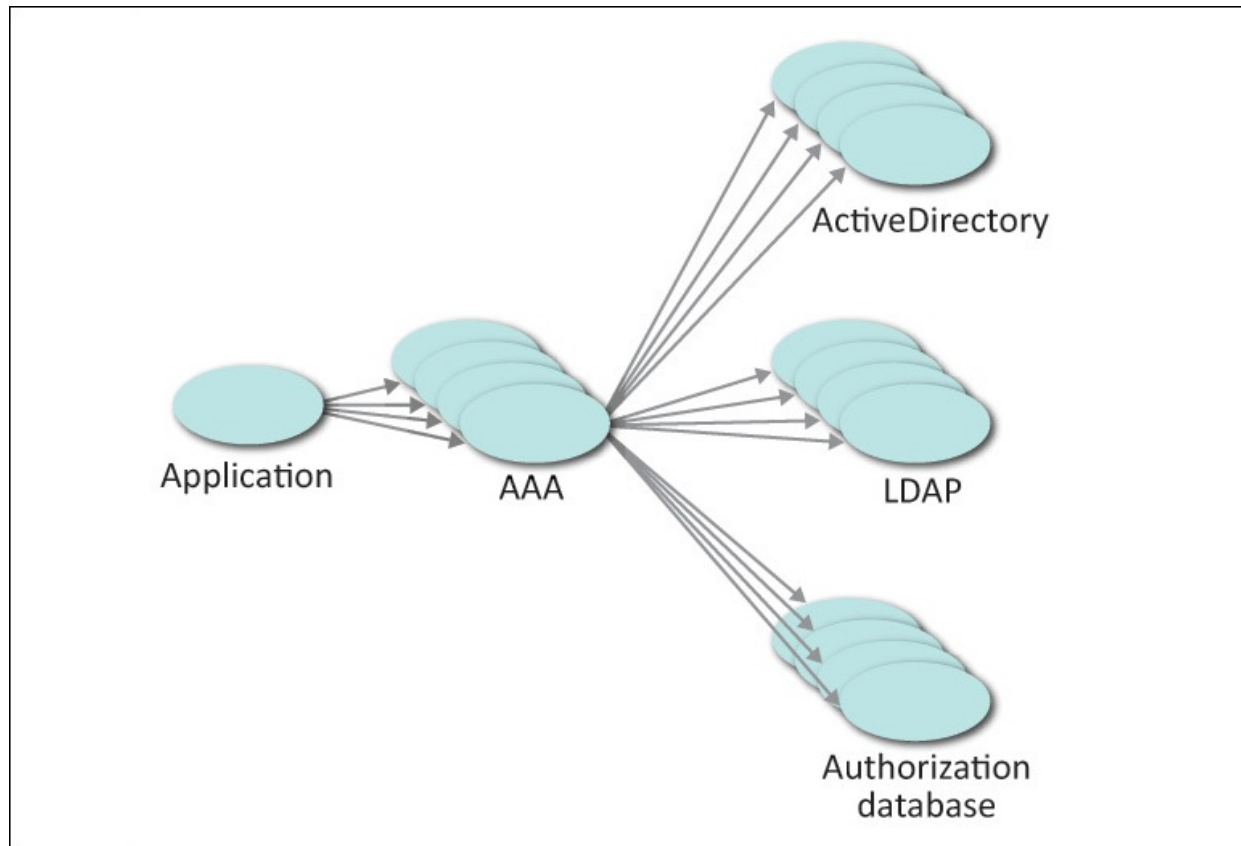


Figure 40.2: A resilient setup for AAA services

40.2.6 Databases

Databases, such as the HR database, CMDB, and centralized authorization database, also provide nameservices, and are often the master databases that drive changes in other nameservice databases and other systems. For example, when someone leaves the company, a change in the HR database that marks this person as no longer an employee should result in automatic changes in all the other directories: the authentication services; the email, phone, and address directories; and the authorization database. As another example, a change in the inventory system that marks a system as decommissioned should release all software licenses reserved by that system, cause DNS entries to be removed,

cause any firewall rules specific to that system to be removed, create tickets for physical decommissioning if necessary, and so on.

Because databases are frequently critical components of organizations, all enterprise-class databases have reliability features. Nondistributed, single-server databases often rely heavily on the reliability features of the underlying server infrastructure. Sometimes they have some custom high-availability clustering features built in. By comparison, distributed databases are typically designed to be highly available even when built on unreliable infrastructure. Volume 2 of this book series goes into more detail on the question of tradeoffs among consistency, availability, and partition tolerance (CAP) for distributed databases, including the CAP theorem, and explores which attributes the various distributed databases that are available today have.

40.3 Access Policy

A nameservice's access policy is typically determined by the classification of the data in that nameservice. It should answer the following questions:

- Which kind of protection or security does this nameservice require?
- What are we trying to protect the records from, and why?
- Do the names in the space need to be protected, or just their attributes?

From whom should the contents of a nameservice be protected? It depends on the nameservice. Everyone should be prevented from reading confidential data in a nameservice, such as a list of encrypted or plaintext passwords associated with user IDs. However, all users of a system may be permitted to be able to map user unique identifiers to usernames to real names, and all employees and contractors in a company may be permitted to look up names in the company directory to get office locations, phone numbers, email addresses, and so on.

The answer can be different for each namespace and may also depend on context. For example, individual lookups on the data contained in the DNS nameservice are essential to the operation of almost all computers, services, and applications. However, a complete dump of all that information can be used to facilitate an attack against the company. Thus, a company would permit DNS queries from everyone, but not zone transfers. Another example is deciding that the individual login IDs of a Unix system can be safely exposed in the outbound email message, on business cards, in advertisements, and so on. However, the complete list of IDs shouldn't be exposed externally, because spammers will use the list for sending junk email.

A company should have formal data classifications, and educate people about

those classifications and what they mean in terms of how the data should be protected. Data owners need to classify their data, and these data classifications should be tracked, usually in a central database that tracks applications, software packages, licensing, and so on.

For example, a company might have four data classifications: public, internal, confidential, and strictly confidential. Marketing material and press announcements would be classified as public, and can be made available to anyone inside and outside the company. The list of company employees, organizational charts, the list of machines and their IP addresses, and so on would be classified as internal data. This data is freely available to people inside the company, but should be exposed to those outside the company on a need-to-know basis, with the minimum required data being exposed. Confidential data would include customer lists, source code, and other data that should be restricted to certain employees and never exposed without appropriate approvals and nondisclosure agreements. Strictly confidential data is the most closely guarded data. In a software company, this would be data that has been received from third parties under a nondisclosure agreement. In banks where there are privacy regulations governing client confidentiality, this would be all customer-identifying information, such as names, addresses, and phone numbers. In the medical industry, patient information is also strictly confidential and its exposure is highly regulated.

For a data set that contains records with different classifications, the highest classification (the one requiring the most protection) should be used for the data set as a whole. Classifying data in this way, and defining rules about data protection that are based on the classifications, makes it easier for everyone to understand and follow the rules.

To continue the DNS example, most companies have only a limited set of DNS names that they want to expose to the Internet, so they maintain two separate DNS nameservices: one that is accessible from the Internet and has data classified as “public,” and one that is accessible only from within the company and has data classified as “internal.”

Printing *etcpasswd*

Tom observed an SA debugging a Unix printing problem. Every time he needed to print a test page he selected a short file, which happened to be *etcpasswd*. Soon dozens of copies accumulated in the wastebasket.

While modern Unix systems do not store actual password information,

this was 1995 and the printouts included enough information to recover most of the passwords using a simple brute-force attack. Even if that wasn't an issue, exposing the usernames of the system was bad practice.

The SA didn't realize that trespassers were often caught dumpster diving to find just that kind of thing.

The lesson learned is that you shouldn't print sensitive information. If for some reason you do, the printout should be shredded before it leaves the building.

40.4 Change Policies

All nameservices need policies for how, when, and by whom the records in a namespace can be changed. Often, different fields in a record will have different policies. For example, a user would be able to change his own password, but not the location of his home directory. A nameservice's change policy should answer the following questions:

- Who can add, change, or delete records?
- Do different fields in the records have different change policies?
- Can the owner of a record change certain fields within the record?

Changes to a nameservice include adding records, removing records, and changing fields in existing records. Within a typical company, only specific people are able to create or delete records in the user ID namespace. However, all individuals may be able to change certain settings in their own records but not others. In contrast, ISPs often have a policy that anyone can create an account, based on the person's ability to provide a credit card number; deleting an account also is customer initiated. Universities often have systems in which professors can create dozens of accounts at a time for the students taking a particular course that will require access to certain machines. However, non-SAs likely cannot create a privileged account or delete other people's accounts. A major web-based email provider also doesn't have a procedure to delete accounts on request, but an account that is not used for a certain period is automatically deleted.

Another facet of protection includes change control and backup policies. Change control is discussed in [Chapter 32](#), “[Change Management](#).” Here, suffice it to say that it is important to be able to roll back a change. Namespaces that are stored in plaintext format can be checked into a revision-control system, such as Git, Mercurial, or Subversion under Unix or SourceSafe under Windows for

network-based management. Backup policies should pay particular attention to namespaces. Backups are the ultimate insurance policy against accidental deletions and changes.

How a namespace is protected from modification is another issue. Sometimes, a namespace is maintained in a flat text file, and modifications are prevented by using the appropriate file-permission controls. In other instances, modifications are done through a database that has its own access controls. It is important to remember that a namespace is only as secure as the methods that can be used to modify it. The method used to update a namespace should be more secure than the systems that depend on that namespace for security. For example, is the method that is used to access and update the namespace encrypted? Is it authenticated through a secure mechanism? If only a password is required, you face a significant risk of someone making unauthorized changes.

40.5 Change Procedures

All namespaces need procedures for additions, changes, and deletions. These procedures should be documented just as the policies are, but the documents may not be available to customers. A small group may be able to operate without these procedures being explicitly written down. The procedures are performed only by the people who invented the system and thus do not require documentation. However, as the system grows and new SAs are hired, confusion is likely to set in. Documentation can serve the dual purposes of providing a basis for training and step-by-step instruction when the task is being performed.

If something can be documented in a clear, concise manner, it can be automated, or a tool can be created to perform the task. Not all tooling or automation needs to be complicated, as we will see in the next section. A tool is a script that someone can run to perform a task. It may be accessible through a web interface, or just from the command line. Automation builds on the tooling so that a person does not need to trigger the tool to run.

For example, an SA may create a tool to be used when a new employee comes on board. This tool might create login accounts, email accounts, and home directories, and perhaps add the user to some email lists and authorization groups based on the employee's department. The voice team may create another tool that configures the phone and voicemail box. Likewise, the security team might create a tool that configures card-key access to let the new employee into appropriate buildings or parts of buildings. When automation is applied, all these tools are triggered automatically when HR adds the new employee into the HR database. For companies where these systems all support LDAP, the LDAP

database becomes the central integration point from which all services are provisioned.

40.5.1 Automation

Automation has many benefits. Automated tasks are responsive, are repeatable, and require minimal or no human intervention. They don't make typos or skip a step when they are tired or rushed. Automation and tooling increase customer satisfaction, because they provide a quicker response, and they make the SA's job more interesting—the SA focuses on maintaining the automation and tooling rather than making lots of repetitive small manual changes. This approach also gives the SAs more time to work on projects because they are not being constantly interrupted to perform small but urgent tasks such as password resets.

Automation can link the HR database to the authentication system such that when a person's employment is terminated, his or her accounts are automatically disabled. This relieves the SAs of being involved in the exit process and having to ensure that all access is disabled in a timely manner. In the case of a potentially hostile termination, it means that HR is completely in control of the timing of notifying the employee and disabling the access, and not reliant on being able to get hold of the SAs. This can protect the company from potentially costly deliberate damage from a disgruntled employee.

A properly maintained CMDB is central to building automation for your fleet of workstations and servers. In [Chapter 4](#), “[Infrastructure as Code](#),” we saw how it can be part of an infrastructure as code (IaC) system. In [Chapter 9](#), “[Workstation Service Definition](#),” [Chapter 10](#), “[Workstation Fleet Logistics](#),” and [Chapter 33](#), “[Server Upgrades](#),” we saw it drive the software installation, patching, and upgrade processes. [Chapter 26](#), “[Running a Datacenter](#),” showed how it can be used to improve datacenter operations.

Leveraging the Inventory

Tom once worked at a site that had an inventory database that could be easily queried from shell scripts. The site also had a program that would run the same command on a list of hosts. Putting them together made it easy to write a program that would make changes globally or on hosts with specific characteristics, such as OS, OS revision, amount of memory, and so on. When a new application was deployed, a query could quickly determine which hosts would need additional memory to support it.

Driving maintenance processes off a database of hosts was very powerful.

40.5.2 Self-Service Automation

Automation can be used to make customers self-sufficient. While most namespace-related requests require administrator privileges, it is possible to create automation that permits a restricted set of requests to be done by the users themselves. The tool can validate input and prevent abuse, but otherwise permit common requests to be done.

For example, many companies lock out user accounts after a certain number of failed login attempts, or after a password expires. Self-service automation could enable a person to unlock the account with the assistance of a colleague who can vouch for the person. Other examples of self-service automation are provisioning of access rights, mailing list creation and maintenance, and updating inventory data.

Case Study: Account Creation at Rutgers

At Rutgers University, many of the computer science classes required that accounts be created for every student on machines in the computer science cluster. Tooling was provided for mass-producing accounts. Teaching assistants had access to a command that would create a large number of accounts for a class by querying the enrollment database. A similar command would delete all the accounts for a particular class when the course ended. As a result of this self-service tool, the SAs didn't have to be involved in maintaining this aspect of the namespace—a major win at an academic institution!

40.6 Centralized Management

Nameservice management should be centralized as much as possible for any given environment. With centralization comes consistency. Otherwise, related nameservices may become scattered around various servers or even different directories on the same server. It is better to have a single host maintain your namespaces and have them distributed to all other hosts. Most sites will already have a centralized configuration management tool. This tool can also be used to maintain the master copies of a site's namespaces, and to distribute (portions of) them to all other hosts. Centralized namespace management also means that tooling and automation are easier to implement and have a larger benefit.

Case Study: A Namespace Cleanup

At one site Tom inherited a Unix cluster that used NIS and a few other

technologies to provide many namespaces to all machines in the cluster. Each namespace had a different update procedure. For most, the process was to update a data file and then run a script that would pick up the changes and push updates to all machines. The scripts were named inconsistently: `update_printcap` for printcaps but `push_aliases` for aliases. For some namespaces, there was no script to run. For some namespaces, the script pushed multiple namespaces.

Part of Tom's indoctrination at this site was to be taught, for every namespace, in which directory it was stored, on which machine, and which script to run to push any changes to the other machines. All this information was taught from memory by a senior SA, because none of it was documented. An SA's competence was judged not on how much Unix knowledge he or she had but on how well the person could memorize a list of oddly named scripts.

By the end of the week, Tom had assembled all this information into a large chart. Tom doesn't have a very good memory, so this complicated system concerned him greatly.

None of the files were stored under a version control system such as Git, RCS, or SVN, so SAs had to rely on tape backups for any kind of revert operation. The SAs couldn't roll back any change without a lot of effort. Tom has his error-prone days, so this also concerned him.

After a month of effort, the master file for every namespace had been moved to a single directory on a single host. These files were maintained under version control so that changes could be rolled back. A new `Makefile` in that directory fired off the appropriate legacy script, depending on which file had been changed. After this transition was made, the system was not only easier to administer but less error-prone. This made it easier to train new SAs, who could then spend more time working and less time remembering oddly named scripts.

Once the new system was stable, it could be optimized. Some of the legacy scripts were brittle or slow. With the system consolidated, there was time to focus on replacing the scripts one at a time with better ways of doing things.

This consolidation also made it easier to introduce new automated processes, such as creating and deleting new accounts. Rather than requiring SAs to remember the names of scripts to create an account, `make help` would list all the scripts for them to choose from.

`create_account` and `disable_account` eliminated the two most complex and error-prone tasks. Running either would ask the appropriate questions and complete the task.

Some sites centralize all namespaces into a SQL database as their single source of truth. They then use this database to feed other systems, such as ActiveDirectory, LDAP, NIS, printer configurations, and so on. Jon Finke of Rensselaer Polytechnic Institute has written several papers on this topic, which can be found at

<http://homepages.rpi.edu/~finkej/Papers.html>.

40.7 Summary

In this chapter, we looked at how namespaces are used in nameservices. Nameservices are all about the data.

Nameservices are part of the fundamental infrastructure of a computing environment. Reliable nameservices are key to the overall reliability and availability of the whole environment.

Nameservices need access and change policies, along with documented change processes. The policies must be established before the procedures, because the policies should drive the procedures. Nameservices can greatly benefit from central management and automation.

Automation is required to do a good job of maintaining nameservices, and some nameservices such as the employee database and the CMDB can be used to drive automation in other nameservices and other systems and services. Centralization is key to automation.

Exercises

1. In your environment, identify all the nameservices that you can think of. What are the namespaces that are used by each nameservice?
2. Which additional data does each nameservice in your environment associate with the names in the namespace?
3. How reliable is each of your nameservices? How do you measure that reliability? Which resilience mechanisms are in place?
4. What are the access and change policies for each of your nameservices?
5. A company has public and internal DNS services. Describe the DNS solution you would propose for a private extranet connection to a third

party, where the third party has access to a limited number of internal services.

- 6.** What are the change procedures for each of your nameservices?
- 7.** In your environment, which automation is in place for maintaining the nameservices? Which automation should be created?
- 8.** Which nameservice maintenance should be pushed down to your customers through automation?
- 9.** How is each nameservice in your company managed—centralized, or not? Which challenges does this model present?

Chapter 41. Email Service

Companies depend on email to conduct business. Everyone expects email to simply work, and outages are unacceptable. It is often the only application that your CEO uses. Your CEO's impression of the reliability of email can have far-ranging effects on other aspects of the SA team, such as its budget and reputation.

Email servers store a significant amount of business-critical information. For many companies, email is the primary way they interact with clients, business partners, and other employees. Email needs to be reliable, available, and fast.

This chapter is about the key features and design elements of email services. It is not a how-to guide for running Microsoft Exchange or other systems. Instead, this chapter covers policies and principles that apply to all corporate email systems.

A reliable, scalable email service must be built on strong foundations. An SA who is designing and building an email service must put the basics first, before trying to add features or scale the service to deal with high traffic volumes. Email systems require well-managed namespaces ([Chapter 39](#)), reliable nameservices ([Chapter 40](#)), service monitoring ([Chapter 38](#)), and backups ([Chapter 44](#)).

A simple, clear, well-documented architecture for the email system is fundamental to building a reliable service. It also is important to use open protocols and standards throughout the email system to ensure maximum interoperability with other sites and other applications within the site. In particular, one key piece of infrastructure that the email system needs to interact with is the namespace management system, which implements the organizational structure of the corporate namespaces that relate to email. Recall from [Chapter 38](#), “[Service Monitoring](#),” that nothing should be called a service until it is monitored.

Email is a method of communicating with the rest of the world, so some parts of the service will always be a target for potential attackers. Security must therefore be considered during the design and implementation of the email system.

Finally, because email contains so much vital information, email privacy, retention, and storage must be examined and brought into compliance with company policy and with any legal requirements.

Migrating to a New Email System

Moving hundreds or thousands of users to a new email system? Don't do it all at once. [Section 20.6](#) has an example about how such a plan can go badly. [Chapter 21](#), “[Service Conversions](#),” has general advice about making large service transitions, in particular [Section 21.7](#).

41.1 Privacy Policy

Every site must have an email privacy policy that is communicated to and acknowledged by every employee. The privacy policy must explain under which circumstances a person's email may be read and by whom. The policy must also explain that email may be inadvertently seen by administrative staff during the course of their work, usually when performing diagnostics. The policy should also state that email that crosses over other networks, such as the Internet, cannot be considered private and that company-confidential information that has not been encrypted should not be emailed to an address on or across another entity's network.

At many companies, email that arrives on or crosses over corporate servers is not considered private. Other companies state that corporate machines should not be used for personal communication, which typically amounts to the same thing. Some companies automatically monitor incoming and outgoing mail for certain keywords or confidential information such as Social Security numbers and other identifiers. Others state that people using the corporate email system have a reasonable expectation of privacy and outline the circumstances in which an expectation of privacy may no longer hold.

Whatever the policy decided by the upper management of the company, the SAs must implement it, and make users aware of it.

41.2 Namespaces

Namespaces are discussed in [Chapter 39](#), “[Namespaces](#).” Here, we focus on the email namespace. The email address namespace at a site is the most visible namespace both externally, to the company's customers and business partners, and internally, to the employees who use it daily. It is crucial to get it right.

The most fundamental part of getting the email namespace right is to use the same email addresses for internal and external email. If one address is used for internal mail and a different one for mail going outside the company, people will inevitably give the wrong email address to customers and business partners,

which can lead to lost business. Don't expect people to remember that they have two email addresses and know which one they should give to whom. It is far simpler for all concerned, including the SAs debugging problems, if everyone has just one email address for both internal and external email.

Standardizing email addresses by using a `first.last`-style email address, such as John.Smith@foo.com, is popular, particularly with management. However, there is too much of a chance that your company will employ two people with the same name (or even same first name, last name, and middle initial). When your second John Smith is hired, `John.Smith` becomes `John.A.Smith` to avoid being confused with the new hire, `John.Z.Smith`. At that point, the first person's business cards become invalid. Business cards, once distributed, are difficult to update.

Some email systems deal with ambiguous `first.last`-style addresses by generating an automatic reply that tries to help the sender figure out which "John Smith" he was trying to reach, possibly listing the first ten matches in the corporate directory. Although this sounds nice, it is not a perfect solution. The replies are useful only if a human receives them. If the person was on any email mailing lists, those messages will now bounce.

Eric Allman brilliantly explains why this kind of formatting is problematic in the `README` file of Sendmail:

As a general rule, I am adamantly opposed to using full names as email addresses, since they are not in any sense unique. For example, the Unix software-development community has two Andy Tannenbaums, at least two well-known Peter Deutsches, and at one time Bell Labs had two Stephen R. Bournes with offices along the same hallway. Which one will be forced to suffer the indignity of being `Stephen.R.Bourne.2`? The less famous of the two, or the one that was hired later?

Instead, try making a namespace that is unique corporation-wide, with usernames such as `chogan`, `tal`, `strata`, and so on. A directory service can be provided to help people look up the email address of the person they want to contact. Customers should be able to select their own token, though an initial default should be preselected, based on an algorithm that combines initials and first or last names. It should be relatively difficult to change once it has been set, to discourage people from making gratuitous changes. Tokens should not be reused for a couple of months, to prevent someone from hijacking the token of a recently removed employee to see which residual email he or she receives.

41.3 Reliability

Email is a utility service. People expect to be able to send and receive email at all times, just as they expect to always have a dial tone when they lift the phone and power when they turn on a light. As with other utilities, people don't realize how much they rely on it until it is no longer working.

A failure of the email system is a very stressful event and results in lots of support calls over a short period. It will inevitably occur when someone has to urgently send important documents, because that is happening all the time, unseen by the SAs. Because the email system is so critical, failures will be emotional times for both the SAs and the customers. The email service is not a service with which SAs should experiment. New systems and architectures should be deployed into the email service only after extensive testing.

More important, costs are associated with a malfunctioning email system. Missed email can cause business panic. Contractual obligations are missed, customers turn to other vendors, and time is lost as people revert to older, slower forms of communication.

Case Study: The Email Service Beta Test

A major technology company was promoting the concept of centrally located server farms for business applications such as email. As a proof of concept, the company quickly moved all 100,000 employees off its local, department-based servers to a single, global server farm. The network connections into the server farm were overloaded to the point of being unusable. As a result, the entire company was unable to communicate for nearly a month until more network capacity was added. Not one to learn from its mistakes, the company then decided to use this server farm to demonstrate a new release of its email software.

This may have been the single largest beta test ever attempted. It was also a disaster. Eventually, the company learned to not take such risks with such a critical application. That realization came too late, however. By this time, many organizations had created rogue email servers for their local members, making the situation worse. Your corporate email system is simply too critical to use as a playground or experimenter's lab.

The national power grids and telephone networks of all developed countries are highly redundant systems, designed and built with reliability in mind. The design of an email system should have a similar focus on reliability, albeit on a

smaller scale.

Start with a clear, simple design. Select hardware and software for their reliability and interoperability. Data should be stored on RAID systems to reduce outages.

Having hot spares for all the email machines is ideal, although many companies cannot justify that expense. If you do not have the luxury of hot spares, have a plan that you can execute rapidly to restore service if anything fails.

41.4 Simplicity

The email system should be simple. Complexity decreases reliability and makes the system more difficult to support.

Limit the number of machines involved in the email service. That limits the number of machines that have to be reliable and the number of places SAs must look to debug a problem. Above all, do not involve desktop machines in the mail-delivery process. While Unix desktops can be configured as email servers, they should not be. They usually do not have reliable backups, power, or other things that a server should have.

An email service has five main parts:

- **Mail transport:** The mail transport agent (MTA) gets email from place to place, usually from server to server.
- **Mail delivery:** Mail delivery agents (MDAs) receive email messages from the MTA and store them at the destination server.
- **Access:** Email access servers provide the access protocols (POP3, IMAP4) that allow a mail user agent (MUA) on a customer workstation to access individual emails.
- **List processing:** List processing is how one message gets delivered to a group of people on a list.
- **Filtering:** This component consists of antispam and antivirus filtering.

For small sites, a simple architecture typically means having all these functions provided by the same machine, possibly with an additional Internet-facing mail-relay system being the interface between the company and the rest of the world. For larger sites, simplicity often involves separating mail transport, mail delivery, and list processing onto different systems or groups of systems. Several dedicated mail relays will deliver mail to either the list-processing machines or the delivery machines. The list-processing machines also use mail relays to

deliver messages to the individual recipients on each list. At a large site, several machines may be involved in email service.

Typically Unix systems run some kind of email system by default since certain systems generate email messages. Such machines are usually configured to relay all locally generated email to the main email servers. If this is not done, such messages are delivered to the local machine and may be thought to be lost, though they are sitting on machines around the network.

Speaking of confusion, hosts that generate email from service accounts such as `root` should be configured to label the sender as being specifically from that server, and the central email system should not rewrite the sender's address unless it is a customer's account. If you have a 500-server web applications cluster, and periodically get email from `dbadmin` about a database access error, it would be good to know immediately which server had the problem rather than having to peek at the `Received` headers.

Case Study: Bad Mail-Delivery Scheme

A computer manufacturer's mail scheme permitted email delivery on any Unix machine on which the recipient had an account. This scheme also exposed the full hostname of the machine from which the person sent the email. For example, email sent from `server5.example.com` had a `From:` line of `user123@server5.example.com`. Whenever someone replied to one of these emails, the reply was addressed to the person at the machine from which the mail had been sent, so it was delivered there rather than to the person's primary email box. Thus, when someone sent an email from a machine that he did not normally use, the reply went to that machine and was "lost." If that address made it into someone's address book and the machine was decommissioned, email would start bouncing. The helpdesk frequently received complaints about lost email, but because the email system was so unstructured, the staff had a difficult time finding out what had happened.

The SAs should have implemented an email system that passed all mail to a central relay that rewrote the sender's email address so that it did not contain the name of the machine that it came from. This practice is known as hostname masquerading. With such an approach, all replies would automatically go through the central mail relay, which would direct the email to each person's primary email box.

Simplicity also means avoiding gateways and other email translation devices. Use the same standards throughout the network and for communication with other sites. Gateways translate email between two or more different formats, often between a proprietary or nonstandard format or protocol and a standard one. Gateways add complexity and are often the source of endless problems at sites that use them. Gateways also typically strip off delivery-history information because it is in a different format, which makes it more difficult to trace problems. See [Section 16.7](#) for reasons you should avoid gateways.

It's better to provide a single mechanism for implementing and managing email lists. There are many different ways of doing so, and sites that have been around for a few years typically use more than one. This lack of standardization makes it more difficult to maintain the lists and much more difficult to implement automated list maintenance and pruning mechanisms. Forcing customers to learn and remember multiple email procedures is unacceptable.

Large sites with multiple servers need to be able to shift people's email boxes between servers to compensate for load imbalances. That is, if a server becomes overloaded, some of the mailboxes are moved to a less-loaded machine. When this happens, each person's email client has to be reconfigured to point to the new machine. One way to avoid that problem is to set up DNS entries for each user in the form of `username.pobox.example.com`, which is an alias to the username's current email server. If the mailbox later moves to a different machine, the DNS data is changed and the client does not need to be reconfigured. (If you provide web-based email readers, use the same DNS name to redirect HTTP requests to the proper web server.) Another strategy is to use an MDA that supports file locking with your MUA clients. These multiple mail access servers can be used to access the same account without having to move mailboxes.

41.5 Spam and Virus Blocking

Electronic junk mail is also known as spam or unsolicited commercial email (UCE). It is common for more than 50 percent of email coming from the Internet to a mail server to be UCE. Since the late 1990s, email has also been a dominant way for computer viruses and other malware to copy itself from computer to computer.

Both spam and malware can be blocked at either the email client or the server. Typically it is best to block them at the server. It is easier to upgrade the server than to upgrade hundreds of email clients. Customers also tend to easily get tricked into disabling the protection software on their clients. Some customers

simply disable it because they think the system will run faster.

There is no perfect antispam software. How can a computer know that you don't want a particular piece of email? There are rare cases in which an incoming email message matches, bit for bit, a known spam message. However, that technique doesn't work on spam that hasn't been seen before. The same is true of antivirus software. Although it is possible to detect email that contains a known virus, it's nearly impossible to detect a newly unleashed one. Thus, blocking bad email has become an arms race. Spammers and virus spreaders figure out a way to get through traditional detection software, then detection-software vendors improve their software, then the attackers improve their techniques, and on it goes.

Centralizing this function to a particular set of machines is important. In fact, there are now services that will do spam analysis for you. One directs email to its service, using the DNS MX record, where it is processed and then delivered to your server. There are privacy concerns with such a service, however. Of course, if the email was secret, it should never have been sent unencrypted over the Internet.

41.6 Generality

One of the fundamental reasons for the existence of an email system is to open communication paths within and outside the company. To successfully communicate with the maximum number of people, the email system should be built around open protocols that are universally accepted and implemented.

For mail transport, this means using an SMTP-based protocol. Because mail transport involves communicating with many other sites, this well-established Internet standard will continue to be supported for a long time to come. SMTP is extremely pervasive on the Internet. It is more likely that a new email protocol would be an extension of SMTP such as the widely used ESMTP, or extended SMTP, rather than a completely new and incompatible protocol. All sites must support SMTP as a transport protocol; for simplicity and generality, it should be the only transport protocol that a site uses.

Generality is also about communications within the company and mail user agents. It applies to the methods that are available for people to read their email. The number of protocols in this area is a little larger and changes more frequently with time because it is usually easy to add support for an additional email client/server protocol without affecting the existing ones. Most sites support one or two of the most popular email client protocols, thereby

supporting almost all mail readers. A site can trivially support lots of mail clients on lots of different OSs if the mail delivery and relay systems use Internet-standard protocols. Supporting the maximum number of clients by supporting a small number of standard protocols means that people who have a strong preference for one client over another are able to use that client. Typically, email clients also support multiple protocols, and it is not difficult to find a protocol in common between the client and the mail delivery server. Conversely, a site using a proprietary protocol locks the customers into a vendor's one or two clients that may not have the features they need or want.

More philosophy and anecdotes on this subject can be found in [Section 16.7](#).

Nonstandard Protocols Are Expensive

A small, successful Silicon Valley Internet startup company in the 1990s was bought by a large, well-established Washington company. The Internet startup used standard protocols for its email service. The clients were primarily Unix machines but also included a significant number of Macs and a handful of Windows PCs. Its email system worked well and was accessible to everyone in the company.

When the company was bought and integrated into the parent company, the employees of the startup had to switch to the parent company's email service, which was based on Microsoft's email product at the time and supported only proprietary standards. The company had to buy a PC running Windows for each of the Unix and Mac users—almost everyone—just so that they could communicate with the company. It was an outrageously expensive solution!

41.7 Automation

As with everything that SAs do, automation can simplify common tasks and ensure that they are performed reliably and accurately. Many areas of email administration should be automated as part of the process of building the service or should be incorporated into existing email services.

In particular, setting up an email account should be automated as part of the account creation process. The automation should include steps putting the person onto any relevant companywide and group-specific email lists. Similarly, removing the email account should be an automated part of account deletion.

Earlier, we discussed the need to occasionally move email accounts between

machines for the purpose of balancing load. Automating this process is important, as it is often a complicated process.

It is best not to provide automatic email forwarding for people who have left the company, because sensitive information may be inadvertently sent by someone who does not realize that person has left. Implementing a redirect message that automatically replies with the new email address and a message stating that the person has left the company is preferred.

Another useful form of email service automation is automating mailing list administration so that the lists can be created, deleted, and administered directly by the customers who need them rather than by the SAs. This provides a better, more responsive service for the list owners as well as relieves the SAs of a recurring duty.

A departing employee should be removed from all internal mailing lists. Periodic automatic checks of active email accounts against the personnel database should also be implemented, along with checks to ensure that email to local accounts is not forwarded outside the company and that disabled accounts are not on any internal mailing lists. A site can also automate the process of examining sensitive internal mailing lists for unauthorized members.

41.8 Monitoring

[Chapter 38](#), “[Service Monitoring](#),” introduced the notion that a service isn’t properly implemented until it is monitored. Email is no exception to that rule. A basic level of monitoring must be performed on an email service: Every machine that is part of the service should be monitored to detect that it is up, and that it is responding to requests on the relevant TCP ports (SMTP on port 25, POP3 on port 110, IMAP4 on port 143, and so on).

All email servers should also be monitored for disk space, disk activity, and CPU utilization. Monitoring disk space helps plan for increases in demand and alert SAs when disk space is nearing capacity. An email service makes huge demands on disk and CPU utilization. Monitoring those elements permits one to detect service-level variability caused by large influxes in email, email loops, and other problems. In those situations, having a baseline of what is normal—past history—helps to identify problems when they happen.

Email to the `postmaster` also must be monitored. The `postmaster` address at every site receives email messages showing mail delivery errors or bounces. By monitoring the email sent to this address, the person in charge of the email service will notice when failures occur and be able to address them.

The bounces sent to `postmaster` contain all the headers from the failed email and the error message indicating why it failed. This information is vital for debugging email problems and often is not supplied when end users report problems.

Finally, the logs on the email machines should be monitored to track the message-flow rate. This data can help you make capacity planning predictions, notice when mail delivery has stopped, and debug problems that can arise when the message-flow rate increases unexpectedly.

You should also consider implementing some more advanced end-to-end monitoring methods for the email system. Because email transmission and delivery comprise a complex series of events, SAs can easily overlook some small aspect of the system if they just perform monitoring on each component. End-to-end monitoring means building a test that emulates someone sending email to someone served by this email system. The test should emulate a customer or revenue-generating transaction as closely as possible to detect all possible problems, including those that are visible only from outside the site network. [Chapter 38](#) covers monitoring in detail. In particular, [Section 38.8](#) discusses end-to-end monitoring in general and includes an example that applies directly to email.

41.9 Redundancy

Because email service is central to the operation of all modern companies, sites should introduce redundancy into the email system as soon as it is realistically possible. [Chapter 17](#), “[Service Planning and Engineering](#),” and [Chapter 18](#), “[Service Resiliency and Performance Patterns](#),” describe some general ways of building reliability into a service. This chapter concentrates on email-specific issues.

When there is no redundant hardware for the email systems, there must be a recovery plan that can be implemented quickly in case of failure. It is easy to introduce redundancy for mail relay hosts and list-processing hosts using DNS Mail eXchanger (MX) records and multiple servers with the same configuration.

Redundancy of mail delivery hosts is different because entire hosts cannot easily be redundant for each other. Instead, you can make the server internally redundant with RAID and other techniques. You can replicate the host that accesses a shared message storage facility using network-attached storage or storage-area network technology. However, in that case, you must be extremely careful to ensure that proper locking and access control are performed.

The client access must be made redundant with transparent failover. To do so, you must understand how the client works. Clients typically cache the result of the initial DNS query that occurs when they try to contact the mail delivery server to download email; therefore, simple DNS tricks will not suffice. Redundancy for the client must happen at the IP level. Several technologies are available for permitting a host to take over answering requests directed to a specific IP address when the original machine at that address dies. Load balancers (layer 4) and the Virtual Router Redundancy Protocol (VRRP) are two mechanisms that are widely used to implement this approach.

Consider all components of the system and how they function when deciding on a redundancy strategy. As part of the decision-making process, consider how the various mechanisms work, how they will affect the rest of your environment, and how other machines will interact with them.

41.10 Scaling

All aspects of the mail system need to scale in advance of demand. Mail transport systems must be prepared to deal with higher volumes of traffic; mail access servers must deal with more simultaneous users; and list-processing systems must handle spikes in traffic and membership. All need to scale in advance of new technologies that significantly increase message sizes.

Mail transport systems need to scale to meet increased traffic levels. Three independent variables govern email traffic levels: the size of the messages, the number of messages per person, and the number of people using the email system. The more people using the email service, the more email messages it will need to handle. The number of messages per person typically increases gradually over time, with spikes around holidays.

The email service also needs to scale to cope with large bursts of traffic that might be triggered by promotional events or significant, unexpected problems. Sudden increases in traffic volume through mail transport systems are not unheard of. The mail transport system should be designed to deal with unexpectedly high traffic peaks. Mail transport systems should also be prepared to store the large quantities of email that might accumulate if there are problems passing the messages downstream.

Mail delivery systems need to scale predictably as the number of people served by the delivery system increases. The mail delivery server will have to scale with time even if there is no increase in the number of people that it serves, because of increases in the number and size of messages that people receive. If

customers store their email on the delivery server for a long time after they have read it, the delivery server will also have to scale to meet storage demand. Mail delivery systems should always have plenty of extra mail spool capacity; the usual rule of thumb for peak use is twice the size of regular use. Sudden, unexpected bursts of large messages can occur.

The Wrong Way to Scale

A university department had its own email delivery server. When it became overloaded, it sometimes refused POP3 client-connection requests. To fix the problem, the SA sent an email around the department, publicly berating customers who were using email clients that automatically checked for new email at a predefined interval, such as every 10 minutes. It was the default configuration for the email clients—not a deliberate, selfish choice as her message had implied. Several customers did take the time to figure out how to turn the feature off in their email clients, and the problem was resolved, at least temporarily. However, the problem kept recurring as new people arrived and the traffic volume increased. As we saw in [Chapter 30](#), “[Fixing Things Once](#),” it is better to fix things once than to partially fix them many times. It provides better service to the customers and ultimately less work for the SAs.

Instead of trying to embarrass and blame her customers, the SA should have been monitoring the mail server to figure out which lack of resources was causing the refused connections. She could then have fixed the problem at its source and scaled the server to deal with the increased demand. Alternatively, some mail server products optimize a client checking for new email by quickly replying that there are no new messages if the client has queried in the last 10 minutes. This requires fewer resources than checking whether new mail has arrived for that user.

As new technologies emerge, message sizes typically increase significantly. In the early days, email was simple text and was transmitted over slow modem connections. When people started sharing basic images and large programs, the larger files were turned into text and broken into small chunks that were sent in separate messages so that they could still be transmitted over modems. Early images were of low resolution and black and white. Higher-resolution monitors and color monitors resulted in significant increases in the size of images that were emailed. When email became common on PC and Mac systems, people started sharing documents and slide presentations, which were large files with

the potential to be huge. More formats and documents, including more higher-resolution images, continue to increase message sizes. The introduction of standard video and audio file formats and the subsequent sharing of those files also increased the volume of data sent in email.

Again, it is important to always have plenty of spare mail spool space. Watch for emerging technologies, and be prepared to scale rapidly when they emerge. Most mail systems let the SAs set message size limits, which can help with size problems, if handled well. However, if such constraints get in the way of people's work, people will find a way around the limitation, such as by splitting the email into several smaller chunks that will pass the size limit or by complaining to the CEO. It's better to use size limits only as a temporary stop-gap measure to deal with an overload problem while you find a permanent solution or possibly to set a really huge limit to prevent people from accidentally sending something enormous.

Organizations moving from POP3 to IMAP4 need to consider special scaling issues, since IMAP4 requires more resources on the server than POP3. POP3 is primarily used for moving email off a server and onto someone's laptop or desktop client. Disk space requirements are minimal because the server is used mostly as a holding area until the client next connects to the server. Once the transfer is complete, the server can delete its copy of the mailbox. The connections from the client are short-lived. Clients connect, download the mailbox, and disconnect. They do not return until it is time to check for new mail, which may be hours or days. IMAP4, however, maintains the mailbox and all folders on the server and simply provides access to it from the client. Therefore, disk space requirements are much greater, and servers experience long connection times. Having the email on the server has the benefit of permitting more centrally controlled and predictable backups. Because of the various network usage patterns, servers must be tuned for many simultaneous, long-lived network connections. This can require extra effort, as many operating systems are not configured, by default, to properly handle thousands of open TCP/IP connections.

List-processing systems have the same volume concerns as delivery and relay systems do, but also have to deal with an increased and more diverse list membership. When a list-processing machine has to deliver messages to a large number of other machines, a number of them will likely be unavailable for some reason. The list-processing system needs to deal gracefully with this situation without delaying the delivery of the message to other systems that are available. Some aspects of scaling a list-processing system are related to software

configuration ([Chalup, Hogan, Kulosa, McDonald & Stansell 1998](#)). Usage of disk space, network bandwidth, CPU, and memory should be monitored and scaled appropriately, too.

41.11 Security Issues

Mail relay hosts that communicate with places outside the company are traditionally targets for attackers, because such communication inherently requires exposure to the Internet or other extranets. These high-value targets for attacks have access to both the outside world and the internal corporate network, and often have special access to internal naming and authentication services. Mail delivery is a complex process; as a result, it is prone to bugs that, historically, have been leveraged as security holes. You should deal with security issues as part of the initial design—security is difficult to add later.

The mail system is also a conduit through which undesirable content, such as viruses, can get into the company. Several vendors offer products that scan email content for viruses and other undesirable or destructive content before it is accepted and delivered to the recipient. Consider whether your site should implement such content scanning and whether it conflicts with the site's privacy policy. If content scanning is implemented, make sure that the application understands the maximum number of possible data formats so that it can examine all the files in, for example, a zip archive attachment. Be aware that some things will slip through the net, however, and that similar scanning should also take place on your customers' desktop machines. When processing thousands or millions of email messages a day, such virus scanning can be a big bottleneck. If you use such systems, plan for high disk I/O rates.

Also consider how the email system fits into the security architecture. For example, if the site has a perimeter security model, which sort of protection does the firewall system provide for email? Does the system have a facility to prevent transmission of messages with suspicious headers that might be trying to exploit security holes? If not, where can that function best be implemented? Can the external mail relay systems be used by unauthorized persons to relay email not destined for the company? How can unauthorized mail relaying be prevented? How do customers access their email when they are on the road or at home? Many of the easiest and most common ways of providing people with access to their email when they are traveling involve transmitting passwords and potentially confidential email unencrypted across public, unsecured networks.

41.12 Encryption

Encryption can be done in many places in an email system. Server-to-server communication should be encrypted using the STARTTLS (RFC3207) protocol extension to SMTP.

Encrypting individual messages is a bit more difficult. There are many different competing standards and, for most software that implements them, the user interfaces tend to be horrible. Available commercial encryption packages are integrated with email clients to varying degrees of transparency. When looking at the various products, consider both the user-interface issues and the key-management issues. An encryption system needs a repository for everyone's encryption keys and a way for those keys to be rescinded if they are compromised. Also consider what to do in a disaster scenario in which something catastrophic happens to a key staff member and important encrypted emails are inaccessible. Some systems have key-recovery mechanisms, but those mechanisms should have adequate controls to ensure that the keys cannot be compromised.

Encryption systems and key management are complex topics and should be researched in detail before being implemented at a site. A well-implemented encryption service is an asset to any company. Be forewarned that there are many security vendors selling products that are more hype than anything else. Useful advice about detecting such products can be found in the article "Snake Oil Warning Signs: Encryption Software to Avoid" ([Curtin 1998](#)).

41.13 Email Retention Policy

Although backups are an essential part of good system administration practice, they also can be harmful to the company in an unexpected way. Many companies have a policy of not performing backups on email or of discarding those backups after a predefined short period of time.

The problem is that if the company becomes involved in a legal battle of any sort, such as defending its patent or intellectual property rights, all documents relating to the case are likely to be subpoenaed, which means searching through all backup tapes for relevant documents. Typically, formal documents relating to the topic will have been kept in well-known places. However, informal documents, such as emails, can be in any mailbox, and all email then has to be searched for potentially relevant documents. Whether on backup tapes or disk, emails that match the search criteria then have to be individually examined by someone who is able to determine whether the document is relevant and should be turned over to the court. This is a very expensive and time-consuming process that the legal department would rather avoid, regardless of whether it turns up

documents in the company's favor. The process of having to search through years' worth of old email on backup tapes is simply too expensive.

Medium-size and large companies usually have a document retention policy that specifies how long certain types of documents should be kept. This policy is in place to limit the amount of document storage space that is needed and to make it easier to find relevant documents. The document retention policy is typically extended to cover email at some point, often in relation to attaining SarbanesOxley Act conformance. At that point, the legal department will request that the SAs implement the policy. If email is distributed across many machines and in some nonstandard places, implementing the policy becomes difficult. In particular, if the email is stored on the laptops and desktops of people who use POP3 servers, the policy needs to cover the laptop and desktop backups and somehow separate the email backup from the system backup, so that the former can be discarded sooner. When designing your email service, it is best to bear in mind that you may be asked to implement this policy and to determine how you will do so.

41.14 Communication

An important part of building any service is communication, particularly telling people what the system's features are and how to use them. Another important component of communication about a service is the documentation of the system for SAs.

For the customers of an email service, it is important to ensure that everyone who uses the system understands the policies associated with the system—for example, regarding privacy, forwarding email off-site, email backups and schedule, any content filtering and its implications, and how the company defines “acceptable use” and applies that definition to email. Customers should be made aware of the risks associated with email, including forwarding material to inappropriate recipients, detecting scams, and recognizing that whenever someone emails you a program to run, it is most likely a virus. Customers should be also made aware of chain letters so that they recognize them and do not propagate them. Finally, customers should be made aware of useful features that may be available to them, such as encryption mechanisms and mailing list administration tools.

Sadly, handing a person a stack of policies is not very effective in making sure those policies are followed. It can be easier to brief new hires verbally or with a short slide presentation and make the complete policies available online. Some firms create a series of web forms, requiring customers to acknowledge and

agree to each policy.

Documenting the email system for other SAs is important. Failover and recovery procedures should be clear and well documented, as should the design of the system, including diagrams that show the flow of mail and which processing happens on which systems. SAs who do not work directly with the email system should be able to perform some preliminary debugging before reporting a problem, and that process should also be well documented. It is particularly important that SAs who may be in different time zones at remote sites understand the mail system architecture and be able to perform basic debugging tasks on their own.

41.15 High-Volume List Processing

Most sites have mailing lists, and many have mailing lists that serve paying customers. For example, a company might have a mailing list that announces product upgrades or new products to its customers. It might have a mailing list that keeps customers up-to-date on events happening at the company or one that announces serious bugs with its products. Other companies have mailing lists for people who are beta testing a product. A nonprofit organization may have one or more mailing lists to keep its members up-to-date on what is happening and which events are being organized. A university may have mailing lists for the students in each class. A service provider will almost certainly have mailing lists for its customers to let them know of outages that might affect them.

Mailing lists are handled centrally on mail servers rather than as an alias in one person's mail client. Everyone can send email to the same address to reach the same list of people. Mailing lists can be protected so that only a few authorized people can send to them, or messages have to be approved by an authorized person before they are sent, or only people who are members of the list can send messages to it. The lists can also be open so that anyone can send email to all the list recipients.

Most list management software gives the SAs the ability to delegate control of list membership, posting restrictions, and message approval to the person who manages the list from a business perspective. The list management software should also be able to permit customers to create and delete their own mailing lists without SA involvement. For example, if a company starts a beta-test program for a particular product, the manager in charge of the beta test should be able to set up the mailing list for the beta users, add and remove them from the list, control who sends messages to the list, and delete the list when the beta test is over.

When unsupervised creation of mailing lists is allowed, however, a policy and monitoring mechanism should be in place for work-related and non-work-related lists. The list policy should clearly indicate which types of lists are appropriate and who can join them. A carpool list for employees in a particular region? Very likely. A carpool list for employees going to baseball games? Less clear. What about a fan list for a particular team? What happens when someone leaves the company? Can that person still be on the baseball list with a new email address? And so on.

Relatively few companies will have high-volume, high-membership lists that require special scaling. Often, such sites are providing external services for others. The requirements for the average site's list-processing service can be met using basic, freely available software, but high-volume lists often exceed the capabilities of those systems. In those situations, investigate commercial software or services that are capable of handling those large volumes, or use open source software and hire a consultant who has designed such systems previously.

High-volume list services should also be built across several redundant systems to avoid cascading outages caused by the list server becoming swamped with work as soon as it recovers from an outage. On a high-volume list server, it is important to monitor the total length of elapsed time from when it starts sending a message to the first person on the list to when it finishes sending it to the last person on the list, excluding sites that it has problems contacting. The list members should not suffer from a large time lag. If people at the end of the list receive the message a day after people at the beginning of this list, it is difficult for them to participate in the conversation in a meaningful way, because they are so far behind the other people on the list.

Email list server technology changes dramatically over time as the demands placed on it increase. As with many areas of system administration, it is important to keep up with technology improvements in this area if you are running such a system. Most enterprises will outsource high-volume list processing to a company that specializes in this service, rather than building the infrastructure themselves.

41.16 Summary

Email is an important service to get right. People rely on it even more than they realize. In many ways, it is like a utility, such as power or water. Scaling the system to meet increasing demand, monitoring the service, and building redundancy should not be an afterthought. Security should be considered from

the outset, too, because email servers are a common target for attackers, and email is a common vector for viruses.

Before building an email system, you must establish several policies related to this system. Companies should carefully consider the email namespace policy, and make sure that the same namespace is used internally and externally. In addition, a privacy policy should be defined and communicated to the people who use the service.

Some companies may want to consider integrating encryption into the email service to provide extra protection for sensitive information. Larger companies may want to implement a policy for reducing how long backups of email are retained and to protect themselves from the expense of searching through old email to find messages of relevance to a court case. Sites where the email service is linked to the revenue stream should consider implementing end-to-end monitoring of the email service. Those sites that run high-volume list servers have special needs that must be addressed and should invest in a commercial package.

Exercises

1. What is the email privacy policy at your site? How many people know about it?
2. How many email namespaces are at your site? Who controls them?
3. Identify your mail relay machines. Do they have other functions?
4. Identify your mail delivery machines. Do they have other functions?
5. Where does list processing happen at your site?
6. When do you anticipate having to scale up your existing email system? Which aspect do you expect to scale, and why? How do you predict when it will run out of capacity?
7. How do you monitor your email system? Is there anything you would like to change or add?
8. How reliable (in percent terms) is email service at your site? How did you arrive at that figure?
9. If your site has multiple delivery machines, explain why each one exists. Could you reduce the number?
10. If your company has remote offices, can people in those offices access their email when the connection to the main site is down?
11. How is email security handled at your site?

- 12.** How does email flow from the Internet into your enterprise? How does it flow out to the Internet? Which security risks and exposures are involved, and how does your design mitigate them? Which improvements can be made?
- 13.** How would you implement a six-month retention schedule for email backups? Would you need to include desktops in this policy? How would you implement that component?

Chapter 42. Print Service

Printing is about getting a paper copy of the information you want, when you want it. Printing is a critical business function. Indeed, office workers tend to rank printing as one of the most important services.

Oddly, many SAs have disdain for printing or anyone who prints a lot. “What happened to the paperless office?” laments the anti-printer crowd. Many SAs pride themselves on how little they print, refusing to work with paper when an electronic version is available. Therefore, many SAs do not appreciate how important printing is to their customers and do not provide good printing support. The priorities of the SAs must align with those of their customers.

Printing is important to customers because, like it or not, business still runs on paper. Contracts need to be signed in ink. Diagrams need to be posted on walls. Paper can go places that computers can’t. People find different mistakes when proofreading on paper than on the screen, possibly as a result of the human ability to take the document to new environments. Even technocrats use paper: Although it may shock you, dear reader, every chapter of this book was proofread on white, bond, laser printer-compatible ... paper.

When a contract must absolutely, positively get there overnight, missing the overnight delivery truck because of a printer jam is an unacceptable excuse. Printing is a utility; it should always work. However, printing technology has radically changed many times during our careers. We therefore choose not to discuss the pros and cons of various print technologies—those choices may not even be available by the time the ink on these pages is dry. Instead, this chapter discusses the invariants of printing: getting the ink on the page, establishing printing policies, and designing print servers. Finally, we discuss ways to encourage environmentally friendly printer use. All these things are important, regardless of the evolution of the technology used to actually print anything.

A successful print system begins with an understanding of the requirements for a good policy, followed by solid design. Without proper policy, the design will be without direction. Without design, printing functions will be unstable and more difficult to use.

42.1 Level of Centralization

What would the perfect print environment look like? Some people want their own printers attached to their own computers. In their perfect world, every

machine would have its own high-speed, highquality printer. This setup is extremely expensive but very convenient for the customers. For others, the key issue is that no matter how many printers exist, they should be able to print from any host to any printer. This design has the benefit that customers are able to “borrow” someone’s highquality, color printer as needed and is certainly a flexible configuration. Finance people look at the high cost of printers and printer maintenance and would prefer to centralize printing, possibly recommending that each building have one high-speed printer and one highquality printer.

The primary requirement of a print system is that people can print to any printer they have permission to use, which might encompass all printers or only a well-defined set of printers. Costs are recouped either in per-page increments, as part of a “tax,” or by having each group—center, department, division—fund its own print usage. If your life is not complicated enough, try using completely different cost-recovery methods for the hardware, supplies, and maintenance.

A typical office arrangement is to have one printer per section of a building, be it a hallway, wing, or floor. These printers can be accessed either by anyone in the company or by anyone in the same cost center. Some individuals may have private printers because of confidentiality issues (they need to print documents containing confidential information), ego (they’re important and demonstrate this by having their own printers), or some other business reason. In addition, high-speed printers, photographic-quality printers, wide-format plotters, and special printers may have special-access control because of their additional cost or operating requirements.

There is a tradeoff between cost and convenience. More printers usually means more convenience: a shorter walk to the printer or increased variety of types of printers available. However, having more printers costs more in a variety of ways.

By comparison, a shared, centralized printer can be so much less expensive that you can use some of the savings to purchase a significantly higher-quality printer. Suppose that inexpensive desktop printers cost \$100 each. If 10 people share a network printer, the break-even point is \$1,000. At the time of this writing, \$1,000 can purchase a nice network printer and still leave room for a few years of maintenance. If 20 people share that printer, they now have a better printer at half the price of individual desktop printers; in addition, they get a little exercise walking to get their printouts.

No Standards for Printers

When there is little or no control over spending, costs will get out of hand. One company had few restrictions on spending for items less than \$600. When the cost of desktop laser printers dropped below that limit, droves of employees started purchasing them, introducing dozens of new models to the support load of the SA team, which often had to install the printers, maintain printer drivers, and debug problems. Without maintenance contracts, people were throwing away printers rather than repairing them.

Management began to crack down on this problem. In an informal audit, the company found that many of the printers that had been purchased were now attached to employees' home computers. Cost controls were put into place, and soon the problem was halted. Most of the printers purchased were slow and had low-quality output compared with the centrally purchased printers located in each hallway. However, employees hated the policy. They were more concerned that a printer be within inches of their desk than be of higher quality or more cost effective.

42.2 Print Architecture Policy

Every site should have certain written architecture policies related to printing. The first policy is a **general printer architecture policy** about how centralized printing will be. The goal of this document is to indicate how many people will share a printer for general printing (that is, one printer per desktop, one per hallway, one per building, one per floor), who qualifies for personal printers, and how the printers will be networked.

The issue of who orders supplies and who resupplies the printers should also be part of the written policy. Some offices simply have the department administrator order the supplies, but sometimes the ordering is centralized. At some sites, the users of a particular printer are expected to refill the paper and change toner; at others, operators monitor printers and perform these tasks. It's best not to permit customers to handle toner cartridges themselves. Sometimes, this procedure is complicated and error prone, not something that should be left to the typical customer. Moreover, you will find that customers will change toner at the slightest sign of quality problems, whereas a trained operator can take other steps, such as shaking the toner cartridge to get another couple hundred pages out of it. Changing toner cartridges needlessly is a waste of money and has environmental implications.

An organization should also have a documented *printer equipment standard*.

This policy should be divided into two parts. The first part should change rarely and should specify long-term standards, such as whether duplexing units (when available) should be purchased, which protocol printers must speak, how they are connected to the network, and so on.

The second part should be a list of currently recommended printers and configurations. For example, you might list an approved configuration or two for purchasing a color printer, a color printer for transparencies, and a couple of black-and-white printers of two or three price ranges. This part of the policy should be updated on a regular basis. Such standards can often save money because they may qualify your company for volume discounts. Problems can be avoided by having all equipment orders filtered through the SA team or other knowledgeable people, to verify the completeness of orders and schedule installations so SAs are not caught unprepared when the devices arrive. Limiting the number of models in use also reduces the number and types of supplies you have to inventory, which saves money and confusion.

Case Study: Recommended Configurations Save SA Time

Recommended configurations should include all the items that customers might forget but that SAs will need to complete the installation. One site had a centralized printing system and provided a printer in each part of the building. These printers were always the same model. A group wanting a printer for its own, dedicated use often made the valiant effort of purchasing the same model as the others around the building.

Unfortunately, such groups typically didn't know to purchase the optional duplexing unit, cable, and network connectivity package. When the SAs were asked to install such printers, they had to give the customer the sad news that there would be a delay while the proper add-ons were purchased so that the device would be usable. This problem continued until a printer standards document was written.

A **printer access policy** should determine who can access which printers and how that permission will be enforced. For example, people may be permitted to print to all printers, only to printers that their department paid for, or somewhere in between. This policy should also specify who can cancel print jobs on printers. For example, people other than administrators should be able to cancel only their own print jobs, whereas administrators should be able to cancel any print job on the spoolers they control. Universities may have to meticulously control this access, because students may be prone to pranks involving canceling

one another's print jobs.

In business, inside a firewall, it is reasonable to permit everyone within the company to print to every printer and to cancel any job on any printer. Being able to print to any printer in the company makes it possible for people to replace faxing with printing. A designer in San Jose can print to the printer in Japan over the corporate WAN, saving the cost of the international phone call. Can this capability be abused? Absolutely. However, misuse is extremely rare and will be even rarer if the cover page names the perpetrator. If employees have a private printer that they don't want others to print to, they can lock their doors. If someone has to ask for a printout, this can be an opportunity to tell the person not to print to the private printer. If people can print to any printer, it needs to be clear to them where the printers are located. You don't want people to print to the wrong country by mistake.

A **printer naming policy** should be established. It is useful for printers to be named geographically; that is, the name indicates where the printer is located. Nothing is more frustrating than having printers named with a theme that does not help locate the printer. Although it may be creative and cool to have printers named `decaf`, `latte`, `mocha`, and `froth`, such a naming scheme makes it difficult for people to find their printouts. Naming a printer after the room number it is in or near makes it easier for customers to find the printer.

One major pitfall to naming a printer after its location is that if the location changes, everyone must update their printer configuration. Otherwise, people will be frustrated to learn that they couldn't find printer `f12rm203` because it is now across the hall in room 206. This is less likely to happen if floorplans have designated printer areas that are unlikely to move.

If the printer is likely to move, or is located in a small office, a more general name might be more appropriate. Perhaps the name might indicate which floor it is on, possibly concatenated with a word such as `north`, `east`, `south`, or `west`.

Printers owned and managed centrally should not be named after the group that uses the printer. If that group moves to another part of the building, the group would want to take that printer. People left behind will lose their printer or have to change their printer configuration, which is like being punished for not doing anything. Printers owned by particular groups should also not be named after the group because when you decide to centralize the service, you want employees to have an easier time letting go of the old ways.

Another option is to have generic printer names, and to keep the location in

the CMDB. The CMDB can then be queried by a web application that locates the closest appropriate printer and shows it on a building plan. Often, a person will locate the nearest suitable printer, find its name (through a label on the printer), go back to his or her computer, set up that printer, and then print the document. The printer label can also indicate its capabilities.

Unix printers often have two-part names. The first part indicates the printer's common name, and the second part may be a code for which kind of printout will be produced. For example, printer `2t408-d` may send the job to the printer in room 2t408 in duplex (double-sided) mode, and `2t408-s` may be an alias that prints to the same printer but in single-sided mode.

Outsourced Printing

If you need a fancy print feature only occasionally, such as binding, three-hole punch, high-volume printing, or extra-wide formats, many print shops accept PDF files. Sending a PDF file to a store once in a while is much more cost-effective than buying a fancy printer that is rarely used.

Nearly all print shops will accept PDFs via email or on their web site. Print shop chains have created printer drivers that “print to the web.” These drivers submit the printout to their printing facility directly from applications. You can then log in to the company's web site to preview the output, change options, and pay. Most will ship the resulting output to you, which means that you never have to leave your office to retrieve the printout.

42.3 Documentation

It goes without saying that the architecture, operations procedures, and software used in your print system should be documented. Documentation is a critical part of any well-run system. SAs must provide three kinds of printing documentation to their customers:

- **Print HOWTO:** This document should include which menus and buttons to click to connect to the printer of the customer's choosing and explain which commands must be executed. For example, a Unix environment might explain the environment variables that must be set, whether `lpr` or `lp` is used, and which options to use to get simplex, duplex, letterhead, or other specialty output. This document shouldn't need to change very often. It should be made available as part of the getting-started manual or web site that new customers are given.

- **List of printers:** This is a catalog of all the printers available, where they are located, and which special features they have, such as color, quality, and so on. This document should tell customers where they can find the print HOWTO document. The list of printers needs to be updated every time a new printer is added or removed from the system. It should be posted on the wall near every printer it lists, in addition to being part of the getting-started manual or web site that new customers are given.
- **Label:** Each printer should have a label that indicates the printer's name. Trays should be labeled if they are intended for transparencies, letterhead, and so on. We recommend a consistent tray scheme. For example, letterhead might always be in tray 2. Labeling printers is such a simple thing, yet many SAs forget to do it. The printer users typically find that this is the most important documentation you can provide. They can usually figure out the rest!

Case Study: Printer Maps

At Google, one can go to <http://print> to view a web site that lists all the printers by location and floorplan maps to help customers find them. The web site uses your IP address to determine your location and defaults to the location and floor you are on. Clicking on the name of a printer brings up a menu of choices, including a link that will automatically add this printer to your computer (doing the right thing no matter which OS you use), display a description of the printer's capabilities, and provide a link that opens a ticket at the helpdesk with the name of the printer already filled in the ticket.

42.4 Monitoring

As you learned in [Chapter 38](#), "[Service Monitoring](#)," nothing deserves to be called a *service* until it is monitored. Two aspects of the print system need this attention. The first is the spooling and printing service itself. The SAs need to monitor each spooler to make sure that the queue hasn't stalled, the spool disk hasn't filled, logs are being recycled, the CPU is not overloaded, the spool disk hasn't died, and so on. This should be a part of the normal monitoring system.

The second aspect that needs to be monitored is the status of each printer. Toner and paper trays need to be refilled. Shelves of paper need to be restocked. Most network printers speak SNMP and can alert your monitoring system that they are jammed, low on toner, or out of paper. Printers can often monitor

exactly how much paper they have remaining, so that SAs can be alerted when they are nearly out of paper, though it's more efficient to let customers load the paper themselves.

Although there are automated means to tell whether a printer is out of paper, there is no automated way to tell whether a good supply of paper is sitting near the printer. SAs can either visit the printers on a regular basis or deputize customers to alert them when paper supplies are running low.

Case Study: Print Tests

Newer printers indicate when they are low on toner, but older printers require a test print. One site used a program to generate a test print on every printer each Monday morning. The page included a time, date, and warning to customers not to throw it out. The bottom half of the page was a test pattern. An operator was assigned to retrieve these printouts and add toner if the test pattern was unreadable. The operator pulled a cart of paper along and restocked the supply cabinets under each printer. This very simple system maintained a high level of service.

42.5 Environmental Issues

As SAs, we must take responsibility for the environmental aspects of printing. Toner is toxic. Printing kills trees, and discarded printouts fill up landfills. Even recycling has both costs and environmental impacts; a page that is not printed has a lower impact than a page that is printed and then recycled. Print systems must be designed to minimize waste. Toner cartridges should be recycled. SAs should be sensitive to the environmental impact of the chemicals used in the printing process. Wasteful printing should be discouraged and paperless solutions encouraged, when possible. Considerable cost savings are also gained by the reduction in paper and toner to be purchased. Waste-disposal charges usually are lowered if recyclable paper is separated.

Customers will recycle when it is easy, and they won't when it is difficult. If there are no recycling bins, people won't recycle. Putting a recycling bin next to each trashcan and near every printer makes it easy for customers to recycle. As SAs, it is our responsibility to institute such a program if one doesn't exist, but creation of such a system may be the direct responsibility of others. We should all take responsibility for coordinating with the facilities department to create such a system or, if one is in place, to make sure that proper paper-only wastebaskets are located near each printer, instructions are posted, and so on.

The same can be said for recycling used toner cartridges. This is an opportunity to collaborate with the people in charge of the photocopying equipment in your organization because many of the issues overlap.

Some issues are the responsibility of the customers, but SAs can help them do the right thing by giving customers the right tools. SAs must also avoid becoming roadblocks that encourage their customers to adopt bad habits. For example, making sure that preview tools for common formats, such as PostScript, are available to all customers helps them print less. Printers and printing utilities should default to duplex (double-sided) printing. Don't print a burst page before each printout. Replacing paper forms with web-based, paperless processes also saves paper, though making them easy to use is a challenge.

Creating a Recycling Program

A large company didn't recycle toner cartridges, even though new ones came with shipping tags and instructions on how to return old ones. The vendor even included a financial incentive for returning old cartridges. The company's excuse for not taking advantage of this system was simply that no one had created a procedure within the company to do it! Years of missed cost savings went by until someone finally decided to take the initiative to create a process. Once the new system was in place, the company saved thousands of dollars per year.

Lesson learned: These things don't create themselves. Take the initiative. Everyone will use the process if someone else creates it.

42.6 Shredding

People print the darnedest things: private email, confidential corporate information, credit card numbers, and so on. We've even met someone who tested printers by printing Unix's `etcpasswd` file, not knowing that the encrypted second field could be cracked. Some sites shred very little, some have specific shredding policies, and others shred just about everything.

Shred anything that you wouldn't want to appear on the front page of the *New York Times*. Err on the side of caution about what you wouldn't want to see there. An option that is more secure is to not print sensitive documents at all.

On-site shredding services bring a huge shredder on a truck to your site to perform the shredding services, whereas off-site shredding services take your

papers back to their shredding station for treatment. Now and then, stories surface that an off-site shredding service was found to not actually shred the paper as it had promised. These may be urban legends, but we highly recommend performing regular spot-checks of your off-site shredding service if you choose to use one. Shredding services are typically quite expensive, so you should make sure that you are getting what you pay for. On-site shredding is more secure. Designate a person to watch the process to make sure that it is done properly.

42.7 Summary

Printing is a utility. Customers expect it to always work. The basis of a solid print system is well-defined policies on where printers will be deployed (desktop, centralized, or both), which kinds of printers will be used, how they will be named, and which protocols and standards will be used to communicate with them. Print system architectures can run from very decentralized (peer-to-peer) to very centralized. It is important to include redundancy and failover provisions in the architecture. The system must be monitored to ensure quality of service.

Users of the print system require a certain amount of documentation. How to print and where the printers they have access to are located should be documented. The printers themselves must be labeled.

Printing has an environmental impact; therefore, SAs have a responsibility to not only work with other departments to create and sustain a recycling program, but also provide the right tools so that customers can avoid printing whenever possible. The best print systems provide shredding services for sensitive documents.

Exercises

1. Describe the nontechnical print policies in your environment.
2. Describe the print architecture in your environment. Is it centralized, decentralized, or a mixture of the two?
3. How reliable is your print system? How do you quantify that?
4. When there is an outage in your print system, what happens, and who is notified?
5. When new users arrive, how do they learn how to use the printers? How do they learn your policies about acceptable use?

6. How do you deal with the environmental issues associated with printing at your location? List policies and processes you have, in addition to the social controls and incentives.
7. Which methods to help limit printing do you provide to your customers?

Chapter 43. Data Storage

The systems we manage store information. The capacity for computers to store information has doubled every year or two. The first home computers could store 160 kilobytes on a floppy disk. Now petabytes—millions of millions of kilobytes—are commonly bandied about. Every evolutionary jump in capacity has required a radical shift in techniques to manage the data.

You need to know two things about storage. The first is that it keeps getting cheaper—unbelievably so. The second is that it keeps getting more expensive—unbelievably so.

This paradox will become very clear to you after you have been involved in data storage for even a short time. The price of an individual disk keeps getting lower. The *price per gigabyte* has become so low that people now talk about *price per terabyte*. When systems are low on disk space, customers complain that they can go to the local computer store and buy a disk for next to nothing. Why would anyone ever be short on space?

Unfortunately, the cost of connecting and managing all these disks seems to grow without bound. Previously, disks were connected with a ribbon cable or two, which cost a dollar each. Now fiber-optic cables connected to massive storage array controllers cost thousands of dollars. Data is replicated multiple times, and complicated protocols are used to access the data from multiple simultaneous hosts. Massive growth requires radical shifts in disaster-recovery systems, or *backups*. Compared to what it takes to manage data, the disks themselves are essentially free.

The industry has shifted in emphasis from having storage to managing the data through its life cycle. This is a big change. Now the discussion is no longer about price per gigabyte but *price per gigabyte-month*. A study published in early 2006 by a major IT research firm illustrated the variability of storage costs. For array-based simple mirrored storage, the report found a two orders of magnitude difference between low-end and high-end offerings.

Storage is a huge topic, with many fine books written on it. This chapter focuses on basic terminology, some storage-management philosophy, and key techniques. Each of these is a tool in your toolbox, ready to be pulled out as needed.

Rather than storage being seen as a war between consumers and providers, we promote the radical notion that storage should be managed as a community

resource. This reframes storage management in a way that lets everyone work toward common goals for space, uptime, performance, and cost. Storage should be managed like any other service, and this chapter provides advice in this area. Performance, troubleshooting, and evaluating new technologies are all responsibilities of a storage service team.

But first, we begin with a whirlwind tour of common storage terminology and technology.

43.1 Terminology

As a system administrator, you may already be familiar with a lot of storage terminology. Therefore, this section briefly highlights the terminology and key concepts used later in the chapter.

43.1.1 Key Individual Disk Components

To understand the performance of storage systems, one must understand the fundamental components that make up such systems. Many storage performance issues are directly related to the physical nature of storage. Here is an explanation of the parts that make up storage systems and how they work.

Spindle, Platters, and Heads

A disk is made up of several platters on which data is stored. The platters are all mounted on a single spindle and rotate as a unit. Data is stored on the platters in tracks. Each track is a circle with the spindle as its center, and each track is at a different radius from the center. A cylinder is all the tracks at a given radius on all the platters. Data is stored in sectors, or blocks, within the track, and tracks have different numbers of blocks based on how far from the center they are. Tracks farther from the center are longer and therefore have more blocks. The heads read and write the data on the disk by hovering over the appropriate track. There is one head per platter, but they are all mounted on the same robotic arm and move as a unit. Generally, an entire track or an entire cylinder will be read at once, and the data cached, as the time it takes to move the heads to the right place (seek time) is longer than it takes for the disk to rotate 360 degrees. Because the platters are rotating, we call these **spinning disks** to differentiate them from solid-state disks.

Solid-State Disks

Solid-state disks (SSD) have no mechanical or moving parts. Instead, information is stored in chips using NAND or other technology, which are like

RAM chips that retain their information when powered off. Unlike with spinny disks, there is no seek time, so SSDs are faster than spinny disks for reading. Writes are slower, but buffering usually hides this delay. Pauses between writes allow the system time to catch up and complete the queue of buffered writes. This works well because most systems tend to do more reads than writes. However, if there is a continuous stream of writes, the buffer fills and writes slow to the speed that they can be completed.

Originally SSDs were packaged in hardware that emulated traditional hard drives. As explained in [Section 5.6](#), these interfaces are being replaced by native connectivity, which improves performance, eliminates hardware components, and reduces cost. While SSDs retain information between reboots and are very reliable, they are not perfect. Backups are still required; RAID should still be used when appropriate.

Drive Controller

The drive controller is the embedded electronics on the drive itself. It implements a data access protocol such as SCSI or SATA. The drive controller communicates with the host to which the disk is attached. Drive controllers are important for their level of standards compliance and for any performance enhancements that they implement, such as buffering and caching.

Host Bus Adapter

The host bus adapter (HBA) is in the host and manages communication between the disk drive(s) and the server. The HBA uses the data access protocol to communicate with the drive controller. A smart HBA, which caches and optimizes operations, can also be a source of performance enhancements. An HBA either is built into the motherboard of the computer or is an add-on card.

43.1.2 RAID

RAID is an umbrella category encompassing techniques that use multiple independent hard drives to provide storage that is larger, more reliable, or faster than a single drive can provide. Each RAID technique is called a **level**. The most important RAID characteristics that every system administrator should know are listed in [Table 43.1](#).

Level	Methods	Characteristics
0	Stripes	Fast reads and writes; poorer reliability
1	Mirrors	Improved reads; good reliability; more expensive
5	Distributed parity	Faster reads; slower writes; more economical
10	Mirrored stripes	Fastest reads; best reliability; most expensive

Table 43.1: Important RAID Properties

Here is a more detailed description of each RAID level:

- **RAID 0:** Also known as striping, RAID 0 spreads data across multiple disks in such a way that they can still act as one large disk. A RAID 0 virtual disk is faster than a single disk; multiple read and write operations can be executed in parallel on different disks in the RAID set. Each spindle works independently. Thus, with four spindles in action, four disk operations can happen at the same time. RAID 0 is less reliable than a single disk, because the whole set is useless if a single disk fails. With more disks, failures are statistically more likely to happen.
- **RAID 1:** Also known as mirroring, RAID 1 uses two or more disks to store the same data. The disks should be chosen with identical specifications. Each write operation is done to both (or all) disks, and the data is stored identically on both (all). Read operations can be shared between the disks, speeding up read access, similar to RAID 0. Writes, however, are as slow as the slowest disk. The write is not complete until all devices have confirmed the write is complete. RAID 1 increases reliability. If one disk fails, the system keeps working as if it was a single disk.
- **RAID 2 and 3:** These levels are rarely used, but are similar to RAID 5. It should be noted that RAID 3 gives particularly good performance for sequential reads. Therefore, large graphics files, streaming media, and video applications often use RAID 3.
- **RAID 4:** This level is also similar to RAID 5, but is rarely used because it is usually slower. In fact, it is faster than RAID 5 only when a file system is specifically designed for it. One example of RAID 4 is the WAFL file system used on the NetApp file server product.
- **RAID 5:** Also known as distributed parity, this level improves reliability at a lower cost than mirroring. RAID 5 is like RAID 0 with one extra disk. Like RAID 0, it uses striping to make larger volumes. However, the additional disk is used to store recovery information. If a single disk fails, the RAID 5 set continues to work because the information on the

remaining disks can be used to determine what was on the failed disk. When the failed disk is replaced, the data on that disk is rebuilt, using the recovery disk. Performance is reduced during the rebuild. RAID 5 gives increased read speed, much like RAID 0. However, writes can take longer, as creating and writing the recovery information require reading information on all the other disks.

- **RAID 6:** Also known as double parity, this level is defined as any form of RAID that can overcome any two concurrent disk failures to continue to execute read and write requests to all of a RAID array's virtual disks.
- **RAID 10:** Originally called **RAID 1 + 0**, this level uses striping for increased size and speed, and mirroring for reliability. RAID 10 is a RAID 0 group that has been mirrored onto another group. Each individual disk in the RAID 0 group is mirrored. Since mirroring is RAID 1, the joke is that this is 1 + 0, or 10. Rebuilds on a RAID 10 system are not as disruptive to performance as rebuilds on a RAID 5 system. Some implementations permit more than one mirror for additional speed or to improve recovery time.

RAID levels other than 0 allow for a **hot spare**, an extra unused disk in the chassis. When a disk fails, the system automatically rebuilds the data onto a hot spare. Some RAID products permit multiple RAID sets to share a hot spare. The first RAID set to require a new disk takes the spare, which saves the cost of multiple spares.

A Mnemonic for RAID 0 and RAID 1

RAID 0 and RAID 1 are two of the most commonly used RAID strategies. People often find it difficult to remember which is which. Here's our mnemonic: "RAID 0 provides zero help when a disk dies."

43.1.3 Volumes and File Systems

A **volume** is a chunk of storage as seen by the server. Originally, a volume was a disk, and every disk was one volume. However, with partitioning, RAID systems, and other techniques, a volume can be any kind of storage provided to the server as a whole. The server sees a volume as one logical disk, even though behind the scenes it is made up of more complicated parts.

Each volume is formatted with a **file system**. Each of the many file system types was invented for a different purpose or to solve a different performance problem. Some common Windows file systems are FAT, DOS FAT32, and

NTFS. Unix/Linux systems have UFS, UFS2, EXT2/EXT3/EXT4, ReiserFS, ZFS, and Btrfs. Some file systems do **journaling**; that is, they keep a list of changes requested to the file system and apply them in batches. This improves write speed and makes recovery faster after a system crash.

43.1.4 Directly Attached Storage

Directly attached storage (DAS) is simply the conventional hard disk connected to the server. DAS describes any storage solution whereby the storage is connected to the server with cabling rather than with a network. This includes a RAID array that is directly attached to a server.

43.1.5 Network-Attached Storage

Network-attached storage (NAS) is a new term for something that's been around for quite a while. It refers to clients accessing the storage attached to a server—for example, Unix clients that use NFS to access files on a server, or Microsoft Windows systems that use CIFS to access files on a Windows server. Many vendors package turnkey network file servers that work out of the box with several file-sharing protocols. Network Appliance and EMC make such systems for large storage needs; Linksys and other companies make smaller systems for consumers and small businesses.

43.1.6 Storage-Area Networks

A storage-area network (SAN) is a system in which disk subsystems and servers both plug into a dedicated network—specifically, a special low-latency, high-speed network optimized for storage protocols. Any server can attach to any storage system—at least as access controls permit. What servers can attach to is a storage volume that has been defined and is referred to by its logical unit number (LUN).

A LUN might be a disk, a slice of a RAID 5 group, an entire storage chassis, or anything the storage systems make available. Servers access LUNs on the block level, not the file system level. Normally, only one server can attach to a particular LUN at a time; otherwise, servers would get confused as one system updates blocks without the others realizing it. Some SAN systems provide **cluster file systems**, which elect one server to arbitrate access so that multiple servers can access the same volume simultaneously. Tape backup units can also be attached to the network and shared, with the benefit that many servers share a single expensive tape drive. Another benefit of SANs is that they reduce isolated storage. With DAS, some servers may be lacking free disk space while others

have plenty. The free space is not available to servers that need it. This situation is called **stranded storage**. SAN technology eliminates this problem because each server can be allocated volumes as big as they need, and no disk space is wasted.

43.2 Managing Storage

Management techniques for storage rely on a combination of process and technology. The most successful solutions enlist the customer as an ally, instead of making SAs the “storage police.”

It is not uncommon for customers to come to an SA with an urgent request for more storage for a particular application. There is no magic solution, but applying the principles outlined in this section can greatly reduce the number of so-called emergency storage requests that you receive. It is always best to be proactive rather than reactive, and that is certainly the case for storage.

43.2.1 Reframing Storage as a Community Resource

Storage allocation becomes less political and customers become more self-managing when storage servers are allocated on a group-by-group basis. It is particularly effective if the cost of the storage service comes from a given group’s own budget. That way, the customers and their management chain feel that they have more control and responsibility.

Studies show that roughly 80 percent of the cost of storage is spent on overhead—primarily support and backups—rather than going toward the purchase of the hard drives. It should be possible to work with management to also pass on at least some of the overhead costs to each group. This approach is best for the company because the managers whose budgets are affected by increasing storage needs are also the ones who can ask their groups to clean up obsolete work and save space.

Sometimes it is not possible to dedicate a storage server to a single group. When a storage server must serve many groups, it is always best to start with a storage-needs assessment for your customer base; when the assessment is complete, you will know how much storage a group needs currently, whether the existing storage is meeting that need, and how much future capacity that group anticipates.

By combining data from various groups’ needs assessments, you will be able to build an overall picture of the storage needs of the organization. In many cases, reshuffling some existing storage may suffice to meet the needs identified.

In other cases, an acquisition plan must be created to bring in additional storage.

Doing a storage assessment of departments and groups begins the process of creating a storage community. As part of the assessment, groups will be considering their storage needs from a business-and work-oriented perspective rather than simply saying, “The more, the better!”

One great benefit of bringing this change in attitude to your customers is that the SAs are no longer “the bad guys,” but instead become people who are helping the customers implement their desired goals. Customers feel empowered to pursue their storage agendas within their groups, and a whole set of common customer complaints disappears from the support staff’s radar.

Raw Versus Usable Disk Capacity

When purchasing storage, it is important to remember that raw storage is significantly different from usable storage.

A site needed a networked storage array to hold 4 terabytes of existing data, projected to grow to 8 terabytes within 2 years. The customer told the vendor, which cheerfully sent an 8 terabyte array system. The customer began configuring the array, and a significant amount of space was consumed for file system overhead plus disks used for RAID redundancy, snapshots, and hot spares. Soon, the customer discovered that current usage amounted to 100 percent of what was remaining. The system would not support any growth.

Luckily, the customer was able to work with the vendor to replace the disks with ones twice as large. Because this was done before the customer took official delivery, the disks were not considered “used.”

Although the site’s future capacity problem was now solved, the problems continued. More disks required more CPU overhead. The application was extremely sluggish for almost 2 weeks while an emergency upgrade to the array processor controller was arranged. The finance people were not very happy about this, having approved the original system, then having to pay for the drive upgrades, and finally needing to write a big check to upgrade the processor.

43.2.2 Conducting a Storage-Needs Assessment

You might think that the first step in storage-needs assessment is to find out who is using which storage systems. That is actually the *second* step. The first step is

to talk to the departments and groups you support to find out their needs. Starting with a discovery process based on current usage will often make people worry that you plan to take resources away from them or redistribute resources without their input.

By going directly to your customers and asking what does and doesn't work about their current storage environment, you will be building a bridge and establishing trust. If you can show graphs of their individual storage growth over the last year and use this information to educate them rather than scold them, it can help everyone understand their real needs.

You may be surprised at what you discover, both positive and negative. Some groups may be misrepresenting their needs, for fear of being affected by scarcity. Other groups may be struggling with too few resources but not complaining because they assume that everyone is in the same situation.

Which kinds of questions should you ask in the storage assessment? Ask about total disk usage, both current and projected, for the next 6 to 18 months. It's a good idea to use familiar units, if applicable, rather than simply "months." It may be easier for customers to specify growth in percentages rather than gigabytes. Ask about the next 2 to 6 quarters, for instance, in a company environment, and the upcoming terms in an academic environment. You should also inquire about the kinds of applications being run and any problems that are being encountered in day-to-day usage. You may feel that you are adequately monitoring a group's usage and seeing a pattern emerge, such that you are comfortable predicting the group's needs, but some aspects of your storage infrastructure may already be stressed in ways that are not readily visible from simple metrics. These pressure points can become evident when customers provide clues.

Can't Get There from Here

A midsize chip development firm working closely with a new partner ordered high-end vendor equipment for a new cluster that would be compatible with the partner's requirements. Performance, affordability, and reliability were analyzed and hotly debated. After the new hardware showed up on the dock, it was discovered that one small detail had been overlooked: The site ordering the new hardware was working with different data sets than the partner site, and the storage solution ordered was not scalable using the same hardware. More than half of the more expensive components (chassis, controller) would have to be replaced to make a larger cluster. Instead of serving the company's storage needs for

all its engineering group for a year, the solution would work for one department for about six months.

It is also possible that upcoming events outside the current norm may affect storage needs. For example, the department you support may be planning to host a visiting scholar next term, someone who might be bringing a large quantity of research data. Or the engineering group could be working on adding another product to its release schedule or additional use cases to its automated testing. Each of these changes, of course, requires a significant increase in storage allocation. Often, the systems staff are the last to know about these things, as your customers may not be thinking of their plans in terms of the IS requirements needed to implement them. Thus, it is very useful to maintain good communication and explicitly ask about customers' plans.

Work Together to Balance System Stress

At one of Strata's sites, the build engineers were becoming frustrated trying to track down issues in automated late-night builds. Some builds would fail and report mysteriously missing files, but the problem was not reproducible by hand. When the engineers brought their problem to the systems staff, the SAs checked the server logs and load graphs for the related systems.

It turned out that a change in the build schedule, combined with new tests implemented in the build, had caused the build and the backups to overlap in time. Even though they were running on different servers, this simultaneous load from the build and the nightly backups was causing the load on one file server to skyrocket to several times the usual level, resulting in some remote file requests timing out. The missing files would cause sections of the build to fail, thus affecting the entire build at the end of its run when it tried to merge everything.

Since this build generally required 12 to 18 hours to run, the failures were seriously affecting the engineering group's schedule. Since backups are also critical, they couldn't be shut off during engineering's crunch time. A compromise was negotiated, involving changing the times at which both the builds and the backups were done, to minimize the chances of overlap. This solved the immediate problem. A storage reorganization, designed to solve the underlying problem, was begun so that the next production builds would not encounter similar problems.

43.2.3 Mapping Groups onto Storage Infrastructure

Having gleaned the necessary information about your customers' current and projected storage needs, the next step is to map groups and subgroups onto the storage infrastructure. At this point, you may have to decide whether to group customers with similar needs by their application usage or by their reporting structure and workgroup.

If at all possible, arrange customers by department or group rather than by usage. Most storage-resource difficulties are political and/or financial. Restricting customers of a particular server or storage volume to one workgroup provides a natural path of escalation entirely within that workgroup for any disagreements about resource usage. Use group-write permissions to enforce the prohibition against nongroup members using that storage.

Some customers scattered across multiple departments or workgroups may have similar but unusual requirements. In that case, a shared storage solution matching those requirements may be necessary. That storage server should be partitioned to isolate each workgroup on its own volume. This removes at least one element of possible resource contention. The need for the systems staff to become involved in mediating storage-space contention is also removed, as each group can self-manage its allocated volume.

If your environment supports quotas and your customers are not resistant to using them, individual quotas within a group can be set up on that group's storage areas. When trying to retrofit this type of storage arrangement on an existing set of storage systems, it may be helpful to temporarily impose group quotas while rearranging storage allocations.

Many people will resist the use of quotas, and with good reason. Quotas can hamper productivity at critical times. An engineer who is trying to build or test part of a new product but runs into the quota limit either has to spend time trying to find or free up enough space or has to get in touch with an SA and argue for a quota increase. If the engineer is near a deadline, this time loss could result in the whole product schedule slipping. If your customers are resistant to quotas, listen to their rationale, and see whether there is a common ground that you can reach, such as emergency increase requests with a guaranteed turnaround time. Although you need to understand each individual's needs, you also need to look at the big picture. Implementing quotas on a server in a way that prevents another person from doing his or her job is not a good idea.

43.2.4 Developing an Inventory and Spares Policy

Most sites have some kind of inventory of common parts. We discuss spares in general in [Section 26.7.2](#), but storage deserves a bit of extra attention.

There used to be large differences between the types of drives used in storage systems and the ones in desktop systems. These variations meant that it was much easier for SAs to dedicate a particular pool of spare drives to infrastructure use. Now that many storage arrays and workgroup servers are built from off-the-shelf parts, those drives on the shelf might be spares that could be used in either a desktop workstation or a workgroup storage array. A common spares pool is usually considered a good thing. However, such a practice may seem arbitrary to a customer who is denied a new disk but sees one sitting on a shelf unused, reserved for the next server failure.

How can SAs make sure to reserve enough drives as spares for vital shared storage while not hoarding drives that are also needed for new desktop systems or individual customer needs? Managing these competing needs is something of a balancing act, and an important component is a policy that addresses how spares will be distributed. Few SAs are able to stock as many spares as they would like to have around, so having an equitable system for allocating them is crucial.

It's best to separate general storage spares from infrastructure storage spares. You can make projections for either type, based on failures observed in the past on similar equipment. If you are tracking shared storage usage—and you should be, to avoid surprises—you can make some estimates on how often drives fail, so that you have adequate spares.

For storage growth, include not only the number of drives required to extend your existing storage, but also whatever server upgrades, such as CPU and memory, might be needed. If you plan to expand by acquiring whole new systems, such as stand-alone network storage arrays, be sure to include spares for those systems through the end of the fiscal year when they will be acquired.

43.2.5 Planning for Future Storage

The particularly tricky aspect of storage spares is that a customer asking for a drive almost always needs something more than simply a drive. A customer whose system disk has failed really needs a new drive, along with a standardized OS install. A customer who is running out of shared disk space and wants to install a private drive really needs more shared space or a drive, along with backup services. And so on.

SAs should not have a prove-that-you-need-this mentality. SAs should strive

to be enablers, not gatekeepers. That said, you should be aware that every time a drive goes out the door of your storage closet, it is likely that something more is required. Another way to think of this scenario is that a problem you know how to solve is happening now, or a problem that you might have to diagnose later is being created. Which one would you rather deal with?

Fortunately, as shown throughout this book, it's possible to structure the environment so that such problems are more easily solved by default. If your site chooses to back up individual desktops, some backup software lets you configure it to automatically detect a new local partition and begin backing it up unless specifically prevented from doing so. Make network boot disks available for customers, along with instructions on how to use them to load your site's default supported installation onto the new drive. This approach lets customers replace their own drives and still get a standardized OS image. Designate a planned quarterly maintenance window to give you the opportunity to upgrade shared storage to meet projected demands before customers start becoming impacted by lack of space. Thinking about storage services can be a good way to become aware of the features of your environment and the places where you can improve service for your customers.

43.2.6 Establishing Storage Standards

Standards help you to say no when someone shows up with random equipment and says, "Please install this for me." If you set storage standards, people are less likely to be able to push a purchase order for nonstandard gear through accounting and then expect you to support whatever they got.

The wide range of maturity of various storage solutions means that finding one that works for you is a much better strategy than trying to support anything and everything out there. Having a standard in place helps to keep one-off equipment out of your shop.

A standard can be as simple as a note from a manager saying, "We buy only IBM," or as complex as a lengthy document detailing requirements that a vendor and that vendor's solution must meet to be considered for purchase. The goal of standards is to ensure consistency by specifying a process, a set of characteristics, or both.

Standardization has many benefits, ranging from allowing a company to keep a common spares pool to minimizing the number of different systems that an SA must cope with during systems integration. As you progress to having a storage plan that accounts for both current and future storage needs, it is important to

address standardization. Some organizations can be very difficult places to implement standards control, but it is always worth the attempt. Since the life cycle of many systems is relatively short, a heterogeneous shop full of differing systems can become a unified environment in a relatively short period of time by setting a standard and bringing in only equipment that is consistent with the standard.

If your organization already has a standards process in place for some kinds of requests or purchases, start by learning that system and add standards to it. There may be sets of procedures that must be followed, such as meetings with potential stakeholders, creation of written specifications, and so on.

If your organization does not have a standards process, you may be able to get the ball rolling for your department. Often, you will find allies in the purchasing or finance departments, as standards tend to make their jobs easier. Having a standard in place gives them something to refer to when unfamiliar items show up on purchase orders. It also gives them a way to redirect people who start to argue with them about purchasing equipment—namely, by referring those people to either the standard itself or the people who created it.

Start by discussing, in the general case, the need for standards and a unified spares pool with your manager and/or the folks in finance. Request that they route all purchase orders for new types of equipment through the IT department before placing orders with the vendor. Be proactive in working with department stakeholders to establish hardware standards for storage and file servers. Make yourself available to recommend systems and to work with your customers to identify potential candidates for standards.

This strategy can prevent the frustration of dealing with a one-off storage array that won't interoperate with your storage network switch, or some new interface card that turns out to be unsupported under the version of Linux that your developers are using. The worst way to deal with attempts to bring in unsupported systems is to ignore customers and become a bottleneck for requests. Your customers will become frustrated and feel the need to route around you to try to address their storage needs directly.

Upgrading to a larger server often results in piles of old disks or storage subsystems that are no longer used. These items must be fully erased before they are discarded. Often, we hear stories of used disks purchased on eBay and then found to be full of credit card numbers or proprietary company information. Improper disposal of storage media can disqualify an organization from PCI compliance, or result in legal action.

Financial decision makers usually prefer to see the equipment reused internally. Here are some suggested uses:

- Use the equipment as spares for the new storage array or for building new servers.
- Configure the old disks as local scratch disks for write-intensive applications, such as software compilation.
- Increase the reliability of key servers by installing a duplicate OS to reboot from if the system drive fails.
- Convert some portion to swap space, if your OS uses swap space.
- Create a build-it-yourself RAID for nonessential applications or temporary data storage.
- Create a global temp space, accessible to everyone, called *homenot_backed_up*. People will find many productivity-enhancing uses for such a service. The name is important: People need a constant reminder if they are using disk space that has no reliability guarantee.

43.3 Storage as a Service

Rather than considering storage to be an object, think of it as one of the many services. Then, you can apply all the standard service basics. To consider something a service, it needs to have an SLA and to be monitored to see that the availability adheres to that SLA.

43.3.1 A Storage SLA

What should go into a storage SLA? An engineering group might need certain amounts of storage to ensure that automated release builds have enough space to run daily. A finance division might have minimal day-to-day storage needs but require a certain amount of storage quarterly for generating reports. A QA group or a group administering timed exams to students might express its needs in response time as well as raw disk space.

SLAs are typically expressed in terms of availability and response time. Expressing SLAs in a measurable, quantifiable unit such as input/output operations per second (IOPS) is critical. Availability for storage could be thought of as both reachability and usable space. Response time is usually measured as latency—the time it takes to complete a response—at a given load. An SLA should also specify mean time to repair (MTTR) expectations.

Use standard benchmarking tools to measure these metrics. This approach has

the advantage of repeatability as you change platforms. The system should still be tested in your own environment with your own applications to make sure that the system will behave as advertised, but at least you can insist on a particular minimum benchmark result to consider the system for an in-house evaluation that will involve more work and commitment on the part of you and the vendor.

43.3.2 Reliability

Everything fails eventually. You can't prevent a hard drive from failing. You can give it perfect, vendor-recommended cooling and power, and it will still fail eventually. You can't stop an HBA from failing. Now and then, a bit being transmitted down a cable gets hit by a gamma ray and is reversed. If you have eight hard drives, the likelihood that one will fail tomorrow is eight times more likely than if you had only one. The more hardware you have, the more likely a failure is to occur. Sounds depressing, but there is good news: There are techniques to manage failures that can bring about any reliability level required.

The key is to decouple a component failure from an outage. If you have one hard drive, its failure results in an outage: a 1:1 ratio of failures to outages. However, if you have eight hard drives in a RAID 5 configuration, a single failure does not result in an outage. Two failures, one happening faster than a hot spare can be activated, are required to cause an outage. In this scenario, we have successfully decoupled component failure from service outages. (A similar strategy can be applied to networks, computing, and other aspects of system administration.)

The configuration of a storage service can increase its reliability. In particular, certain RAID levels increase reliability, and NASs can also be configured to increase overall reliability.

The benefit of centralized storage (NAS or SAN) is that the extra cost of reliability is amortized over all users of the service.

RAID and Reliability

All RAID levels except for RAID 0 increase reliability. The data on a redundant RAID set continues to be available even when a disk fails. In combination with an available hot spare, a redundant RAID configuration can greatly improve reliability.

It is important to monitor the RAID system for disk failures, however, and to keep in stock some replacement disks that can be quickly swapped in to replace the failed disk. Every experienced SA can tell a horror story of a RAID system

that was unmonitored and had a failed disk go unreplaced for days. Finally, a second disk dies, and all data on the system is lost. Many RAID systems can be configured to shut down after 24 hours of running in degraded mode. It can be safer to have a system halt safely than to go unmonitored for days.

NAS and Reliability

NAS servers generally support some form of RAID to protect data, but NAS reliability also depends on network reliability. Most NAS systems have multiple network interfaces. For even better reliability, connect each interface to a different network switch.

Costs of Reliability

When asked, most customers ask for 100 percent reliability. Realistically, however, few managers want to spend what it takes to get the kind of reliability that their employees say they would like. Additional reliability is exponentially more expensive. A little extra reliability costs a lot, and perfect reliability is more costly than most people can imagine. The result is sticker shock when researching various storage uptime requirements.

Providers of large-scale reliability solutions stress the uptime and ease of recovery when using their systems and encourage you to calculate the cost of every minute of downtime that their systems could potentially prevent. Although their points are generally correct, these savings must be weighed against the level of duplicated resources and their attendant cost. That single important disk or partition will have a solution requiring multiple sets of disks. In an industry application involving live service databases, such as financial, health, or e-commerce, one typically finds at least two mirrors: one local to the datacenter and another at a remote datacenter. Continuous data protection (CDP), discussed later, is the most expensive way to protect data and, therefore, is used only in extreme situations.

High-availability data service is expensive. It is the SA's job to make management aware of the costs associated with storage uptime requirements, work it into return on investment (ROI) calculations, and leave the business decision to management. Requirements may be altered or refocused to get the best possible tradeoff between expense and reliability.

43.3.3 Backups

One of the most fundamental components of a storage service is the backup strategy. [Chapter 44](#), “[Backup and Restore](#),” is dedicated to backups; here, we

simply point out some important issues related to RAID, NAS, and SAN systems.

RAID Is Not a Backup Strategy

RAID can be used to improve reliability, but it is important to realize that RAID is not a substitute for a backup strategy. For most RAID configurations, if two disks fail, all the data is lost. Fires, earthquakes, floods, and other disasters will result in all data being lost. A brownout can damage multiple disks or even the RAID controller. Buggy vendor implementations and hardware problems could also result in complete data loss.

Your customers can, and will, delete critical files. When they do, their mistake will be copied to the mirror or parity disk. Some RAID systems include the ability to take file snapshots—that is, the ability to view the file system as it was days ago. This is also not a backup solution. It is simply an improvement to the customer-support process when a customer accidentally deletes a file. If those snapshots are stored on the same RAID system as the rest of the data, a disaster or double-disk failure will wipe out all data.

Backups to some other medium, be it tape or even another disk, are still required when you have a RAID system, even if it provides snapshot capabilities. Neither RAID nor a snapshot will help you recover a RAID set after a fire in your datacenter.

It is a very common mistake to believe that acquiring a RAID system means that you no longer have to follow basic principles for data protection. Don't let one of these disasters happen to you!

Whither Backups?

Once, Strata sourced a RAID system for a client without explicitly checking how backups would be done. She was shocked and dismayed to find that the vendor claimed that backups were unnecessary! The vendor did plan—eventually—to support a tape device for the system, but that would not be for at least a year. Adding a high-speed interface card to the box—to keep backups off the main computing network—was an acceptable workaround for the client. When purchasing a storage system, ask about backup and restore options.

RAID Mirrors as Backups

Rather than using a mirror to protect data all the time, some systems break, or

disconnect, the mirrored disks so they have a static, unchanging copy of the data to perform backups on. This is done in coordination with database systems and the OS to make sure that the data mirror is in a consistent state from an application point of view. Once the backups are complete, the mirror set is reattached and rebuilt to provide protection until the next backup process begins. The benefit of this approach is that backups do not slow down normal data use, since they affect only disks that are otherwise unused. The downside is that the data is not protected during the backup operation, and the production system runs much more slowly while the mirror is being rebuilt.

Many SAs use such mirroring capabilities to make an occasional backup of an important disk, such as a server boot disk, in case of drive failure, OS corruption, security compromise, or some other issue. Since any error or compromise would be faithfully mirrored onto the other disk, the system is not run in true RAID 1 mirror mode. The mirror is established and then broken so that it will not be updated. After configuration changes, such as OS patches, are made and tested, the mirror can be refreshed and then broken again to preserve the new copy. This strategy is better than restoring from a tape, because it is faster. It is also more accurate, since some tape backup systems are unable to properly restore boot blocks and other metadata.

RAID Mirrors to Speed Backups

A RAID set with two mirrors can be used to make backups faster. Initially, the system has identical data on three sets of disks, known as a **triple-mirror** configuration. When it is time to do backups, one mirror set is broken off, again in coordination with database systems and the OS to make sure that the data mirror is in a consistent state. The backup can then be done on the mirror that has been separated.

Done this way, backups will not slow down the system. When the backup is complete, the mirror is reattached, the rebuild happens, and the system is soon back to its normal state. The rebuild does not affect the performance of the production system as much, because the read requests can be distributed between the two primary mirrors.

NAS and Backups

In a NAS configuration, typically no unique data is stored on client machines; if data is stored there, it is well advertised that it is not backed up. This practice introduces simplicity and clarity into that site, especially in the area of backups. It is clear where all the shared customer data is located, and as such, the backup

process is simpler.

In addition, because shared customer data is stored on the NAS servers, the load for backing up this data is shared primarily by the NAS server itself and the server responsible for backups; thus, this load is isolated from the application servers and departmental servers. In this configuration, clients become interchangeable. If someone's desktop PC dies, the person should be able to use any other PC instead.

SANs and Backups

As mentioned previously, SANs make backups easier in two ways. First, a tape drive can be a SAN-attached device. Thus, all servers can share a single, expensive tape library solution. Second, by having a dedicated network for file traffic, backups do not interfere with normal network traffic.

SAN systems often have features that generate snapshots of LUNs. By coordinating the creation of those snapshots with database and other applications, the backup can be done offline, during the day, without interfering with normal operations.

43.3.4 Monitoring

If it isn't monitored, it isn't a service. Although we cover monitoring extensively in [Chapter 38](#), "[Service Monitoring](#)," it's worth noting here some special requirements for monitoring storage service.

A large part of being able to respond to your customers' needs is building an accurate model of the state of your storage systems. For each storage server, you need to know how much space is used, how much is available, and how much more the customer anticipates using in the next planning time frame. Set up historical monitoring so that you can see the level of change in usage over time, and get in the habit of tracking it regularly. Monitor storage-access traffic, such as local read/write operations or network file access packets, to build up a model that lets you evaluate performance. You can use this information proactively to prevent problems and to plan for future upgrades and changes.

Seeing monitoring data on a per-volume basis is typical and most easily supported by many monitoring tools. Seeing the same data by customer group allows SAs to do a better job of giving each group individualized attention and allows customers to monitor their own usage.

In addition to receiving notifications about outages or system/service errors, you should be alerted to such events as a storage volume reaching a certain

percentage of utilization or spikes or troughs in data transfers or in network response. Monitoring CPU usage on a dedicated file server can be extremely useful, as one sign of file services problems or out-of-control clients is an ever-climbing CPU usage. With per-group statistics, notifications can be sent directly to the affected customers, who can then do a better job of self-managing their usage. Some people prefer to be nagged over strictly enforced space quotas.

By implementing notification scripts with different recipients, you can emulate having hard and soft quotas. When the volume reaches, for instance, 70 percent full, the script could notify the group or department email alias containing the customers of that volume. If the volume continues to fill up and reaches 80 percent full, perhaps the next notification might go to the group's manager, to enforce the cleanup request. It might also be copied to the helpdesk or ticket alias so that the site's administrators know that there might be a request for more storage in the near future.

To summarize, the following are storage-related items that SAs should monitor:

- **Disk failures:** With redundant RAID systems, a single disk failure will not cause the service to stop working, but the failed disk must be replaced quickly or a subsequent failure may cause loss of service.
- **Other outages:** Monitor access to every network interface on a NAS, for example.
- **Space used/space free:** The most frequently asked customer question is about how much space is used or how much remains. By providing this information to customers on demand on the internal web site, you will be spared many tickets!
- **Rate of change:** This data is particularly helpful in predicting future needs. By calculating the rate of usage change during a typical busy period, such as quarterly product releases or the first semester of a new academic year, you can gradually arrive at metrics that will allow you to predict storage needs with some confidence.
- **I/O local usage:** Monitoring this value will let you see when a particular storage device or array is starting to become fully saturated. If failures occur, comparing the timing with low-level I/O statistics can be invaluable in tracking down the problem.
- **Network local interface:** If a storage solution begins to be slow to respond, comparing its local I/O metrics with the network interface metrics and network bandwidth used provides a clue as to where the scaling failure may be occurring.

- **Networking bandwidth usage:** Comparing the overall network statistics with local interface items, such as network fragmentation and reassembly, can provide valuable clues for optimizing performance. It is usually beneficial to specifically monitor storage-to-server networks and aggregate the data in such a way as to make it easily viewable outside the main network statistics area.
- **File service operations:** Providing storage services via a protocol such as NFS or CIFS requires monitoring the service-level statistics as well, such as NFS badcall operations.
- **Lack of usage:** When a popular file system has not processed any file service operations recently, it often indicates some other problem, such as an outage between the file server and the clients.
- **Individual resource usage:** This item can be a blessing or a slippery slope, depending on the culture of your organization. If customer groups self-police their resources, it is almost mandatory. First, customers care greatly about the data, so it's a way of honoring their priorities. Second, they will attempt to independently generate the data anyway, which loads the machines. Third, it is one fewer reason to give `root` privilege to non-SAs. Using `root` for disk-usage discovery is a common reason cited why engineers and group leads “need” `root` access on shared servers.

43.3.5 SAN Caveats

Since SAN technologies are always changing, it can be difficult to make components from different vendors interoperate. Try sticking with one or two vendors, and test their product offerings extensively. Focusing on a small number of vendors helps to establish a rapport. If you are a regular customer, their sales engineers will have more motivation to support you.

It's best to subject new models to significant testing before you integrate them into your infrastructure, even if they are from the same vendor. Vendors acquire outside technologies, change implementation subsystems, and occasionally make mistakes just like any other company. Vendors' goals are generally to improve their product offerings, but sometimes the new offerings are not considered improvements by folks like us.

Create a set of tests that you consider significant for your environment. A typical set might include industry-standard benchmark tests, application-specific tests obtained from application vendors, and attempts to run extremely site-specific operations, along with similar operations at much higher loads.

Paul Kilmartin, director of availability and performance engineering at eBay, gave this advice during his LISA 2003 keynote presentation: When vendors offer to show you their latest and greatest products, kick them out. Your data is too valuable to take risks. Tell such vendors that you want to see only the stuff that has been used in the field for a while. Let other people work through the initial product bugs. This is your data, the most precious asset your company has. Not a playground.

43.4 Performance

Performance means how long it takes for your customers to read and write their data. If the storage service you provide is too slow, your customers will find a way to work around it, perhaps by attaching extra disks to their own desktops or by complaining to management.

The most important rule of optimization is to measure first, optimize based on what was observed, and then measure again. Often, we see SAs optimize based on guesses of what is slowing a system down and make changes that do not actually improve performance. Measuring means using operating system tools to collect data, such as which disks are the most busy or the percentage of reads versus writes. Some SAs do not measure but simply try various techniques until they find one that solves the performance problem. These SAs waste a lot of time with solutions that do not produce results. Blind guessing can do more harm than good.

The primary tools that an SA has to optimize performance are RAM, SSD, and spindles. RAM is fastest, then SSD, then traditional disk. With more RAM, one can cache more and use the disk less. With more spindles, the load can be spread out over more disks working in parallel.

General Rules for Performance

The general rules for I/O performance can be summarized as follows:

- Never hit the network if you can stay on disk.
 - Never hit the disk if you can stay in memory.
 - Never hit memory if you can stay on chip.
 - Have enough money, and don't be afraid to spend it.
-

43.4.1 RAID and Performance

We've already discussed how RAID affects performance. The important thing to

remember is that read performance is improved by spreading it over more spindles because doing so improves parallelism. Therefore RAID improves read performance for all RAID levels under normal circumstances.

Write performance is also improved for RAID 0 since the writes are spread out over multiple spindles. RAID 1 requires double the number of writes, one for each mirror, but they are done in parallel so this usually has the same performance as writing to a single disk. The difference, however, is that the write is not complete until the last mirror is updated. If one mirror writes more slowly than the other, the net result is slower writes.

Writes in RAID 2 and higher can be much slower, as computing the RAID parity requires reading the block in the same position on all other disks. However, the other blocks are read in parallel. Some systems do some very smart optimizations. The other blocks do not need to be read if their contents are in cache memory or if a batch of writes is being performed that fills an entire track.

Performance drops dramatically if a disk has failed. This situation is called degraded mode because the system is slower due to the various recovery methods that come into play. If RAID 1 has a single failed disk, the system has the same performance as a single disk. That is, you lose the performance benefits of striping across many disks. RAID 2 and higher now must calculate the missing data by reading all remaining disks—which can be significantly slower than a normal read operation.

Once a broken disk is replaced, the RAID subsystem must be rebuilt. For RAID 1, this means reading every block of the good mirror and writing it to the new disk. RAID 2 and higher go through a similar process, reading every block of every disk, calculating the missing data or parity, and writing it to the new disk. The disk operations involved in rebuilding a RAID set compete with other disk operations and reduce performance. Most RAID systems can adjust the priority of the rebuilding. Of course, the lower the priority given to this operation, the longer the rebuilding will take. This leaves the system vulnerable to a second disk failure for a longer amount of time.

43.4.2 NAS and Performance

NAS-based storage allows SAs to isolate the file service workload away from other servers, making it easy for SAs to consolidate customer data onto a few large servers rather than have it distributed all over the network. In addition, applying consistent backup, usage, and security policies to the file servers is

easier.

Many sites grow their infrastructures somewhat organically, over time. It is very common to see servers shared within a department or particular user group, with the server providing both computing and file-sharing services. Moving file-sharing services to a NAS box can significantly reduce the workload on the server, improving performance for the customers. File-sharing overhead is not completely eliminated, as the server will now be running a client protocol to access the NAS storage. In most cases, however, there are clear benefits from such separation.

43.4.3 SSDs and Performance

SSDs are generally more expensive than traditional spinny disks. A common strategy is to use SSDs where performance is important and to use traditional spinny disks elsewhere.

Imagine your database application is built using spinny disks. Suppose it requires five replica servers to process all the transactions it receives and stay within the expected performance parameters. Moving to SSDs would permit the same volume of work to be processed with only two replica servers. Thus, the break-even point occurs when the increased cost of using SSDs is equal to the cost of the three servers that can be eliminated. Imagine if servers cost \$10,000 each. As long as the increased cost of SSDs is less than \$30,000, SSDs are the less expensive solution.

For most applications, this price-point was surpassed around 2010. The only way to know for sure is to benchmark your application both ways and do the math. As SSDs become faster and less expensive, the business case for SSDs will only become more obvious.

43.4.4 SANs and Performance

Because a SAN moves file traffic off the main network to a dedicated SAN network, using a SAN reduces network congestion on the primary network.

Sites were building their own versions of SANs long before anyone knew to call them that, using multiple high-speed interfaces on file servers and segregating file service traffic to an isolated network. Christine and Strata were co-workers at a site that was an early adopter of this concept. The server configurations had to be done by hand, with a bit of magic in the automount maps and in the local host and DNS entries, but the improved performance was worth the effort.

SANs have been so useful that people have started to consider other ways in which storage devices might be networked. One technique is to treat other networks as if they were direct cabling. Each SCSI command is encapsulated in a packet and sent over a network. Fiber channel (FC) does this using copper or fiber-optic networks. The fiber channel becomes an extended SCSI bus, and devices on it must follow normal SCSI protocol rules. The success of fiber channel and the availability of cheap, fast TCP/IP network equipment led to the creation of **iSCSI**, which sends basically the same packet over an IP network. This allows SCSI devices, such as tape libraries, to be part of a SAN directly. ATA over Ethernet (AoE) does something similar for ATAbased disks.

With the advances in high-speed networking and the increased affordability of the equipment, protocol encapsulations requiring a responsive network are now feasible in many cases. Expect to see the use of layered network storage protocols, along with many other types of protocols, increase in the future.

Since a SAN is essentially a network with storage, SANs are not limited to one facility or datacenter. Using high-speed, long-distance, networking technologies such as SONET, a SAN can be “local” to multiple datacenters at different sites.

43.4.5 Pipeline Optimization

An important part of understanding the performance of advanced storage arrays is appreciating how they manage a **data pipeline**. In a data pipeline, items that might be needed next are loaded into memory ahead of need so that access times are minimized. CPUs use this technique as part of their caches, as described in [Section 15.2.2](#). This is why, for some CPU-intensive jobs, a Pentium III with a large L2 cache could outperform a Pentium IV.

Pipelining algorithms are extensively implemented in many components of modern storage hardware, especially in the HBA but also in the drive controller. These algorithms may be *dumb* or *smart*. A so-called dumb algorithm has the controller simply read blocks physically located near the requested blocks, on the assumption that the next set of blocks that are part of the same request will be those blocks. This tends to be a good assumption, unless a disk is badly fragmented. A smart pipelining algorithm may be able to access the file system information and pre-read blocks that make up the next part of the file, whether they are nearby or not. Note that for some storage systems, “nearby” may not mean *physically near* the other blocks on the disk, but rather *logically near* them. Blocks in the same cylinder are not physically nearby, but are logically nearby, for example.

Although the combination of OS-level caching and pipelining yields excellent performance when reading data, writing data is a more complex process. Operating systems are generally designed to ensure that data writes are *atomic*, or at least as near-atomic as possible, given the actual hardware constraints. Atomic, in this case, means in one piece. Atoms were named before people recognized the existence of subatomic physics, with its protons, electrons, neutrons, and such. People thought of an atom as the smallest bit of matter, which could not be subdivided further.

This analogy may seem odd, but in fact it's quite relevant. Just as atoms are made up of protons, neutrons, and electrons, so a single write operation can involve a lot of steps. It's important that the operating system not record the write operation as complete until all the steps have completed. This means waiting until the physical hardware sends an acknowledgment, or ACK, that the write occurred.

One optimization is to ACK the write immediately, even though the data hasn't been safely stored on disk. That's risky, but there are some ways to make it safer. One way is to do this only for data blocks, not for directory information and other blocks that would corrupt the file system. (We don't recommend this technique, but it is an option on some systems.) Another way is to keep the data to be written in RAM that, with the help of a battery, survives reboots. Then the ACK can be sent as soon as the write is safely stored in that special RAM. In this case, it is important that the pending blocks be written before the RAM is removed. Tom once moved such a device to a different computer, not realizing that it was full of pending writes. When the new computer booted up, the pending writes were written onto the unsuspecting disk of the new system, which was then corrupted badly. Another type of failure might involve the hardware itself. A failed battery that goes undetected can create a disaster after the next power failure.

sync Three Times Before halt

Extremely early versions of Unix did not automatically sync the write buffers to disk before halting the system. The SAs would be trained to kick all the users off the system to acquiesce any write activity, then manually type the `sync` command three times before issuing the `shutdown` command. The `sync` command is guaranteed to schedule only the unwritten blocks for writing; there can be a short delay before all the blocks are finally written to disk. The second and third `sync` commands weren't actually needed, but were done to force the SA to

delay for long enough for all the blocks to be written to disk.

43.5 Evaluating New Storage Solutions

Whether a particular storage solution makes sense for your organization depends on how you are planning to use it. Study your usage model to make an intelligent, informed decision. Consider the throughput and configuration of the various subsystems and components of the proposed solution.

Look especially for hidden gotcha items. Some solutions billed as being affordable get that way by using your server's memory and CPU resources to do much of the work. If your small office or workgroup server is being used for applications as well as for attaching storage, obviously a solution of that type would be likely to prove unsatisfactory.

Test All Parts of a New System

Testing a new storage system doesn't mean testing just the storage aspects of the system. An example cited on an SA mailing list mentioned that a popular controller used in SATA arrays sent malformed email alerts, which the site's email system silently discarded. If a site administrator had not tested the notification system, warnings would have gone missing and the system would have limped along until it completely failed.

43.5.1 Drive Speed

A common problem is finding that an attractively priced system is using very slow drives and that the vendor did not guarantee a specific drive speed. It's not uncommon for some small vendors that assemble their own boxes to use whatever is on hand and then give you a surprise discount, based on the less-desirable hardware. That lower price is buying you a less useful system.

Although the vendor may insist that most customers don't care, that is not the case for your situation. Insist on specific standards for components, and check the system before accepting its delivery. The likelihood of receiving undesirable hardware increases when nonstandard parts are used, complicating the vendor's in-house assembly process. Be polite but firm in your insistence on getting what you ordered.

43.5.2 Fragmentation

Moving the disk arm to a new place on the disk is extremely slow compared to

reading data from the track where the arm is. Therefore, operating systems make a huge effort to store all the blocks for a given file in the same track of a disk. Since most files are read sequentially, this can result in the data being quickly streamed off the disk.

However, as a disk fills, it can become difficult to find contiguous sets of blocks to write a file. File systems inevitably become fragmented over time. Previously, SAs spent a lot of time defragmenting drives by running software that moved files around, opening up holes of free space and moving large, fragmented files to the newly created contiguous space.

This is not a worthwhile way for SAs to spend their time on modern operating systems. Modern systems are much better at not creating fragmented files in the first place. Hard drive performance is much less affected by occasional fragments. Defragmenting a disk puts it at huge risk owing to potential bugs in the software and problems that can come from power outages while critical writes are being performed.

We doubt vendor claims of major performance boosts through the use of their defragmenting software. The risk of destroying data is too great. To echo Paul Kilmartin's words, this is important data, not a playground.

Fragmentation is a moot point on multiuser systems. Consider an NFS or CIFS server. If one user is requesting block after block of the same file, fragmentation might have a slight effect on the performance received, with network delays and other factors being much more important. A more typical workload would be dozens or hundreds of concurrent clients. Since each client is requesting individual blocks, the stream of requests sends the disk arm flying all over the disk to collect the requested blocks. If the disk is heavily fragmented or perfectly unfragmented, the amount of movement is about the same. Operating systems optimize for this situation by performing disk requests sorted by track number rather than in the order received. Since operating systems are already optimized for this case, the additional risk incurred by rewriting files to be less fragmented is unnecessary.

43.5.3 Storage Limits: Disk Access Density Gap

The density of modern disks is quite astounding. The space once occupied by a 500 MB MicroVAX disk can now house several terabytes of storage. However, the performance is not improving as quickly.

Improvements in surface technology are increasing the size of hard disks by 40 to 60 percent annually. Drive performance, however, is growing by only 10 to

20 percent. The gap between the increase in how much a disk can hold and how quickly you can get data on and off the disk is widening. This gap, which is known as **disk access density** (DAD), is a measurement of I/O operations per second per gigabyte of capacity (OPS/second/GB).

In a market where price/performance is so important, many disk buyers are mistaking pure capacity for the actual performance, completely ignoring DAD. DAD is important when choosing storage for a particular application. Ultra-high-capacity drives are wonderful for relatively low-demand resources. Applications that are very I/O intensive, especially on writes, require a better DAD ratio.

As you plan your storage infrastructure, you will want to allocate storage servers to particular applications to ensure that your system delivers optimal performance. It can be tempting to purchase the largest hard disk on the market, but two smaller disks will provide better performance. This reality is especially disappointing when one considers the additional power, chassis space, and cooling that are required for the extra disk.

A frequently updated database may potentially be structured so that the busiest tables are assigned to a storage partition made up of many smaller, higher-throughput drives. Engineering file systems subject to a great deal of compilation but also having huge data models, such as the file systems of a chip-design firm, may require thoughtful integration with other parts of the infrastructure.

When supporting customers who seem to need both intensive I/O and high-capacity data storage, you will have to look at your file system performance closely and try to meet the customers' needs cleverly.

43.5.4 Continuous Data Protection

CDP is the process of copying data changes in a specified time window to one or more secondary storage locations. That is, by recording every change made to a volume, you can roll forward and back in time by replaying and undoing the changes. In the event of data loss, you can restore the last backup and then replay the CDP log to the moment you want. The CDP log may be stored on another machine, maybe even in another building.

Increasingly, CDP is being used not in the context of data protection, but rather in the area of service protection. The data protection is a key element of CDP, but many implementations also include multiple servers running applications that are tied to the protected data.

Any CDP solution is a process as much as a product. Vendor offerings usually consist of management software, often installed with the assistance of the vendor's professional services division, to automate the process. Several large hardware vendors offer CDP solutions that package their own hardware and software with modules from other vendors to provide vendor-specific CDP solutions supporting third-party applications, such as database transaction processing.

CDP is commonly used to minimize recovery time and reduce the probability of data loss. It is generally quite expensive to implement reliably, so a site tends to require compelling reasons to implement it. Typically, sites implement CDP for one of two reasons: to become compliant with industry-specific regulations or to prevent revenue losses and/or liability arising from outages.

CDP is new and expensive, so it is generally reserved for solving only problems that cannot be solved any other way. One market for CDP is where the data is extremely critical, such as financial information. Another is where the data changes at an extremely high rate. If losing a few hours of data means trillions of updates, the adoption of CDP can be easier to justify.

43.6 Common Data Storage Problems

Modern storage systems use a combination of layered subsystems and per-layer optimizations to provide fast, efficient, low-maintenance storage—most of the time. This section highlights common ways in which storage solutions can turn into storage problems.

Many of the layers in the chain of disk platter to operating system to client are implemented with an assumption that the next layer called will do the right thing and somehow recover from an error by requesting the data again.

The most common overall type of problem is that some boundary condition has not been taken into account. A cascading failure chain begins, usually in the normal layer-to-layer interoperation, but sometimes, as in the case of power or temperature problems, at a hardware level.

43.6.1 Large Physical Infrastructure

Modern storage solutions tend to pack a significant amount of equipment into a comparatively small space. Many machine rooms and datacenters were designed based on older computer systems, which occupied more physical space. When the same space is filled with multiple storage stacks, the power and cooling demands can be much higher than the machine room design specifications.

Many mysterious failures have ultimately been traced to temperature or power issues.

When your system is experiencing mysterious failures involving corruption of arrays or scrambled data, it can make sense to check the stability of your power infrastructure to the affected machine. Include power readings in your storage monitoring for just this reason. We've been both exasperated and relieved to find that an unstable NAS unit became reliable once it was moved to a rack where it could draw sufficient power—more power than it was rated to draw, in fact.

A wattage monitor, which records real power use, can be handy when evaluating the requirements of storage units. Drives often use more power to start up than to run. A dozen drives starting at once can drain a shared PDU enough to generate mysterious faults on other equipment.

43.6.2 Timeouts

Timeouts can be a particularly troubling problem, especially in heavily optimized systems that are implemented primarily for speed rather than for robustness. NAS and SAN solutions can be particularly sensitive to changes in the configuration of the underlying networks.

A change in network configuration, such as a network topology change that puts an extra router hop in the storage path, may seem to have no effect when implemented and tested. However, under heavy load, that slight delay might be just enough to trigger TCP timeout mischief in the network stack of the NAS device.

Sometimes, the timeout may be at the client end. With a journaling file system served over the network from a heavily loaded shared server, Strata saw a conservative NFS client lose writes because the network stack timed out while waiting for the file system to journal them. When the application on the client side next requested it, the file received did not match; the client application crashed.

43.6.3 Saturation Behavior

Saturation of the data transfer path, at any point in the chain, is often the culprit in mysterious self-healing delays and intermittent slow responses, even triggering the timeouts mentioned previously. When doing capacity planning, take care not to confuse the theoretical potential of the storage system with the probable usage speeds.

A common problem, especially with inexpensive and/or poorly implemented

storage devices, is that of confusing the speed of the fastest component with the speed of the device itself. Some vendors may accidentally or deliberately foster this confusion.

Examples of statistics that are only a portion of the bigger picture include the following:

- Burst I/O speed of drives versus sustained I/O speeds (most applications rarely burst)
- Bus speed of the chassis
- Shared backplane speed
- Controller and/or HBA speed
- Memory speed of caching or pipelining memory
- Network speed

Your scaling plans should consider all these elements. The only reliable figures on which to base performance expectations are those obtained by benchmarking the storage unit under realistic load conditions.

A storage system that is running near saturation is more likely to experience unplanned interactions between delayed acknowledgments implemented in different levels of hardware and software. Since multiple layers might be performing in-layer caching, buffering, and pipelining, the saturation conditions increase the likelihood of encountering boundary conditions—among them, overflowing buffers and updating caches before their contents can be written. As mentioned earlier, implementers are probably relying on the unlikelihood of encountering such boundary conditions; how these types of events are handled is usually specific to a particular vendor's firmware implementation.

43.7 Summary

In this chapter, we discussed the most common types of storage and the benefits and appropriate applications associated with them. The basic principles of managing storage remain constant: Match your storage solution to a specific usage pattern of applications or customers, and build up layers of redundancy while sacrificing as little performance as possible at each layer.

Although disks have grown cheaper over time, managing them has become more expensive. Considering storage as a service allows you to put a framework around storage costs and agree on standards with your customers. To achieve that goal, you must have customer groups with which to negotiate those standards and, as in any service, perform monitoring to ensure the quality level

of the service.

The options for providing data storage to your customers have increased dramatically, allowing you to choose the level of reliability and performance required for specific applications. Understanding the basic relationship of storage devices to the operating system and to the file system gives you a richer understanding of the way that large storage solutions are built up out of smaller subsystems. Concepts such as RAID can be leveraged to build storage solutions that appear to a server as a simple, directly attached disk, but whose properties are highly tunable to optimize for the customer applications being served.

We also discussed the serious pending problem of disk density versus the bandwidth of disk I/O, an issue that will become more critical in the coming years.

Exercises

1. Which kinds of storage have you seen in use during your own lifetime? How many of them were “the next big thing” when introduced? Do you still have some systems at home?
2. Search for on-demand storage pricing. How do the features of the lowest-priced storage compare to those of the highest-priced option? Which price points do you find for various features?
3. How would you characterize your organization’s main storage systems, based on the taxonomy introduced in this chapter? Do you think that the current storage system used is a good match for your needs, or would another type be more useful?
4. Do you have a list of the common kinds of storage dataflow in your organization? What’s the ratio of reads to writes?
5. RAID 1 and higher use multiple drives to increase reliability. Eight drives are eight times more likely to have a single failure in a given time period. If a RAID 5 set had eight drives, do these two factors cancel each other out? Why or why not?
6. A hard drive is 10 times faster than RAM. Suppose that you had a huge database that required access that was as fast as RAM. How many disk spindles would be required to make 1,000 database queries per second as fast as keeping the database entirely in RAM? (Assume that the RAM would be on multiple computers, each of which could perform a share of the queries in parallel.) Look up current prices for disks and RAM, and calculate which would be less expensive if the database were 10 gigabytes,

1 terabyte, and 100 terabytes.

7. Which of the performance rules in the sidebar “[General Rules for Performance](#)” are addressed by the use of HBAs with storage? Explain.
8. Do you keep metrics on disk performance? If you had to improve the performance of your local storage solution, what are some places you might be able to make a difference without ripping the whole thing out and starting over?
9. Which RAID characteristics would you want for an array supporting real-time data collection from environmental sensors or factory monitoring, and why?
10. Are the storage services in your organization set up for optimal usage? Which kinds of changes would you make to improve the storage environment?

Chapter 44. Backup and Restore

Everyone hates backups. They are inconvenient. They are costly. Services run slower—or not at all—when servers are being backed up. However, customers *love* restores. Restores are why SAs perform backups.

Being able to restore lost data is a critical part of any environment. Data gets lost. Equipment fails. Humans delete data both by mistake and on purpose. Judges impound all lawsuit-related documents that were stored on your computers on a certain date. Shareholders require the peace of mind that comes with the knowledge that a natural or other disaster will not make their investment worthless. Data also gets corrupted by mistake, on purpose, and by gamma rays from space. Backups are like insurance: You pay for it even though you hope to never need it. In reality, you need it.

Although the goal is to be able to restore lost data in a timely manner, it is easy to get caught up in the daily operational work of doing backups and to forget that restoration is the goal. As evidence, the collective name typically used for all the equipment and software related to this process is “backup system.” It should really be called “backup and restore systems” or, possibly more fittingly, simply the “data restoration system.”

This book is different in the way it addresses backups and restores. Readers of this book should already know which commands their OSs use to back up and restore data. We do not cover that information. Instead, we discuss the theory of how to plan your backups and restores in a way that should be useful no matter which backup products are available.

After discussing the theory of planning the backups and restores, we focus on the three key components of modern backup systems: automation, centralization, and inventory management. These three aspects should help guide your purchasing decisions. Once the fundamentals are established, we discuss how to maintain the system you’ve designed well into the future.

The topic of backups and restores is so broad that we cannot cover the entire subject in detail. We have chosen to cover the key components. Books such as *Unix Backup and Recovery* ([Preston 1999](#)) and *Windows NT Backup and Restore* ([Leber 1998](#)) cover the details for Unix and Microsoft environments, respectively, in great detail.

Backup and restore service is part of any data storage system. One study found that the purchase price of a disk is a mere 20 percent of the total cost of

ownership, with backups being nearly the entire remaining cost. Buying a raw disk and slapping it into a system is easy. Providing data storage as a complete service is difficult. The price of disks has been decreasing, but the total cost of ownership has risen mostly because of the increasing cost of backups. Therefore, an efficient backup and restore system is your key to cost-effective data storage.

With regard to terminology, we use **full backup** to mean a complete backup of all files on a partition; Unix users call this a “level 0 backup.” The term **incremental backup** refers to copying all files that have changed since the previous full backup; Unix users call this a “level 1 backup.” They are often referred to simply as “incrementals.” Incremental backups grow over time. That is, if a full backup is performed on Sunday and an incremental backup each day of the week that follows, the amount of data being backed up should grow each day because Tuesday’s incremental backup includes all the files from Monday’s backup, as well as what changed since then. Friday’s incremental backup should include all the files that were part of Monday’s, Tuesday’s, Wednesday’s, and Thursday’s backups, in addition to what changed since Thursday’s backup. Some systems perform an incremental backup that collects all files changed since a particular incremental backup rather than the last full backup. We borrow the Unix terminology and call those *level 2 incremental backups* if they contain files changed since the last level 1 backup, or *level 3* if they contain files changed since the last level 2 backup, and so on.

Note that we refer to the backup media as tape in this chapter, even though we recognize that there are many alternatives.

44.1 Getting Started

Engineering your backup and restore system should begin by determining the desired end result and working backward from there. The end result is the desired restore capability of the system. Restores are requested for various reasons, and the reasons that apply to your environment affect further decisions, such as development of a policy and a schedule.

We start by defining corporate guidelines, which drive your SLA for restores based on your site’s needs, which becomes your backup policy, which dictates your backup schedule.

- The **corporate guidelines** define terminology and dictate minimums and requirements for data-recovery systems.
- The **SLA** defines the requirements for a particular site or application and is guided by the corporate guidelines.

- The **policy** documents the implementation of the SLA in general terms, written in English.
- The **procedure** outlines how the policy is to be implemented.
- The detailed **schedule** shows which disk will be backed up when. This may be static or dynamic. Such a schedule usually consists of the policy translated from English into the backup software's configuration.

Beyond policies and schedules are operational issues. Consumables can be expensive and should be included in the budget. Time and capacity planning are required to ensure that we meet our SLA during both restores and backups. The backup and restore policies and procedures should be documented from both the customer and the SA perspectives.

Only after all that is defined can we build the system. Modern backup systems have three key components: automation, centralization, and inventory management. Each of these is discussed in turn.

44.2 Reasons for Restores

Restores are requested for three reasons. If you do not understand them, the backup and restore system may miss the target. Each reason has its own requirements. The reasons are as follows:

- **Accidental file deletion:** A customer has accidentally erased one or more files and needs to have them restored.
- **Disk failure:** A hard drive has failed, and all data needs to be restored.
- **Archival:** For business reasons, a snapshot of the entire “world” needs to be made on a regular basis for disaster-recovery, legal, or fiduciary reasons.

It is interesting to note that the three types of restore requests typically serve three types of customers. Individual file restores serve customers who accidentally deleted the data—in other words, the direct users of the data. Archival backups serve the needs of the legal and financial departments that require them—people who are usually far detached from the data itself. Judges can't subpoena documents that aren't backed up. Complete restores after a disk failure serve the SAs who committed to providing a particular SLA. Backups for complete restores are therefore part of the corporate infrastructure.

In an environment that bills for services with a fine granularity, these kinds of backups can be billed for differently. If possible, these customer groups should be individually billed for these special requirements, just as they would be billed

for any service. Different software may be required, and there may be different physical storage requirements and different requirements for who “owns” the tapes.

Passing the Cost to the Right Customer

During a corporate merger, the U.S. Department of Justice required the companies involved to preserve any backup tapes until the deal was approved. This meant that old tapes could not be recycled. The cost of purchasing new tapes was billed to the company’s legal department. It required the special service, so it had to pay.

44.2.1 Accidental File Deletion

In the first case, customers would prefer to quickly restore any file as it existed at any instant. Unfortunately, that usually isn’t possible. However, when the data is on a SAN or NAS with the snapshot feature, it is usually possible to perform a self-service restore with reasonable time granularity. The customer may lose up to an hour of changes, but that is better than a day’s worth, and the data can be recovered quickly, without losing additional time waiting for an SA to perform the restore.

In an office environment without SAN or NAS, you can typically expect to be able to restore a file to what it looked like at any one-day granularity and to have it take three to five hours to have the restore completed. Even in those environments, it can still be possible to make the restore self-service, but it will be slower.

To an SA, the value of snapshots is that they reduce workload, because the most common type of request becomes self-service. To customers, the value of snapshots is that they give them new options for managing their work better. Customers’ work habits change as they learn they can rely on snapshots. If the snapshots are there forever, as is possible with CommVault, customers will manage their disk utilization differently, knowing that they can always get back what they delete. Even if snapshots are available going back only a fixed amount of time, customers tend to develop creative, new, and more efficient workflows.

Snapshots also increase customer productivity by reducing the amount of lost data that must be manually reconstructed. When they accidentally delete data, customers may reconstruct it rather than wait for the restore, which may take hours or even days. Everyone has made a change to a file and later regretted making the change. Reconstructing the file manually is an error-prone process,

but it would be silly to wait hours for a restore request to be completed. With snapshots, customers are less likely to attempt to manually reconstruct lost data.

The most common reason for requesting a restore is to recover from accidental file deletion. Making this case fast and self-service benefits everyone.

44.2.2 Disk Failure

The second kind of restore is related to disk failure—or any hardware or software failure resulting in total file system loss. A disk failure causes two problems: loss of service and loss of data. On critical systems, such as e-commerce and financial systems, RAID should be deployed so that disk failures do not affect service, with the possible exception of a loss in performance. However, in noncritical systems, customers can typically (in common office environments) expect the restore to be completed in a day. Although they do not like losing data, they usually find a single day of lost work to be an acceptable risk. Sometimes, the outage is between these two extremes: A critical system is still able to run, but data on a particular disk is unavailable. In that case, there may be less urgency.

This kind of restore often takes a long time to complete. Restore speed is slow because large volumes of data are being restored, and the entire volume of data is unavailable until the last byte is written. To make matters worse, a two-step process is involved: First the most recent full backup must be read, and then the most recent incremental(s) are read.

44.2.3 Archival Purposes

The third kind of restore request is archival. Corporate policies may require you to be able to reproduce the entire environment with a granularity of a quarter, half, or full year in case of disasters or lawsuits. The work that needs to be done to create an archive is similar to the full backups required for other purposes, with the following differences:

- Archives are full backups. In environments that usually mix full and incremental backups on the same tapes, archive tapes should not be so mixed.
- Some sites require archive tapes to be separate from the other backups. This may mean that archive tapes are created by generating a second, redundant set of full backups. Alternatively, archival copies may be generated by copying the full backups off previously made backup tapes. Although this alternative is more complicated, it can, if it is automated, be

performed unattended when the jukebox is otherwise unused.

- Archives are usually stored off-site.
- Archive tapes age more than other tapes. They may be written on media that will become obsolete and eventually unavailable. You might consider storing a compatible tape drive or two with your archives, as well as appropriate software for reading the tapes.
- If the archives are part of a disaster-recovery plan, special policies or laws may apply.

When making archival backups, do not forget to include the tools that go with the data. Tools get upgraded frequently, and if the archival backup is used to back up the environment, the tools, along with their specific set of bugs and features, should be included. Make sure that the tools required to restore the archive and the required documentation are stored with the archive.

Although there are some types of specialized backup and restore scenarios, most of them fit into one of three categories defined earlier in this section.

44.2.4 Perform Fire Drills

The only time you know the quality of your backup media is when you are doing a restore. This is generally the worst time to learn that you have problems. You can better assess your backup system if you do an occasional fire drill. Pick a random file and restore it from tape to verify that your process is working.

Automated Fire Drill Requests

The first time Tom saw a backup fire drill was when he was working with Tommy Reingold at Bell Labs. Tommy wrote a small program that would randomly select a server, then randomly select a file on that server, then email the SAs to ask for a copy of that file as it was a week ago. He was able to sleep better at night knowing that these weekly requests were satisfied successfully.

It can be useful to do an occasional fire drill that involves restoring an entire disk volume. The speed at which an entire disk volume can be restored is often unknown because it is so rarely requested. Restoring a file, or even a directory of files, as regularly requested by customers will not help you determine how long a full disk restore will take, because of the huge difference in the quantity of data being restored. Bottlenecks are likely to go unnoticed until an emergency happens. It is better to restore an entire disk volume occasionally than to

discover a bottleneck when you are under the gun to bring a system back into service. When doing these fire drills, it is important to time them and monitor such things as disk, tape, and network utilization. If you are not seeing the performance you expect, you can review the statistics you have collected to help determine what needs to be improved.

If you think that you don't have enough free disk space to do a full disk fire drill, you might want to do this kind of testing whenever you have installed a new server but before it goes into production. You should have at least one spare partition available, and the drill will be a good burn-in for the new hardware.

If some tapes are stored off-site, the fire drills should include both off-site and on-site tapes to completely exercise the system.

The person who verifies that the data recovered in the fire drill is valid should not be the same person who is responsible for performing the backups. This separation of duties creates a system of checks and balances.

44.3 Corporate Guidelines

Organizations need a corporate-wide document that defines terminology and dictates requirements for data-recovery systems. Global corporate policymakers should strive to establish minimums based on legal requirements rather than list every specific implementation detail of the items that are discussed later in this chapter.

The guideline should begin by defining why backups are required, what constitutes a backup, and which kind of data should be backed up. A set of retention guidelines should be clearly spelled out. There should be different SLAs for each type of data: finance, mission critical, project, general home directory data, email, experimental, and so on.

The guidelines should list a series of issues that each site needs to consider, so that they are not overlooked. For example, the guidelines should require sites to carefully plan when backups are done, not simply do them at the default "midnight until they complete" time frame. It wouldn't be appropriate to dictate the same window for all systems. Backups usually have a performance impact, so they should be done during off-peak times. E-commerce sites with a global customer base will have very different backup windows than offices with normal business schedules.

Backups Slow Down Services

In 1999, Lucent got some bad press for badly timed backups. The

company had outsourced its backups to a third party, which ran them at peak hours. This practice adversely affected the performance of the web server, annoying a technology columnist, who used the opportunity to write a long article about how dumb telecommunication companies are when it comes to the Internet. He assumed that the poor performance was due to a lack of bandwidth. Although your backup-related performance problems might not make the news, they can still be embarrassing. People remember the bad PR, not the remediation.

If you are the person writing the global corporate requirements document, you should begin by surveying various groups for requirements: Consult your legal department, your executive management, the SAs, and your customers. It becomes your job to reach consensus among them all. Use the three major types of restores as a way of framing the subject.

For example, the legal department might need archival backups to prove copyright ownership or intellectual property rights. Insurance might require general backups that are retained for at least six months. The accounting department might need to have tax-related data kept for seven years but recorded only on a quarterly basis. Increasingly, legal departments are requiring a short retention policy for email, especially in light of the fact that key evidence in the Microsoft lawsuit was gained by reviewing Microsoft's email archives. Most companies insist that email archives be destroyed after six months.

It is important to balance all these concerns. You might have to go through several rounds of surveys, revising the requirements until they are acceptable to all involved.

Some companies, especially start-ups, may be too small to have guidelines beyond "there will be backups." As the company grows, consider adopting corporate guidelines, based on the requirements of your investors and legal counsel.

44.4 A Data-Recovery SLA and Policy

The next step is to determine the service level that's right for your particular site. An SLA is a written document that specifies what kind of service and performance that service providers commit to providing. This policy should be written in cooperation with your customers. Once the SLA is determined, it can be turned into a policy specifying how the SLA will be achieved.

To establish an SLA, list the three types of restores, along with the desired

time to restoration, the granularity and retention period for such backups (that is, how often the backups should be performed and how long the tapes should be retained), and the window of time during which the backups may be performed (for example, midnight to 8 AM).

For most SAs, a corporate standard already exists, with vague, high-level parameters that they must follow. Make sure that your customers are aware of these guidelines. From there, building the policy is usually very straightforward.

The example SLA we use in the remainder of this chapter is as follows: Customers should be able to get back any file with a granularity of one business day for the past six months and with a granularity of one month for the last three years. Disk failures should be restored in four hours, with no more than two business days of lost data. Archives should be full backups on separate tapes generated quarterly and kept forever. Critical data will be stored on a system that retains user-accessible snapshots made every hour from 7 AM until 7 PM, with midnight snapshots held for one week. Databases and financial systems should have higher requirements that should be determined by the application's requirements and therefore are not within the scope of this example policy.

The policy based on this SLA would indicate that there will be daily backups and that the tapes will be retained as specified. The policy can determine how often full versus incremental backups will be performed.

44.5 The Backup Schedule

Now that we have an SLA and policy, we can set the schedule, which is specific and lists details down to which partitions of which hosts are backed up when. Although an SLA should change only rarely, the schedule changes often, tracking changes in the environment. Many SAs choose to specify the schedule by means of the backup software's configuration.

Continuing our example, backups should be performed every business day. Even if the company experiences a nonredundant disk failure and the last day's backups failed, we will not lose more than two days' worth of data. Since full backups take significantly longer than incremental backups, we schedule them for Friday night and let them run all weekend. On Sunday through Thursday nights, incremental backups are performed.

You may have to decide how often full backups are performed. In our example, the requirement is for full backups once a month. We could, in theory, perform one-quarter of our full backups each weekend. This leisurely rate would meet the requirements of our policy, but it would be unwise. As we noted earlier,

incremental backups grow over time until the next full backup is completed. The incremental backups would be huge if each partition received a full backup only once a month. It would save tape to perform a full backup more often.

However, backup software has become increasingly automated over the years. It is common to simply list all partitions that need to be backed up and to have the software generate a schedule based on the requirements. The backups are performed automatically, and email notification is generated when tapes must be changed.

Let's look at an example. Suppose that a partition with 4 TB of data is scheduled to have a full backup every 4 weeks (28 days) and an incremental backup all other days. Let's also assume that the size of our incremental backup grows by 5 percent every day. On the first day of the month, 4 TB of tape capacity is used to complete the full backup; on the second day, 200 GB; the third day, 400 GB; the fourth day, 600 GB; and so on. The tape capacity used on the eleventh and twelfth days is 2 TB and 2.2 TB, respectively, which total more than a full backup. This means that on the eleventh day, it would have been wiser to do a full backup.

[Table 44.1](#) shows this hypothetical situation in detail with daily, 7-day, 14-day, 21-day, 28-day, and 35-day cycles. We assume zero growth after day 20 (80 percent) in the longer cycles because the growth of incrementals is not infinite.

Day Number	Cycle					
	Daily	7-Day	14-Day	21-Day	28-Day	35-Day
1	4.0	4.0	4.0	4.0	4.0	4.0
2	4.0	0.2	0.2	0.2	0.2	0.2
3	4.0	0.4	0.4	0.4	0.4	0.4
4	4.0	0.6	0.6	0.6	0.6	0.6
5	4.0	0.8	0.8	0.8	0.8	0.8
6	4.0	1.0	1.0	1.0	1.0	1.0
7	4.0	1.2	1.2	1.2	1.2	1.2
8	4.0	4.0	1.4	1.4	1.4	1.4
9	4.0	0.2	1.6	1.6	1.6	1.6
10	4.0	0.4	1.8	1.8	1.8	1.8
11	4.0	0.6	2.0	2.0	2.0	2.0
12	4.0	0.8	2.2	2.2	2.2	2.2
13	4.0	1.0	2.4	2.4	2.4	2.4
14	4.0	1.2	2.6	2.6	2.6	2.6
15	4.0	4.0	4.0	2.8	2.8	2.8
16	4.0	0.2	0.2	3.0	3.0	3.0
17	4.0	0.4	0.4	3.2	3.2	3.2
18	4.0	0.6	0.6	3.4	3.4	3.4
19	4.0	0.8	0.8	3.6	3.6	3.6
20	4.0	1.0	1.0	3.8	3.8	3.8
21	4.0	1.2	1.2	3.8	3.8	3.8
	...	...	...	...	...	...
42-day total	168.0	49.2	66.6	91.6	94.6	107.2
Daily full	100%	29%	40%	55%	56%	64%
Best case	341%	100%	135%	186%	192%	218%

Table 44.1: Tape Usage for 4 TB Data, 5 Percent Change Daily

The worst case would be doing a full backup every day, or 168 TB of data written to tape. This would waste tape and time. Most environments have more data than could be backed up in full every day. Compared with the best case, daily full backups use 241 percent more tape capacity. This chart shows that the longer the cycle, the closer we get to that worst case.

The best case in this example is the 7-day cycle, or 49.2 TB of data written to tape. The jump to a 14-day cycle requires approximately a one-third increase in tape usage, with the same amount required for the jump to the 21-day cycle. Longer cycles have insignificant increases because of our assumption that incrementals never grow beyond 80 percent of the size of a full backup. If this example were our actual environment, it would be relatively efficient to have a 7-day or 14-day cycle or anything in between.

[Figure 44.1](#) graphs the accumulated tape capacity used with those cycles over 41 days, showing a running total for each strategy. The “daily” line shows a linear growth of tape use. The other cycles start out the same but branch off, each at its own cycle.

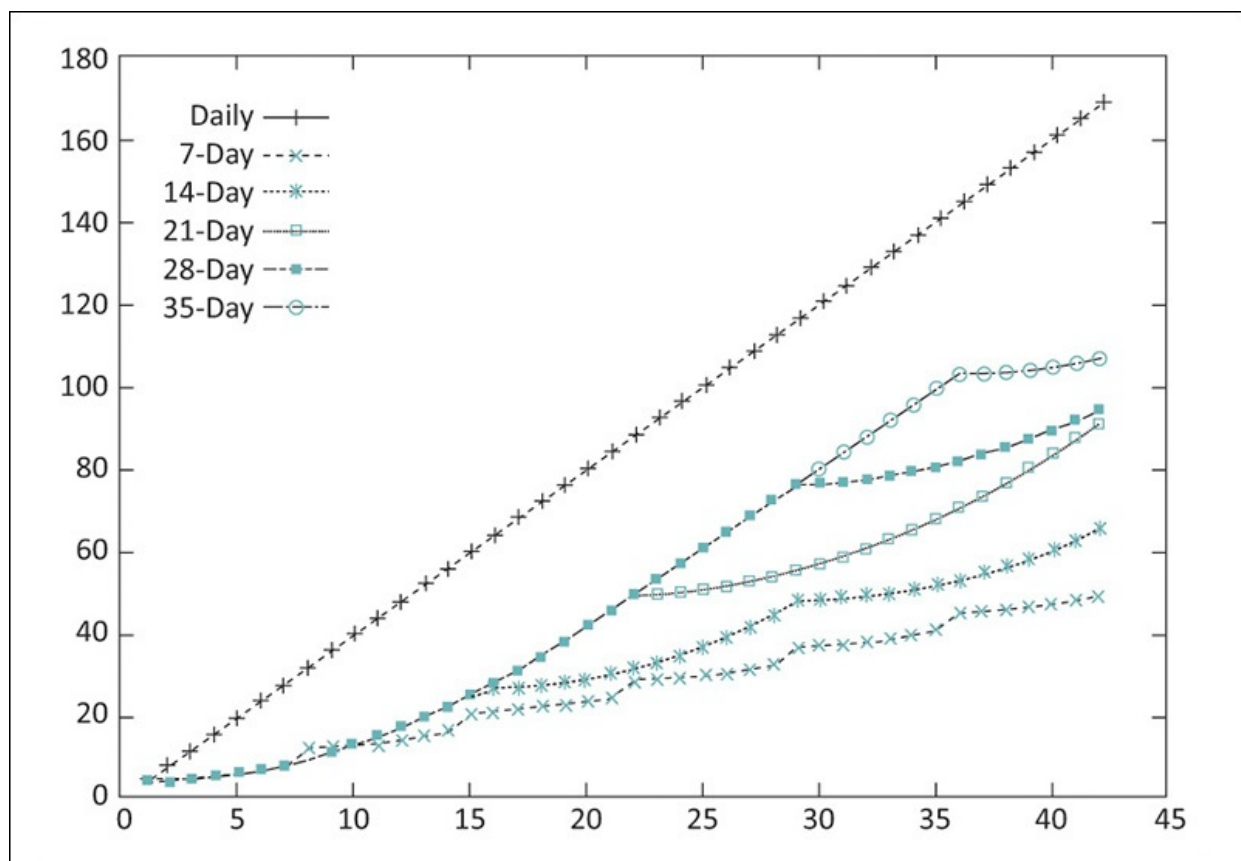


Figure 44.1: Accumulation of tape use by the cycles in [Table 44.1](#)

The first example illustrates the fundamentals in a simple scenario. A more complex and realistic model addresses the fact that most data access is a relatively low proportion of the data on disk. Our rule of thumb is that 80 percent of accesses are generally to the same 20 percent of data and that customers modify about half the data they access. Although we still can't tell in advance which data will change, we can predict that the first incremental backup will be

10 percent of the data size and that each subsequent increment will grow by 1 percent until the next full backup resets the cycle ([Table 44.2](#)).

Day Number	Cycle					
	Daily	7-Day	14-Day	21-Day	28-Day	35-Day
1	4.00	4.00	4.00	4.00	4.00	4.00
2	4.00	0.40	0.40	0.40	0.40	0.40
3	4.00	0.44	0.44	0.44	0.44	0.44
4	4.00	0.48	0.48	0.48	0.48	0.48
5	4.00	0.52	0.52	0.52	0.52	0.52
6	4.00	0.56	0.56	0.56	0.56	0.56
7	4.00	0.60	0.60	0.60	0.60	0.60
8	4.00	4.00	0.64	0.64	0.64	0.64
9	4.00	0.40	0.68	0.68	0.68	0.68
10	4.00	0.44	0.72	0.72	0.72	0.72
11	4.00	0.48	0.76	0.76	0.76	0.76
12	4.00	0.52	0.80	0.80	0.80	0.80
13	4.00	0.56	0.84	0.84	0.84	0.84
14	4.00	0.60	0.88	0.88	0.88	0.88
15	4.00	4.00	4.00	0.92	0.92	0.92
16	4.00	0.40	0.40	0.96	0.96	0.96
17	4.00	0.44	0.44	1.00	1.00	1.00
18	4.00	0.48	0.48	1.04	1.04	1.04
19	4.00	0.52	0.52	1.08	1.08	1.08
20	4.00	0.56	0.56	1.12	1.12	1.12
21	4.00	0.60	0.60	1.16	1.16	1.16
	...	...	...	...	...	...
42-day total	168.00	42.00	36.96	39.20	41.16	47.04
Daily full	100%	25%	22%	23%	25%	28%
Best case	455%	114%	100%	106%	111%	127%

Table 44.2: Tape Usage for 4 TB Data, 10 Percent Change on Day 1, 1 Percent Change Thereafter

In this case, the 14-day cycle is the best case, with the 21-day cycle a close second. The 7-day cycle, which had been the most efficient cycle in our previous

example, comes in third because it does too many costly full backups. Again, the worst case would be doing daily full backups. Compared with the best case, daily full backups use 355 percent more tape capacity. We can also observe that the 7-through 28-day cycles are all more similar to each other (between 6 percent and 15 percent of the best case), whereas in our previous example, they varied wildly.

When we graph the accumulations as before, we see how similar the cycles are. The graph in [Figure 44.2](#) shows this. (*Note:* This graph omits the daily full backups so as to expose greater detail for the other cycles.)

The optimal length of the cycle differs for every environment. So far, we have seen an example in which a 7-day cycle was the obvious best choice and another in which it was obviously not the best. Careful tuning is required to determine what is best for your environment. If you are starting from scratch and have no past data on which to base your decision, it is reasonable to start with a 14-day cycle and tune it from there. By reviewing utilization reports and doing a little math, you can determine whether a longer or shorter cycle would use less tape. Obviously, these decisions should be in compliance with the SLA and policy.

Recent development work has given the backup software the ability to tune itself. Although it can be difficult for a human (or even an SA) to keep track of the growing needs of hundreds of disk volumes, such monitoring is simple bookkeeping to a computer. Eventually, all commercial backup software will likely provide some kind of dynamic schedule.

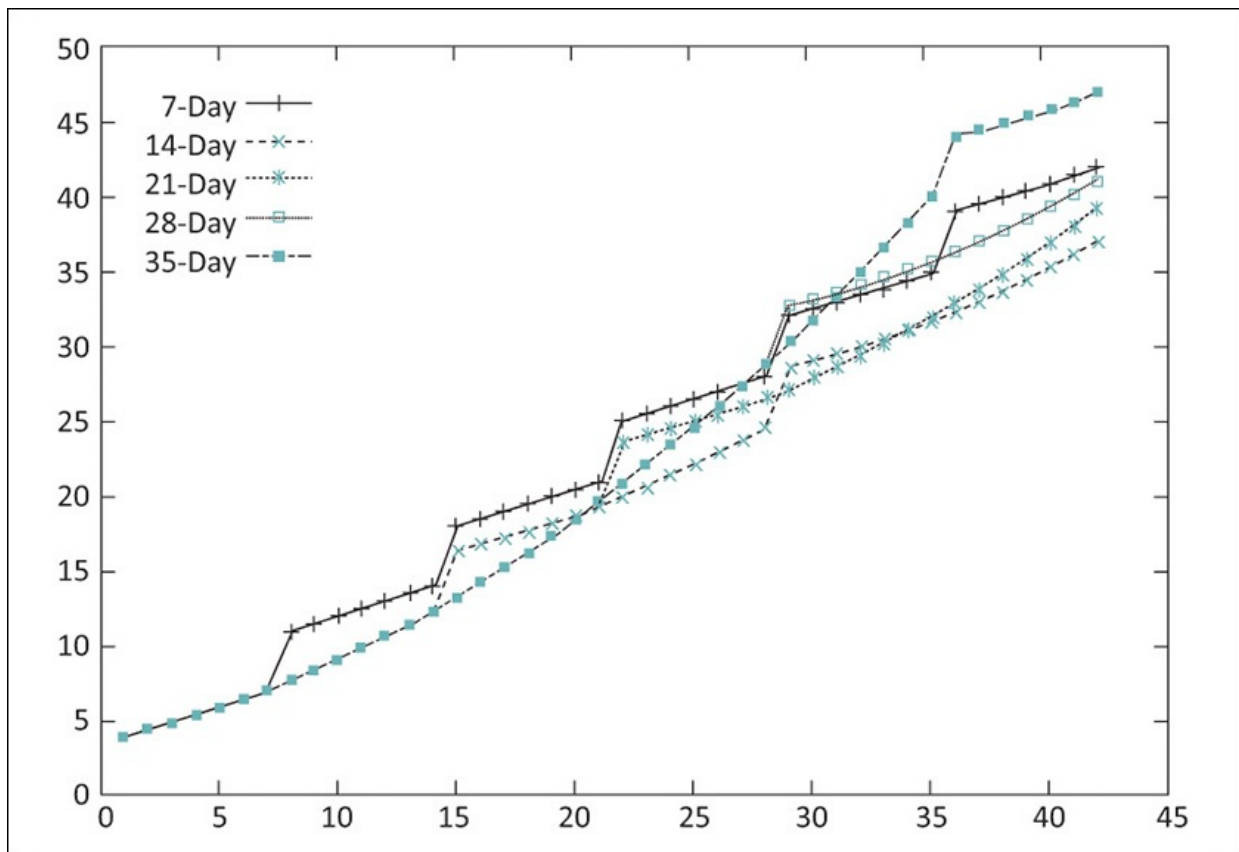


Figure 44.2: Accumulation of tape use by the cycles listed in [Table 44.2](#)

Case Study: The Bubble-Up Dynamic Backup Schedule

Dynamic schedules do not need to be complicated. Tom once created a simple dynamic schedule as follows. The SLA was that every partition on every server was to have a backup done each night, whether it was incremental or full, and full backups should be done every seven to ten days.

The list of partitions was sorted by backup date, putting the least recently backed-up partitions at the front of the list. The first partitions were designated for full backups that night on a separate set of tape drives. The remainder of the partitions received incremental backups.

As a result, any failed backups tended to bubble up toward the top of the list and be the first priority the next night. Backups typically failed because of a down host or, more likely, because the tape ran out of space. The software wasn't capable of continuing a backup onto a second tape. (This was in the days before affordable jukeboxes.)

The system could be tuned in two ways. If the partitions weren't

receiving full backups often enough, additional tape units were allocated to that function. If the incremental tapes were filling, more tape drives could be allocated for that function. In this case, the SAs had to watch whether the tapes containing the incremental backups were not simply getting dangerously full but also whether those backups were taking longer than their backup window permitted.

It is important to understand what your vendor means by incremental. Some systems have only a single level of incremental backup. Others have incremental backups that record all the changed files since the last backup of the same level—sometimes called *true incrementals*, or differentials. Take the time to test the system yourself to make sure that you understand how your vendor's backup system operates.

Another way to conserve tape is to perform two levels of incrementals, if your system supports it. For example, a full backup (level 0) is run on the first day of the month, followed by nightly incremental backups that capture any files modified since the original full backup: level 1 incrementals. The size of these incrementals grows to an excessive size by the middle of the month. On the fifteenth of the month, level 2 incrementals are begun; they record any file that has changed since the last level 1 backup. The mid-month incremental backup should return to being fairly small. This saves tape in the same way that doing incrementals instead of full backups saves tape.

There are two downsides to this schedule. First, it is much more complicated to track the various types of backups, although this is not a problem if the system is fully automated and maintains a good inventory. Second, it makes restores more difficult and error prone: You must now read the level 0, level 1, and level 2 tapes to make sure that all the files have been restored. This takes more time and, if the process is manual, the extra complication means that it is more prone to error. There is also a reliability factor: If there is a 1:1,000 chance that a tape is bad, having to rely on three tapes instead of two is an increased risk.

44.6 Time and Capacity Planning

Restores and backups are constrained by time. Restores need to happen within the time permitted by the SLA of the service. Most systems slow down considerably when backups are being performed. Some services must be shut down entirely during backups.

The speed of a backup is limited by the slowest of the following factors: read

performance of the disk, write performance of the backup medium, bandwidth, and latency of the network between the disk and the backup medium. Restore time is affected by the reverse of those factors. Tape units frequently write to the tape at a much slower speed than they read from it.

Many new SAs believe that vendors' statements about tape drive speeds and capacities bear some relationship to performance. They are gravely mistaken. The difference can be huge; we've seen a difference of up to 1,500 percent. Vendors continually tune and refine their backup algorithms for speed but often ignore restore speed; most of their customers don't think to demand fast restores, and the ones who need it are willing to pay extra for it.

44.6.1 Backup Speed

The slowest link in the chain will determine the speed at which the backup will happen. The backup process is also affected by mechanical issues. Most tape drives write at a high speed if they are being fed data as quickly as they can write (streaming mode), but downshift to a considerably slower speed if they are not being fed data quickly enough to keep up with the tape write speed. If the drive has no data to write, the drive must stop, reverse position, and wait until it has enough data to start writing again. Drive manufacturers call this the **shoe-shining effect**, as the read/write mechanism moves back and forth over the same spot on the tape. In addition to slowing tape performance, such repetition puts undue stress on the tape medium.

Therefore, if the server cannot provide data quickly enough, backup speed is dramatically reduced. For example, if network congestion is slowing the data's movement to the tape host, backups may be significantly slower than if congestion is not an issue. Many times, we've been asked to diagnose problems with slow backups, only to find that the network in place is slower than the maximum write speed of the tape unit.

When building a backup and restore system, you must take into account the speed of the various interconnections and ensure that the slowest link will not prevent you from meeting your time goals. It is common to use a dedicated network used exclusively by file servers to talk to their backup host. One of the first benefits that caused SANs to become popular was the ability to move backup traffic off the primary network.

To alleviate performance problems during backups, it is common to use a disk drive as a buffer. Previously, systems backed up data by copying it from one server to a tape unit on a centralized backup host. Now it is common to install a

lot of disks on the backup host. Servers back up their data to the disks on the backup host, often one file per disk volume per server. The backup host can then write the completed backup files at full tape speed. In many configurations, all of the servers write their backup data at night, and the backup host spends the day writing out the data to tape. This kind of arrangement is known as disk-to-disk-to-tape (D2D2T). It works better because local disk access is more deterministic than access across the network. Rather than having to engineer things so that the data can flow the entire path at the speed required for the tape drives, one must ensure only that such speed can be achieved from the local disk to the tape unit.

44.6.2 Restore Speed

Restores will also be as slow as the slowest link, but additional factors come into play. Finding a single file on a tape can take as long as a full disk restore itself. First, the system must skip other volumes stored on that tape. Then the system must read the particular volume to find the particular file to be restored. If the tape mechanism has no ability to fast-forward or skip to a particular data segment, this operation can be painfully slow.

Restoring an entire disk is extremely slow, too. The main issue affecting the restore speed is not the drive read speed, but rather the file system write speed. Writes are much less efficient than reads on almost every file system, and reconstructing a file system often leads to worst-case performance, particularly with journaled file systems. We've seen it take 5 to 15 times longer to restore the disk drive than to back it up. This is a very nasty surprise for most people.

If the server is able to receive data quickly enough that the tape drive can stay in streaming mode, the restore can happen at maximum speed. However, if the tape drive's data buffer fills up, the drive will slow down the tape speed or, possibly, stop the tape mechanism completely. Rewinding to the proper position and getting back up to speed is a huge delay.

The only way to know for sure whether your time goals have been met is to try it. Timing both a test backup and a test restore can validate your design. Over time, experience will help you determine what will work and what won't. However, it can be difficult to gain usable experience given that backup and restore systems tend to be reengineered every few years. You can instead rely on the experience of others, either a friendly sales engineer or a consultant who specializes in backup and restore systems.

44.6.3 High-Availability Databases

Some applications, such as databases, have specific requirements for ensuring that a backup is successful. A database manages its own storage space and optimizes it for particular kinds of access to its complex set of data tables. Because the layout and data access methods are usually opaque to the backup software, the database usually is written to tape as a single unit, or file. If the data in that file changes as it is being written, information may be lost or corrupted because the records for locating the data may be missing or incorrect on the backup tape. Databases often need to be shut down so that no transactions can occur during the backup to ensure consistency.

If the database has high-availability requirements, it is not acceptable to shut it down each night for backups. However, the risks associated with not doing a backup or performing the backup while the database is live are also unacceptable. Some vendors of backup software offer modules for performing backups of particular databases. Some of these substantially reduce the risks associated with a live database backup. However, it is generally safest to back up the data when the database is not running. This is usually achieved by having the database mirrored—using RAID 1 + 0, for example. The database can be stopped long enough to disconnect a mirror. The disconnected mirror disks are in a consistent state and unaffected by database transactions and can be safely written to tape. When the backup is finished, the mirror disks can be reconnected to the live database, and the mirror will be brought back up-to-date automatically.

In many environments a high-availability database is triply mirrored, with one set of disks actively mirroring the database and one set being detached and backed up.

44.7 Consumables Planning

Your backup and restore policy and schedule affect how quickly you will use consumables: tapes, tape cleaners, and so on. Estimating this rate of usage also requires doing math. Using our example policy again, incrementals can be recycled after being stored 6 months, and full backups, except those set aside as archives, can be recycled after 3 years.

Initially, there are no tapes to recycle. For the first 6 months, new tapes must be purchased for everything you do. Mathematically, you can project how many tapes will be needed by examining the schedule. Suppose that 6 days a week, 8 tapes will be used per day. That is 48 tapes a week, or 1,248 tapes for the first 6 months. Tapes cost about \$80 each, or about \$99,840 for the first 6 months. By comparison, a jukebox looks like a bargain.

Because the cost of tapes is continually decreasing, we recommend purchasing them in monthly or quarterly batches. A rule of thumb is to make the first batch a double batch to establish a cache of blank tapes in case future orders are delayed. Otherwise, the cost of tapes often decreases more quickly than the volume discounts you might receive by purchasing large batches up front.

In the second 6 months, you can recycle all the incrementals, so you need to purchase new tapes only for full backups. Let's assume that 9 tapes per week are full backups and that your incrementals are growing at a rate that requires you to purchase an additional 1 tape per week. Thus, you will need only 260 tapes in the second half of the year, costing \$18,200 if you assume that the cost per tape has dropped to \$70 by then. If these estimates are correct, recycling tapes will make your budget for tapes in the second 6 months only about 18 percent as much as what you paid in the first 6 months:

Tape cost (first year): \$118,040

The second and third years should also require about 260 new tapes per 6 months, or \$36,400 per year:

Tape cost (second and third years): \$36,400/year

Tape cost (first 3 years): \$190,840 total, or \$5,301/month average

After 3 years, you can recycle all tapes from the first year (1,508 tapes), except those marked as archival. If you do full backups every 14 days, the archival tapes should be all the full-backup tapes created in the first 2 weeks of any quarter, or 72 tapes per year ($9 \times 2 \times 4$). The remaining tapes total 1,436. Thus, you need to purchase only about 70 to 80 tapes per year. Assuming \$70 per tape, your budget for new tapes is reduced to \$5,000 to \$6,000 per year:

Tape cost (fourth and future years): \$6,000/year

Tape cost (first 4 years): \$196,840 total, or \$4,100/month average

Although the fourth year is the least expensive, it is likely the last year before you must upgrade to new technology with an incompatible medium. If the old system is still around serving legacy systems, the tapes available for recycling should be sufficient for your diminished needs.

Let's look at how things would be different if the policy kept full backups, except for archival copies, for only 1 year. The 4-year tape cost will be significantly lower. During the second and future years, you could recycle all tapes except the 72 tagged as archives. The second, third, and fourth years would cost less than \$6,000 each:

Modified policy 3-year cost: \$129,240 total, or \$3,590/month

average

Modified policy 4-year cost: \$134,840 total, or \$2,809/month
average

This single policy change didn't affect the first-year cost at all, but reduced both the 3-year and 4-year average costs by approximately 32 percent.

When setting the backup and restore policy, technical people commonly want backups that are retained forever, and financial people want a policy that saves as much money as possible. To strike a balance requires calculations based on your best predictions for the cost of consumables. It can be helpful to show people the cost models of what they have requested.

44.7.1 Tape Inventory

A pile of backup tapes with no index or inventory is only slightly more useful than no backups at all. The inventory is critical to being able to do restores in a timely manner. Large, automated backup systems maintain that inventory online. Often, special precautions must be taken in backing up the inventory, because the system that does the backups will want to update the inventory as it is being copied. You might consider printing a minimal tape index at the end of your nightly backups and keeping those printouts in a notebook. The inventory is a good candidate for storage on a system protected with RAID or similar technology.

Being able to restore files is dependent on the quality of your inventory. The better the inventory is, the more quickly the restore can be done. If there is no inventory, you will have to read tapes in reverse chronological order until the needed data is found. If the inventory lists only which partitions are on which tape, you will have to read each tape with data from that partition until the requested file is found. If the customer can remember the last time the file was modified, this can help the search, but it will still take a long time. Full restores will not be so impaired.

If the system stores a file-by-file inventory of each tape, the entire search process can be performed quickly in database queries; the exact tapes required will then be loaded. For example, if a couple of files on various directories plus a couple of entire directories need restoration, the software can deduce exactly which full and incremental backup tapes have the data that needs to be restored. All the required tapes will be loaded into the jukebox, and the software will perform all the restores, using all the available tape drives in the jukebox.

Keeping a file-by-file inventory requires a lot of disk space. Some commercial

products can strike a balance by maintaining the file-by-file listing for recent tapes and a simple partition listing for all others. The file-by-file listing can be reconstructed on demand if older files must be restored.

Software should be able to rebuild the inventory if it is lost or destroyed. In theory, you should be able to load the jukebox with the most recent tapes, click a button, and in hours or days, have the inventory be rebuilt. A site could do backups by night and inventory reconstruction by day.

A good inventory should also track how often a particular tape is reused. Most tape technologies become unreliable after being reused a certain number of times. You should expect backup software to be able to tell you when to destroy a tape.

In extreme situations, you may need to do a restore without access to the inventory, without access to a license server, and without the complete backup system working. Although a good inventory is critical to normal operations, make sure that the system you use doesn't prevent you from reading the data off a tape when you have nothing but the tape and a manual at your disposal. Look for all these features when selecting your backup and restore solution.

44.7.2 Backup Media and Off-Site Storage

Backup tapes must be stored somewhere safe. It doesn't make sense to spend large amounts of money and time on security systems to protect your data, then store your backup tapes in an unlocked room or cabinet. Your backups are the company's crown jewels. Finding secure and convenient storage space for them can be difficult in today's cramped office buildings. However, a set of well-locked cabinets or a large safe in an otherwise insecure room may be sufficient.

If your backups are to hedge against the risk of a natural disaster that would destroy your entire machine room, they should not be stored in the machine room itself or in a room that would be flooded if your datacenter were also the victim of a broken pipe.

Off-site storage of backup media is an even better idea. With this approach, a set of backup tapes or archival copies is kept at a safe distance from the computers that generated the backups. This need not be complicated or expensive.

The off-site storage facility can hold the backup tapes or copies of them. The choice of whether to store the actual tapes or just copies of them in such a facility is a tradeoff in convenience versus risk. Storing copies hedges against the risk that the tapes may be damaged or lost in transit. However, making the

additional copies may be laborious. Instead, the tapes themselves are usually stored off-site. Of course, this practice affects your ability to do restores in a timely manner. To minimize the inconvenience to your customers who require restores, you may choose to keep last month's full backups off-site, seeing that most restore requests are from the current month's backups. In turn, customers should understand that restore requests from tapes that are more than 30 days old but newer than 60 days may incur a delay.

When selecting an off-site storage facility, consider all the same issues that should be addressed with the on-site storage facilities: Is the space secure? Who has access to the facilities? Which policies, guarantees, or confidentiality contracts are in place? There are many ways to go about storing the backup media off-site, ranging from informal systems to large commercial digital warehouse services.

Informal Off-Site Storage

In some situations, an informal policy is sufficient. A fledgling start-up had a policy that every Wednesday, the head of its data processing department took home the backup tapes that were one week old. On Wednesday morning of the following week, she brought in the tapes that she had brought home the week before. One concern was that if she were in an auto accident, the tapes could be destroyed. The risk could be reduced by increasing the cycle to once a month. The risk could be completely eliminated by having her bring only copies of the tapes home. The other concern was the security of her house. The company provided a fireproof safe for her house to store the tapes in. The safe was secured so that it could not be stolen.

Many companies use an off-site records-storage service. This used to be a luxury service that only big financial companies could afford, but it has grown into a large market that serves all. These services provide pick-up and drop-off service, a four-or eight-hour turnaround time on requests to have particular tapes returned, and so on. Although their cost may sound prohibitive initially, they save a lot of hassle in the long run. They can also provide suggestions about typical policies that companies use for off-site storage. They even provide cute little locked boxes for you to put your tapes in when they are ready to be picked up.

Some security issues arise when an organization decides to store its tapes off-site. A third-party company paid to store the tapes must be bonded and insured.

Read the contract carefully to understand the limits of the company's liability. You'll find it disappointingly small compared with the value of the data on the tapes. Unfortunately, the horror stories you hear about these services are true. We've gotten back other people's tapes from storage facilities. In one case, we were not able to retrieve a tape that was sent out for storage and had to request the tape that contained the previous backup of that file. The author of the file was not happy.

It is important to keep good track of which tapes are sent out and to record every tape that is received back. Audit this tape movement and watch for mistakes; mistakes are an indicator of the storage service's overall quality. This can be your best defense. Imagine not being able to retrieve a critical tape and later finding out from your operators that occasionally the wrong tapes had come back in the past. Look for tapes that should have come back but didn't, tapes that shouldn't have come back but did, and tapes from the wrong company coming to you. If you are going to bring a complaint to the vendor, it is critical to have a written log of every mistake made in the past. Obviously, if these mistakes aren't one in a million, you need to change vendors.

Home-Grown Off-Site Storage

Companies with multiple buildings can provide their own off-site storage systems. One division of a company had its people spread out across two buildings that were 40 miles apart. The people exchanged tapes on a regular basis. Because they were all one division, they could even perform quick restores across their corporate network if waiting an hour or two for delivery was unreasonable.

Networked Off-Site Backups

A research center in New York had a lot of bandwidth between its two facilities and discovered that it was nearly unused at night. For several years, the facilities performed backups for each other's site over this WAN link. This had the benefit of making all tapes off-site. The bandwidth between the two facilities was large enough that restores could still be done in a reasonable amount of time. The management figured that it was an acceptable risk to assume that both buildings would not be destroyed simultaneously.

As network bandwidth becomes cheaper, the economics of performing

backups to other sites via a network becomes more reasonable. Commercial backup services have sprung up within Internet colocation facilities to service the needs of dot-coms. This kind of backup is easy because extremely high-speed networks can be installed within a colocation facility. As bandwidth becomes cheaper and over-the-network backup technology becomes more reliable, secure, and accepted, we foresee this kind of service becoming even more common.

Internet-Based Backup Systems

When Tom was 13 years old, he thought that an over-the-network backup service would be a fine idea but was discouraged when he realized that it would take hours to back up his 160 KB floppy disks over his 300-baud modem. When cable modems and xDSL service brought high-speed Internet access to homes, companies sprang up offering over-the-Net backup services. Even without high-speed access, these services are fine for backing up small things, such as a student's doctoral dissertation. Tom started backing up his personal computer to one of these services the moment he got cable modem access at his house to vindicate his age-13 inventiveness.

44.8 Restore-Process Issues

Important issues involving the restoration process require a lot of thought and planning. First, it is important to set expectations with customers. They should know what the backup policy is and how to request a file restore. Even a simple explanation such as this is sufficient:

Backups are performed only on data stored on servers (your PC's Z : drive, or Unix /home directory) every night between midnight and 8 AM. *We never do backups of your PC's local C : drive.* If you need a file recovered, go to [insert URL] for more information, or send email to "help" with the name of the server, the file's complete path, and which date you need the restore from. Barring problems, simple restores are done in 24 hours.

It is a good idea to include this information in any kind of new-user orientation documents or presentations and have it as a banner ad on your internal web portal. If your policy excludes certain machines from being backed up, it is particularly critical that people are aware of this fact.

You must think about the security implications of any restore request. Does

this person have the right to receive these files? Will the file permissions and ownership change as a result of the restore? Will the data be restored to the same place with the same permissions, or to a new place with possibly different security implications? Will it overwrite existing data?

There is a critical security issue here: Restore requests must be validated. Millions of dollars of security infrastructure can be defeated by a careless restore. Obviously, a restore of files to someone's directory on the server they originated from has few security implications. However, restoring a directory that is part of a project to someone's home directory may have security implications, especially if the person is not a member of the project team.

Although this kind of security attack may sound rare, it becomes a bigger risk as larger domains of control are outsourced. In a small company, it may be normal for a manager to request a restore of files from the directory of a staff member, and the SA can verify that the staff/manager relationship is valid because everyone knows everyone. However, in a 50,000-person company, how do you verify who is in which organization? Therefore, as a company grows larger, it becomes more critical that a well-defined procedure exists for validating restore requests.

It is key that multiple people be able to perform restores, not just the person who designed the system. Commonly, the engineer who designed the system invests time in automating the daily ritual of changing tapes so the process is as simple as possible. This allows a lower-paid clerk to do the task. However, designers often forget that it isn't wise to be the only person who knows how to do a restore. The restoration process should be well documented. The documentation should be kept online, and a printed version should be stashed on or near the backup hardware. The amount of effort taken to document and train people on a type of restore should be proportional to how often the restore is requested. Many people should be trained on the most common request: simple file restoration. This procedure should be easy to follow. A couple of people should be trained on how to restore an entire disk volume. This may require additional technical knowledge because it may involve replacing failed disks or knowing who has hardware training. Finally, a few senior SAs should be trained on how to restore a failed boot disk. This may be difficult to document because every server is a little different, but the key issue to document is how to do a restore on a system that is in some half-up state or how to do a restore when the machine with the tape inventory is down. All these documents should list the customer-support contact information for all vendors involved, as well as the maintenance contract numbers and passwords required to receive service.

44.9 Backup Automation

Not automating backups is dangerous and stupid. It is dangerous because the more you automate, the more you eliminate the chance of human error. Backups are boring, and if they aren't automated, they will not be reliably done. If they aren't done properly, it will be very embarrassing to have to face your CEO's question: "But why weren't there backups?"

Three aspects of the backup procedure can be automated: the commands, the schedule, and tape management and inventory. In the early days, there was no automation. Individual commands were typed by hand every time backups were done. Often, backups were started by the last shift before leaving for the night. The schedule was simple. There was little or no inventory except the labels on the tapes. The first step in automation was scripts that simply replicated those commands that were previously typed manually. However, deciding what was to be backed up when was still a human task, and very little inventory management was done. Soon, software implemented the scheduling algorithms that humans had done manually. Eventually, the algorithms were improved, offering dynamic schedules that went beyond the scope of what humans could reasonably do. Finally, the task of physically manipulating tapes was automated through the use of jukeboxes. Automated systems alerted a clerk to remove a certain set of tapes from a jukebox and to replace them with new tapes. With a well-maintained inventory, today's fully automated system can even automate the process of tracking which tapes are to be recycled and printing reports of tapes that are due for recycling.

Not all sites need such sophisticated automation, but all sites need to have at least the first two levels of automation in place. All this may seem obvious, but every site has one or two machines that have manual backups. Often, they are outside a firewall and unreachable from the central backup system. It is critical to introduce at least simple, rudimentary automation for these systems. If it is not automated, it will not happen.

We harshly call failure to automate backups stupid because manual backups are a waste of skills and time. With automation, the daily task of performing backups can be done by someone who has a lower skill level and therefore will be less expensive. Having a highly paid SA spend an hour a day changing tapes is a waste of money. Even if it takes a clerk twice as long to perform the function, it will be less expensive, because during those hours highly skilled SAs will be able to work on tasks that only they can accomplish. This is why corporate executives do not send their own faxes. (The person in the back row who is trying to point out that many executives can't figure out how to operate a

fax machine should please sit down.) It is better business practice to have executives do things that only they can do and to move all other tasks to lower-paid employees.

The only thing worse than no automation is bad automation. Bad automation automates many aspects of a task but does not alleviate the think work that must be done. Good automation doesn't simply do stuff for you; it reduces the brainwork you must do.

Backup System That Required Brainwork

Once upon a time, an SA had to reckon with a backup system that automated the major aspects of the task at hand and even printed pretty labels for the tape covers. However, the system failed to reduce the thinking aspect of the job. The software did a grand job of issuing the right backup commands at the right time, used a highly dynamic schedule to optimize tape use, and dealt with the security and political issues that were in place, which had the side effect of requiring about 10 small tape drives in the main datacenter and an additional 10 tape drives scattered in various labs around the building. (This was before tape jukeboxes were inexpensive.) However, every morning, the SA responsible for backups had to review 20 email messages, one for each tape unit, and decide whether that tape had enough space remaining to fit the next day's backups. To change a tape, a program was run, and much later the tape was ejected. As a result, it took an hour or two to complete the daily tape changes. The SA, being lazy, did not want to think through 20 tapes per day and make a decision for each one. He knew that eliminating this daily task would be like gaining 5 to 10 hours of additional time for other projects.

His solution was based on his realization that he could buy more tapes but not more time. He noticed that the tape drives connected to large servers needed to be changed often, whereas the tape drives scattered in labs required infrequent changes. Rather than starting off each morning with an hour of fretting over which tapes should be changed to optimize tape use, stop world hunger, and find a cure for cancer, he simply stopped changing tapes on Tuesdays and Thursdays. That gained him nearly 4 hours. If a new tape was started on Monday, Wednesday, and Friday, the risk of filling a tape by the next day was quite rare. He had not noticed this pattern previously because he hadn't spent the time to study the logs in that much detail. The time gained would be more valuable than the

occasional full tape/missed backup situation. Next, he determined that the tape drives in the labs filled up very rarely, and he eliminated the grand tour of the facilities to change all the lab tape drives, except for once a week. That gained about 3 hours. By restructuring how the tape changes were performed, he gained an additional business day of time each week.

The software being used was home-grown, and it would have been politically difficult to replace it before the author left the group, which he eventually did. Until then, this new process was a real time saver. In fact, the new procedure was so simple to explain to others that the SA was able to shift the daily process to a clerk, thus eliminating the task from his daily workload altogether. Victory!

Manual backups and home-grown backup software used to be very commonplace. Tracking new technologies, hardware, and OSs is costly. The more complicated backup requirements become, the more reason you have to purchase commercial software rather than trying to build your own system. With commercial products, the cost of development and support of the software is divided over the entire customer base.

Automated Backup Permits Delegation

A good friend of ours, Adam Moskowitz, was determined to build a backup system that was so automated that he could delegate the daily work to his company's administrative assistant, at least most of the time. Each day, the system sent email to her and Adam with the status of the previous night's backup. The message included instructions about which tapes to change or stated that there was a problem for Adam to fix. Adam automated more and more of the error situations so that over time, there was less and less for him to do. Soon months would pass without requiring Adam's intervention.

44.10 Centralization

Another fundamental design goal of modern backup systems is centralization. Backups should be centralized because they are expensive and important. Making the right investments can spread the cost of the backup and restore system over many systems.

Two major costs can be reduced through centralization. First, tape changes are costly because they are labor intensive. Second, the equipment itself is costly

because it involves precision mechanical parts spinning at high speeds. The tolerance for error is low.

Without centralization, a tape unit must be attached to every machine that needs to be backed up, and someone has to be paid to walk to every machine to change tapes. The result: paying for many expensive pieces of hardware and a large amount of physical labor.

Network-based backup systems let you attach large backup machinery to one or a few hosts that contact others to initiate backups. Network-based backups were adopted once networks were plentiful and reliable.

Jukeboxes are large devices that hold dozens, hundreds, or even thousands of tapes and contain robotic arms that shuttle tapes from their storage locations into one of multiple tape units. Jukeboxes are expensive, but their cost is amortized over all the systems for which they perform backups and vastly reduce labor costs. Tape drives, being mechanical, often break. The right software can detect a broken tape unit and simply use the remaining units in a jukebox to complete its tasks. Without network-based backup systems, you must either forgo backups on a system if its tape unit has died or install an extra tape unit on each system to ensure that there will always be one working tape unit. Certainly, a jukebox is less expensive than that! Jukeboxes also enable much of the sophisticated automation described elsewhere in this chapter.

44.11 Technology Changes

Free your mind of the notion that there is one good piece of software or hardware that you will use for all backups during the entirety of your career as an SA. Instead, start thinking in general terms and move with the changes in technology.

There is one constant in backup and restore systems: Disk technology and tape technology keep leapfrogging each other, never at the same pace. Embrace this inconsistent forward motion, rather than fight it. Some years, tape technology will zoom ahead, and you'll feel as though you can back up anything. Other years, you'll see disk technology forge ahead and wonder whether you will ever be able to back up all that you have. To understand why this happens, let's look at the historical growth patterns of each technology.

Disk size grows in small increments. Every couple of months, slightly larger drives are available. Overall disk capacity doubles every 15 to 18 months, and historically applications have been quick to use what is available. This means that about every other year, you will be backing up disks that contain twice as

much data.

Tape capacity over the years has grown in larger leaps but spread out over years rather than months. Consumers are less willing to upgrade tape-backup hardware, so the industry tends to provide forklift upgrades every two to three years. Resist the urge to upgrade your tape technology very often, because it simplifies life to not have to deal with many, many different tape formats. Most sites tend to use primarily what was the latest technology when their system was installed, but may also have a couple of legacy systems on the old platform. These legacy systems either haven't been upgraded yet or will be decommissioned soon, so upgrading the tape technology would be a waste of money. Sites then retain one or two tape drives for all previous tape technologies that they still have in their archives.

Keep One Nine-Track Tape Drive

Here's a way to amass favors owed to you. An SA at a large company discovered that he was the only person in the main complex with the ability to read those old reel-to-reel nine-track tapes. Although it was rare that anyone needed to read such a tape, anyone who did need such a tape read was desperate to do so. Outside companies will do tape conversions at a very high price, and the SA knew that people within the company would not want to pay such fees. He made sure that the tape drive was in a place that he wouldn't mind visitors using and attached it to a machine on which he could create guest accounts. Every year, he gained a couple of valuable favors, which he redeemed later. It also helped him gain a reputation as a nice guy, which can be more valuable in a large company than you would expect.

The unbalanced growth of disk and tape technologies affects how you can do backups. In the old days, most backup software was home-grown and simplistic. Tapes were small. Robotic tape libraries were too expensive for most companies, so splitting a backup across two tapes was either a manual process or impossible. Therefore, when QIC tapes could store 150 MB, SAs would split disks into partitions of 150 MB. Then 8-mm tapes with 2.5 GB capacity became popular because disks were commonly holding half to a full gigabyte of data. SAs thought that their problems with mismatched tape and disk sizes were over; data disks could be one huge partition. The upgrade to 8-mm tapes with 5 GB capacity came around the same time as disk capacity grew to 4 GB. However, when disks grew to their next leap (9 GB), the tape industry had not caught up.

This stalled the sale of these larger disks and was a boon to the commercial backup software industry, which could invest in making software to drive jukeboxes and handle the complicated task of splitting backups over multiple tapes. Next came DLT technology, which could hold 70 GB, again leapfrogging disk size. And history repeated itself when disks grew beyond 70 GB.

What can we learn from this? Change is constant. Trust your vendors to provide leaps in both areas, but don't be surprised when they get out of sync.

44.12 Summary

This chapter is about restores, for which backups are a necessary evil. This is a policy-driven issue. We are continually surprised to find sites that do not predicate their backup system on a sound policy.

There are three kinds of restore requests: accidental file deletion, recovery from disk failure, and archival. Each of these has different SLAs, expectations, and engineering requirements. More important, each serves distinctively different customer groups, which you may want to bill separately. The backup and restore policy is set based on these parameters. The policy should also state that the validity of backups must be tested with fire drills.

Once a backup and restore policy is in place, all decisions flow easily. From the policy, you can develop a backup schedule that is a specific list of which systems are backed up and when. One of the most difficult parts of determining this schedule is deciding how many days of incremental backups should be done before the next full backup. Current software does these calculations and can create a highly dynamic schedule. The policy helps you plan time, capacity, consumables, and other issues. Communicating the policy to the customers helps them understand the safety of their data, which systems are not backed up, and the procedure they should use if they need data restored. Making customers aware of which systems are not backed up is important as well.

A modern backup system must be automated to minimize human labor, human thought, human decisions, and human mistakes. In the old days, backups were a significant part of an SA's job, so it was reasonable that they consumed a significant amount of an SA's time. Now, however, an SA is burdened with many new responsibilities. Backups are a well-defined task that can, and should, be delegated to others. We delegate the creation of the software to commercial vendors. We delegate the daily operational aspects to clerks. We can even delegate simple restores to the customers who request them. We focus our time, instead, on designing the architecture, installing the system, and handling the

periodic scaling issues that come up. All this delegation leaves us more time for other tasks.

Modern backup systems are centralized. Doing backups over the network to a central, large backup device saves labor. The cost of large jukeboxes is amortized over the number of machines they serve.

Well-architected backup systems have excellent inventory systems. The system must have an excellent file inventory so that restores are done quickly and an excellent tape inventory so that tapes are recycled according to schedule.

Backup technology is changing all the time. As disk capacity grows, SAs must upgrade their ability to maintain backups. They must free their minds of the notion that any backup and restore system they install is their final solution. Instead, they must be prepared to scale this system continually and to replace it every three to five years.

Restores are one of the most important services you provide to your customers. The inability to restore critical data can bankrupt your company. The flawless execution of a restore can make you a hero to all.

Exercises

1. What is your restore policy? Do you have to work within corporate guidelines? If so, what are they? Or is your policy created locally?
2. [Tables 44.1](#) and [44.2](#) do not take into consideration the fact that tapes are usually changed every day, whether or not they are full. Why are the examples still statistically valid? Why might it be a bad idea to put multiple days' backups on the same tape?
3. Some vendors' incremental backups record only files changed since the last incremental. How does this affect how restores are done? What are the pros and cons of this technique?
4. Vendor terminology for "incrementals" varies. How would you construct a test to find out which variation a vendor implements? Perform this test on two different OSs.
5. What are the benefits and risks of using a backup system that can continue a dump onto a second tape if the first tape gets full?
6. The example in [Section 44.7](#) assumed that tapes could be recycled with no limit. Assume that a tape can be used 15 times before it must be discarded. Calculate how many tapes would have to be discarded each year for the first 4 years of the example.

7. [Section 44.7](#) didn't calculate how much would be spent on cleaning tapes. Assume that drives need to be cleaned every 30 days and that a cleaning tape can be used 15 times. Calculate how many cleaning tapes are required for the two examples in [Section 44.7](#).
8. [Section 44.7](#) assumed that the amount of data being backed up didn't change over the 4 years. This is not realistic. Reproduce the calculations in that section, based on the assumption that the amount of data stored doubles every 18 months.
9. [Section 44.7](#) specifies a particular tape-reuse regimen. What if new tapes were to be reused only 10 times? What if tapes kept forever were to be new tapes only?
10. Which aspects of your current backup and restore system should be automated better?
11. "A modern backup system must be automated to minimize human labor, human thought, human decisions, and human mistakes." What is the point being made here?

Chapter 45. Software Repositories

This chapter is about software repositories and the software packages that they serve. A software repository is a system that stores software packages and makes them available for installation by clients. The packages are usually ready-to-run, precompiled executable and related data files. You may already be familiar with repository systems such as Debian’s Advanced Package Tool (APT); Yellowdog Updater, Modified (Yum); and Microsoft Windows OneGet and Chocolatey.

Whether providing software to external customers, the global open source world, or an internal group of users, software repositories are a basic element of every computing infrastructure. In this chapter we discuss the many ways that repositories are used, the anatomy of such systems, and best practices for maintaining them.

A **software repository** stores and makes available a library of **software packages** that can be installed on client machines. A repository is usually accessible over a network by many hosts, usually called clients. The software is packaged so that it is ready to install. Clients access the repository through a package management system that tracks available packages, inventories what has been installed, and manages upgrades when newer versions are available.

A software package is a container. It is a single file that contains everything needed to install an application, patch, or library. You may be familiar with file formats such as ZIP, Unix `tar`, and `cpio`. Such files contain the contents of many smaller files plus metadata. The metadata is like a table of contents or index. It encodes information needed to unpack the individual files plus file ownership, permissions, and timestamps. Packages typically include the binary executables to be installed, any related data files, configuration data, and machine-interpretable instructions on how to install and remove the software.

Some software packages contain patches for other packages. A **patch** is a replacement for a file that has already been installed, intended to fix a bug or add a new feature. Patches are smaller than the original package and are often used to save bandwidth or disk space. For example, a bug in a device driver does not require the entire OS to be replaced, but rather may be fixed by installing a package with a single file that overwrites a specific file. The term “patch” comes from the physical world, where one might sew a small square of cloth over a hole in a piece of clothing, thus eliminating the need to replace the entire article of clothing.

With few exceptions, software repositories are accessed via a **pull** mechanism: Clients initiate the process of downloading software and installing it. Contrast this to a push mechanism, where a central server sends software updates to individual machines on some kind of schedule. The benefit of a pull system is that clients have more control: They can request software on their schedule. The client chooses which packages are installed. If the server cannot be reached, the client is responsible for retrying the download process. The downside of a pull system is that the server can become overloaded if too many clients decide to pull at the same time. Another downside is that the people who control the client machines may pull updates rarely, or never, meaning that security vulnerabilities are not fixed in a timely manner.

A push system controls the schedule centrally and therefore can avoid such situations. The downside of a push mechanism is that it lacks flexibility. It requires the server to know exactly which clients exist, and which clients require which software packages, and is responsible for retrying downloads to clients that are down. The centralized control makes it impossible to have anonymous or independent clients. For example, an OS vendor providing a repository does not know the location of every machine running its OS. A pull mechanism is more flexible because it decentralizes control.

45.1 Types of Repositories

Software repositories can be used for many different purposes. Most uses tend to be one of the following, a variation, or a hybrid:

- **Software release distribution:** An organization that produces software can use a repository to make that software available to others. This is a common practice for software vendors as well as teams and individuals who want to make software available. The repository may be accessible to only specific individuals, to everyone within a company or organization, or to the general public. Within a company, software development teams may release software by placing it in a repository that is accessible only within the enterprise. Sometimes repositories are accessible externally over the Internet, either freely to everyone or only to paying customers. Since different operating systems have different repository formats, a large software provider may maintain multiple repositories, one for each format, plus a separate repository for each version of each OS.
- **OS repository:** Operating system vendors set up repositories to distribute entire operating systems. Each subsystem of an operating system is packaged separately. This allows users to install only the parts of the OS

they require. Software patches and other updates can be delivered as packages, too. There are benefits to a unified system that uses the same package format and repository system for OS subsystems, applications, and updates. It is less work and fewer tools need to be developed.

- **Vendor mirror:** A vendor mirror is an exact copy of the vendor's OS repository. The vendor's repository is the **upstream repository**, and the organization's copy is the **local mirror**. A mirror is a complete copy, updated periodically. Contrast that to a cache that stores a local copy of packages on demand. Typically a local mirror is set up by a site with many hosts running the same OS. Dozens or hundreds of machines being installed at the same time can overload the organization's Internet connection. Using the local mirror is faster and less taxing on the local Internet connection as well as the vendor's repository. A side benefit is that using a local mirror hides internal details of the organization—how many clients there are, how often they poll, and so on.
- **Controlled mirror:** Sometimes it is useful to add a delay between when a vendor releases a package and when local machines have access to it. A controlled mirror is used to create this delay. The delay gives the enterprise time to test the packages and release them for use only after approval.

45.2 Benefits of Repositories

Software repositories benefit users in many ways. They can provide a wide variety of tools, saving the users from having to research and find tools themselves. Packages also embed knowledge that users may not have. For example, a package encodes which other packages must be installed (dependencies), and can configure the software as part of installation. Packages leverage the expertise of others as well. Compiling software from source is difficult and is beyond the skill set we should expect from users. The maintainer of a package is also the expert in the software, possibly packaging only releases that are known to be stable, and skipping intermediate releases. Users can't be expected to have such extensive knowledge. Packages also provide consistency. By using packages, users benefit from standardized file placement, documentation, and so on. Finally, packages save time. A package installs pre-tested binaries rather than building and testing the software on each machine.

Software repositories benefit system administrators, too. It is easier to support a package when the same bits are being installed on all machines. Software repositories enable us to control which software is installed on hosts. Security is improved because we can use a repository to distribute “known good” packages

and detect malicious tampering. Packages provide a focus point for testing software before they are released. Package systems that have the ability to “back out” packages permit us to easily revert when bad packages do get released.

There is value in curation. Maintaining a well-curated repository requires time and effort. It isn’t free, although the cost is amortized over all users. Users spend less time searching for software and figuring out how to compile it.

It is useful to create repositories for specific customer groups. Self-administered repositories are useful for teams to share releases internally or externally. Department-specific repositories are a useful way to distribute software specific to a particular department, possibly limited by license. SAs often create repositories for their own tools.

Repositories can be helpful in maintaining legal compliance. People are more likely to comply with legal restrictions when doing so is the easy thing to do. Including commercial software in the depot is as difficult as the software’s license is complex. If there is a site license, the software packages can be included in the repository like anything else. If the software automatically contacts a particular license server, which in turn makes a licensed/not licensed decision, the software can be made available to everyone because it will be useless to anyone not authorized.

If, however, software may be accessed only by particular customers, and the software doesn’t include a mechanism for verifying that the customer is authorized, it is the SA’s responsibility to make sure that the license is enforced. [Section 47.5](#) describes two low-effort techniques for tracking software licenses.

Why Not Just Install All Software Everywhere?

Rather than managing a repository and tracking which packages are and aren’t installed on a host, wouldn’t it be easier to simply install all the packages of an operating system? Wouldn’t that simplify our systems by eliminating the need for a repository management system?

There are many reasons why this does not work. Disk space on a host, while huge compared to the old days, is a finite resource. Some applications conflict with each other and cannot both be installed. Some software has a license that charges for the software if it is installed even if it isn’t used.

Some software packages install a service, such as a web or email server, that is activated by virtue of being installed. Thus, installing all packages would enable many rogue email servers, web servers, and so on.

Even if we could install all software packages on all hosts, we would still need repositories and the related management systems for OS installation and patches.

Lastly, there is a security risk in having installed software that is not used. A software package that is not installed cannot cause a security problem.

45.3 Package Management Systems

A software repository is made up of three parts: the server, the clients, and the packages. The server hosts the packages, making them available to the clients. The clients are the machines that use the service. The packages are the basic unit of exchange. Packages are stored on the server, downloaded to the client, and installed.

There are many package formats and repository formats. The formats of each type are tightly coupled to each other. For example, the Yum repository system serves only RPM packages. The APT system serves only DEB packages.

Similarly, most operating systems are tightly coupled to their repository system. Yum is used by SUSE Linux, RedHat, and many others. APT is used by Debian, Ubuntu, and many others. It is possible to mix and match, but doing so is rarely attempted because the package management system is so engrained in the way the OS works. For example, there are utilities for RedHat that will install a DEB package but can confuse the Yum system.

Some operating systems have a native package management system plus third-party ones. Microsoft Windows has its own software update mechanism and package format. There are also third-party package management systems that enhance or replace it, offering additional features or support options. Microsoft is standardizing on the OneGet client, which can talk to Chocolatey and other repositories. macOS has its own package system, and third-party products add an enterprise management layer on top of it. macOS also has various third-party package management systems for adding ports of commonly used open source software (Homebrew, Fink, MacPorts). These third-party systems try to avoid conflicts with the native system by placing their files outside of the normal system folders. While avoiding conflicts, this setup also means that it is possible to end up with two or even three copies of the same library.

45.4 Anatomy of a Package

Software packages contain all the files related to a software application, its installation, and its removal. This includes executables, installation-related scripts, data files, configuration files, documentation, and anything else that is part of the application.

45.4.1 Metadata and Scripts

The package includes metadata that describes where the files contained within the package are to be installed, which OS or OS variant the package may be installed on (release compatibility), and which other packages must be installed for this package to work (package dependencies). Some package systems have very sophisticated dependency mechanisms that can specify package versions that are required, minimum version numbers, optional dependencies, and dependencies that are required only when building the software from source.

Packages also supply code that is to be run at certain points. This code is called a script because it is written in a scripting language such as `bash`. As many as six different script opportunities exist: pre-install, post-install, pre-upgrade, post-upgrade, pre-removal, and post-removal. These scripts are all optional. If they are supplied with the package, they are run at the appropriate time.

Installation of a package involves running the pre-install script, copying the files into place, and running the post-install script. The post-installation script might install default configuration settings if no preexisting configuration is found or start the service that the software provides. For example, a package that installed the Apache HTTP web server might start the server. Pre-install scripts are rare. A modern Linux distribution may have thousands of packages, but few contain pre-install scripts.

Removal of the software involves running the pre-removal script, removing the files that were installed, and then running the post-removal script. Package systems track exactly which files were installed so they can be removed without requiring removal scripts. The pre-and post-removal scripts usually clean up caches and other files, but rarely remove any user-generated content.

Pre-and post-upgrade scripts generally do data conversion and sanity checking to assure that the upgrade will happen without failure.

45.4.2 Active Versus Dormant Installation

There are many different package systems, each with different features and benefits. Some have different philosophies. For example, some install services in

an active state, others in a dormant state. Debian Linux packages tend to install in an active state. For example, the Debian package of the Apache and Postfix start the services on installation. Red Hat takes the opposite view and installs software in a dormant state. The assumption is that the user will want to start the service himself or herself after properly configuring it.

45.4.3 Binary Packages

A package is generally made up of the output (known as **artifacts**) of the compilation process plus any additional files needed: configuration files, documentation, sample files, graphics, and so on.

Software packages usually do not include the source code for the executables being installed. Software written in compiled languages such as C, C++, and Go starts as source code that is compiled into executables (called **binaries** for historical reasons). Contrast this to interpreted languages such as Perl, Python, PHP, and Ruby, where the source code is the file that is run. There is no compilation step, so the source code is included in the package.

45.4.4 Library Packages

A **library package** does not store an application or other software, but instead includes libraries of code that other packages need. The Windows term is DLL; in Unix these are often called **shared libraries**. For example, many different packages may require the compression library called zlib. Other packages can list this library package as a dependency, meaning that it may be automatically installed if it is not already available. This reduces the size of the package and has the benefit that if a bug is fixed, only the zlib package must be updated, rather than dozens (or hundreds) of individual packages.

45.4.5 Super-Packages

Some packages include no actual files but simply a list of other packages to install. For example, a group of packages may be installed together so often that it becomes convenient to be able to install them all by simply specifying a single super-package. For example, a large software system may be divided into multiple packages; depending on which features are needed, users may then install some or all of the packages. There may be one particularly popular configuration that involves a particular subset of those packages.

The ability to tell people to install a single package rather than a list of individual packages enhances the convenience of the installation process. For

example, there may be a super-package for each department in an organization. Users do not need to know which individual packages are required for their workstation, but simply that they should install the super-package provided to (and possibly named after) their department. The list of packages in that super-package may change over time, with packages being added and removed as business needs change.

Super-packages are best named after the abstract concept or goal to be achieved. Documentation and configuration files can rely on this name not changing. Changes are made to the super-package itself. For example, a package named “foo” might always contain the latest release of a fictional software system called “foo.” However, there may be a super-package called “foo-stable” that installs an older, more stable release.

45.4.6 Source Packages

So far we have talked about packages containing compiled code. Source packages, in contrast, contain the instructions and files required to build packages. Packaging all the required information in one file ensures that this data can be easily archived, transported, or shared.

A source package includes the original source code, the configuration files, and patch files. The source code is exactly how it came from the author, usually still in a compressed TAR archive file. Often the source code must be modified, even if just a little bit, for it to compile on a particular OS. A source package includes **patch files** that are machine-readable instructions indicating which files to manipulate and how. They typically look like the output of the Unix `diff` command. The package build process uses a program that takes the original source files and modifies them as directed by the patch files. The result should be a set of files that are ready to be compiled on that OS. On Unix, the `patch` command does this work. The source package also includes compilation instructions, installation instructions, and often OS-specific example configurations and documentation that would not otherwise be provided by the original author.

The author of the original source code generally does not have the ability to produce packages in every format desired, and certainly does not have the time and energy to test the software on every operating system. A source package encapsulates the OS-specific knowledge that someone else may possess. Because the source package is decoupled from what the author originally releases, the process is democratized. One person (or organization) produces the source code. For each OS, a different volunteer takes responsibility for

maintaining the source package so that the software is available for that OS. When a new release of the source code becomes available, the source package can be updated with the newest source code and the new package can be produced quickly. Even if different patches are required, the source packager has an indication of what was required for previous releases and can use that as a guide for modifying the new release. If the source packager is unavailable, the new person has a record of the exact process used to build the package.

Contrast this with what typically happened before source packages existed. One person would get the source code and build the package. To do so, he or she might tweak a few files so that the code compiles and runs on a particular OS. These changes often were not recorded. The next software release required the source packager to recall those changes from memory—something that is clearly prone to error. If a new person took over, he or she had to start from scratch, possibly by guessing the way through the process or reverse-engineering what had been done previously. As a result, the process was not very repeatable or consistent.

Source packages are useful in other ways, too. If a package is not available for your operating system but a source package is available for a similar operating system, often the source package will build the package you need without any changes being necessary. If not, at least you have a working example to build on.

While most sites simply download binary packages that others have created, some sites download the source packages and use them to create the binary packages that will be deployed in their network. This is often done for performance reasons. For example, there are dozens of CPUs that fit into the 64-bit x86 architecture. The newer CPUs have features that can significantly boost performance. Compiling software to use these features results in executables that do not work on older CPUs. For this reason, packages meant for general distribution are built in a way that doesn't take advantage of these new CPU features. By using the source package to build a private version of the binary package, you can tune the settings so that it is compiled for the exact kind of CPU it will be used on. The full power of the CPU can then be realized.

Source packages also have some disadvantages. Their use shifts the responsibility of testing to you. Code compilation is slow, so installing packages from source is extremely time-consuming. Compilation is also error prone and can fail, leaving you to debug the problem yourself. Lastly, you could introduce errors, resulting in a package that does not install or—even worse—installs but malfunctions in unpredictable ways.

45.5 Anatomy of a Repository

A repository is not just where packages are stored, but a service that is provided. Like any other service, it should be planned, be monitored, and have disaster recovery plans. In the simplest sense, a repository is a directory (or directories) that contains packages (one file per package), plus indexes or other metadata to make transactions faster.

Repositories are usually accessed by HTTP or FTP. In fact, most repositories are simply a web server or FTP server that gives read-only access to static files. Repositories can also be accessed via network file systems such as NFS or CIFS. If a repository is to be accessed only by particular users, some kind of authentication and authorization credentials (username and password) are required. Such authentication should be done over a secure connection, such as HTTPS.

Source Code Control Is Not a Distribution System

One anti-pattern we see frequently is distributing files by checking them out from a source code control system such as Git or Mercurial. The client periodically does a pull, and the local files are updated with the latest versions of all files. There are many reasons why this approach should be avoided.

Technical reasons include the reality that such systems may not create reproducible results. Such systems usually cannot accurately reproduce file permission and ownership. File merge conflicts can result in problems that require human intervention to resolve. There is little control over which machines receive a release and when. You can potentially end up with many machines with different releases. It is difficult to support software when there is no specific release version number installed, merely a timestamp.

More importantly, this strategy violates the principle that the files that were tested should be the ones that are deployed. Each release should be a set of files that are tested and, if the tests pass, are released. Using a source code repository as a distribution system means that the files that were used in testing may not be exactly the same as the files placed into production. Discrepancies in the files are often due to little surprises such as directory permissions, servers being down, and scripting errors.

The process should be *source* → *test* → *production*. It should not be *source* → *test*, followed later by *source* → *production*. In theory the two

should be the same, but in practice they are not.

45.5.1 Security

Some repositories cryptographically sign all packages so that a person can verify that the package received is bit-for-bit the same as the package the builder created. This detects malicious actors, accidental data transmission errors, and other problems. Cryptographically signing packages means not having to trust the various servers, networks, caches, and mirrors that sit between the package builder and you. While we are concerned about bad actors, it is our experience that cryptographic signatures usually detect more mundane (and more frequent) issues such as a misconfigured mirror or corrupt cache. Signatures are more likely to prevent you from installing a package that was truncated due to a lack of disk space than from the actions of a malicious hacker—but we're glad it helps in either situation.

Tip: Transport Versus End-to-End Security

HTTPS is transport security. It secures the communication to the repository, assuring the repository is who it claims to be and validating the integrity of the data being transmitted. It also prevents eavesdropping, which is less important if the packages are widely available.

Cryptographically signing packages provides end-to-end security. It verifies who created the package and that no bits have changed since it was created.

While both are useful, end-to-end security is usually more important, unless privacy is required, because it is more comprehensive.

45.5.2 Universal Access

All the benefits of a software repository disappear if the repository is inaccessible. This commonly happens when the repository is behind a firewall and the client machine is a laptop that travels in and out of the organization's network.

Users are unhappy to find that their software updates kick in the moment their VPN connection is enabled. They enabled their VPN connection in preparation for doing work, but instead find their bandwidth hogged by software updates.

The best practice is to make software repositories accessible over the Internet and to do so securely. Secure in this context means that the connections are

authenticated and private. Authentication is important so that proprietary information is not leaked to those who should not have access. The connections should be private (encrypted) to prevent eavesdropping.

If you implement this strategy, the authentication and privacy should be as strong or stronger than that afforded by the VPN system in use.

Case Study: Google’s Secure Repository for macOS

Google’s macOS systems access a repository that is accessible over the Internet, and the authentication uses the same client certificate on the machine that is used for VPN access. Unifying on one certificate means only one certificate infrastructure is needed. Considering how difficult such systems are to manage correctly, this is a big win. Google puts its macOS software repositories on the cloud hosting platform so that machines can access it whether they are on the Google network or on a VPN. The software repository server is open sourced as the Simian Project.

45.5.3 Release Process

Often multiple repositories are used to coordinate the stages of the release process, such as those described in [Chapter 19, “Service Launch: Fundamentals,”](#) and [Chapter 20, “Service Launch: DevOps.”](#) For example, packages may initially be placed in a development repository. After testing, the package is promoted to a repository for beta software. Beta packages are eventually promoted to the production repository.

The policy should specify who may promote a package between stage repositories. For example, any developer may be allowed to put new packages in the development repository and promote them to beta after the developer’s own tests are complete. QA engineers might be able to promote packages from beta status to the production repository. There may be additional intermediate repositories for early adopters.

45.5.4 Multitiered Mirrors and Caches

Earlier we discussed mirroring repositories. As your network grows, there are benefits from using a more sophisticated mirroring strategy.

Companies with many different locations may choose to have multiple mirrors around the company. For example, there may be a primary mirror at the location with the biggest Internet connection or the one with the most users. Then there

may be additional mirrors, one per geographical region. These mirrors copy from the main mirror. Clients would point at the nearest mirror.

Whether the best choice is to use a multitiered mirror is a matter of determining if the needs outweigh the costs. The costs include not only the hardware, but also the logistics of delivering the hardware and maintaining it remotely. Delivering hardware internationally can be difficult, and performing repairs remotely even more so. These challenges must be balanced against the benefits of potentially faster service and the reduction in bandwidth use. On the one hand, faster service may not be as important if updates are done in the background. On the other hand, if the packages are required for OS installation, slow access time can cause the installation to fail, whereas faster access can enable better ways of managing machines. Mirroring may save bandwidth if the mirror is used a lot, but it may actually increase bandwidth needs if the mirroring process receives more new packages than are used. If mirroring uses 50 GB of bandwidth a day, on average, there would need to be a lot of installation activity required to justify the mirror.

It may not make sense to mirror the entire corpus to each region or building. For example, engineering tools may not be needed in offices that are set up purely for sales. It would be a waste to mirror engineering tools to these sites. One can put these tools in a separate repository that is not mirrored to such sites.

If the entire corpus does not need to be distributed to every mirror, a pull system can be used that caches packages on demand instead of trying to maintain a complete and perfect mirror. The tradeoff here is bandwidth and speed. A cache miss could result in a long wait to install a package. On the bright side, significant bandwidth savings can be achieved. An on-demand system can be as simple as using a caching web proxy configured to cache very large files; the defaults for such systems often are set to not cache large files.

There are many ways for a client to find the nearest mirror or cache:

- **Configuration management:** The configuration management system that controls the host can embody the knowledge of how to determine the best mirror and configure the client appropriately.
- **Windows AD Sites:** Every client is associated with an AD Site, which directs clients to the nearest mirror.
- **Existing systems:** Many companies have a system to direct users to the nearest replica of a service that can be leveraged for mirrors. These are usually based on IP anycast or DNS geolocation.
- **HTTP redirects:** If HTTP is used, the URL can point to a server that

returns an HTTP redirect to the best mirror based on the IP address of the request.

- **Manual configuration:** A simple manual process is to name the mirrors after the regions they serve and to configure the machine to talk to the appropriate mirror. If the machine is not mobile, this may be good enough. For example, naming the mirrors `mirror-eur`, `mirror-usa`, and `mirror-aus` makes it quite clear which one a machine should be using.

45.6 Managing a Repository

As a service, a repository needs a policy describing how it is intended to be used and operated. Here are some key points to decide and document:

- Who has access? Is anonymous and/or Internet access permitted?
- Who can add packages?
- Who can promote a package between stages?
- How is a user's identity authenticated?
- For which OSs are the packages suitable?
- Where should users report bugs?
- How do customers request packages to be added?
- How are software licenses managed? How is compliance verified?
- Is source code archived somewhere in case the original source is unattainable?

When you create a repository, it should be clear who has access and how those people are authenticated. Is access restricted by virtue of being behind the company firewall, or is the repository accessible from the Internet and thus open to all? When users connect, must they authenticate who they are, or is anonymous access permitted? Users with such privileged access (such as the ability to upload new packages) should be authenticated via means that are appropriate to the sensitivity of the packages themselves.

There needs to be a policy that describes who can contribute software packages to the depot. It can be dangerous to permit anyone to provide software that everyone will run. Someone could install a Trojan horse (a malicious program with the same name as some other program). Someone who is not aware of all the customer requirements might upgrade a tool and inadvertently create compatibility problems.

45.6.1 Repackaging Public Packages

Rather than using premade, public packages, one may choose to remake them or augment them.

The public package for a system may not be exactly as you need it. It can be useful to rebuild the package, slightly altering it in some way, such as giving it different compile-time options. Tom recently found himself doing this for a package that was built without SSL encryption. Enabling SSL was a build-time option, so it could not be enabled any other way than by rebuilding the software. The process of extracting the files from the source package, manipulating the build flags, and rebuilding the package was completely automated. This way, as the source package was updated for new releases, Tom's organization soon had its equivalent package. To avoid confusion, the new package was named differently; the company name was appended to the original name.

As described in [Section 45.4.6](#), sometimes it is useful to rebuild a package for a specific CPU variation. This should be done only if benchmarks verify the improvement.

It can be tempting to repackage software so that it includes configuration files or other data needed by your site. It's better to put those additional files in a separate package and set the primary package as a dependency so they are installed together.

It is useful to use packages to distribute site-specific configuration files or other data. However, you should not use packages as a replacement for a configuration-management (CM) system. CM systems offer flexibility and features that are not found in packaging systems. They scale better as complexity grows and provide a better development environment. CM systems are declarative, which is a better fit for configuring machines and services. Most of all, sites that attempt this eventually find they need a CM system to manage all their package installations, at which point they find themselves reinventing a CM system, often badly.

45.6.2 Repackaging Third-Party Software

Sometimes software vendors ignore the native package system of an operating system and provide software with their own installation mechanism. It can be convenient to repackage such software so that it can be distributed using your existing native infrastructure.

It can be tempting to reverse-engineer the vendor's install process and emulate it with the package system's functions. This is not a sound approach. Such a

process is a lot of work, as it requires updating for each new software release. It is also error-prone and the vendor is unlikely to support the resulting installation.

A better strategy is to create a package that includes the installation files and software as they came from the vendor, plus scripts to do an unattended installation. If an unattended installation mode is not documented, check with the vendor to see if one exists but simply hasn't been documented. There's a good chance a web search will turn up someone else who has found an official or unofficial method. If all else fails, file a feature request with the vendor. Software vendors that do not provide unattended installation are bad and should feel bad.

One system that facilitates this strategy is Chocolatey for Microsoft Windows systems. Chocolatey is a standardized package format for storing the installation files as they came from the vendor, along with instructions on how to activate the unattended install process.

45.6.3 Service and Support

Bug reporting becomes complex with third-party packages. If someone reports a bug with an open source package, should the system administrator fix the bug directly, pass the bug on to the open source project, or tell the person to report the bug directly to the author? If the system administrator is able to rebuild the package with a home-grown fix, what happens when the next release of the software arrives? Is the system administrator now forever responsible for fixing the bug and rebuilding the package?

Nontechnical users cannot be expected to interface directly with the original author. Technical users probably don't need the system administrator to assist them in addressing the bug. The policy should clearly state a level of service, defining which packages are critical and will receive direct involvement by the SAs. The other packages would be designated as receiving just the SAs' "best effort."

Service requirements for a repository are like those for any service: The service should be monitored for availability and capacity (network bandwidth, disk space), access should be logged, the service should be engineered to have the desired uptime and resiliency level, and disaster recovery (backups) should be maintained. Different uses of repositories have different requirements. For example, if a repository is simply a mirror of another repository, backups may not be required if re-mirroring would take less time than restoring from backups. However, even in this situation backups are required for the mechanism that

mirrors the upstream repository (software, configuration data, and so on). Hardware failures should be planned for, with hot or cold spares.

45.6.4 Repository as a Service

Providing a repository as a service enables others to enjoy the benefits of providing packages without all the effort. Each group of users that has a repository is called a **tenant**. As in an apartment building, each tenant has its own space and cannot (or should not be able to) interfere with the other tenants. The difference between providing repository service for yourself and providing it for others is substantial.

Your goal as a service provider should be to make the system as self-service as possible. Users should be able to add packages, promote packages, and do all other operations without your assistance. You might even make enrolling in the service be self-service.

Another way to help tenants be self-sufficient is to enable them to control access without needing your intervention. For example, certain designated people may have the ability to edit the list of who can upload new packages or promote packages.

The system must be constructed so that the actions of one tenant cannot interfere with the other tenants. One tenant should not be able to upload packages to other tenants' repositories, change their settings, and so on. If one tenant's repository is suddenly overloaded with requests and the system becomes flooded, all tenants may be negatively affected. This is called resource contention. Systems should be engineered so that resource contention triggered by one tenant does not affect the others. Such engineering is complex and may not be worthwhile for small systems. In this case the alternative is to be able to detect the situation and mitigate it, possibly through a rate-limiting mechanism or a policy of disabling such repositories in the event of an emergency.

Managing a software repository is a lot of work and requires a large amount of expertise that non-SAs do not possess. By providing this management service for others, we reduce the duplication of effort and, more importantly, prevent something from being done badly. As we discussed in [Section 45.5](#), a well-run service requires capacity planning (disk space, network bandwidth) and disaster-recovery planning (backups, redundancy).

To properly do capacity planning, one needs to know the user's future needs with respect to disk space, network bandwidth, and other resources. Monitoring should be in place to determine past and current resource use to help predict

these needs. Capacity planning can be as simple as asking each customer group, on a quarterly basis, what its predicted use will be for the next few quarters.

Tenants should be aware of the disaster-recovery planning done on their behalf. If one tenant receives an unexpected flood of traffic that affects other tenants, will that tenant be cut off or are other measures in place to deal with this scenario? Are full backups provided, or are tenants expected to maintain the original copies of packages and restore them to the repository if data is lost? If they are expected to retain all originals, will there be periodic fire drills to verify they are able to perform the restore?

45.7 Repository Client

The machine that uses a repository server is the client. This machine runs some kind of client software that interfaces with the server. In the simplest form, it knows how to connect to the server and download a package by name and version. More typically, the software is able to search the repository, select a particular version, manage dependencies, and so on. The client software usually tracks which software is presently installed and can manage upgrading old packages, seeking out dependencies and so on.

Most client software can point at many repositories. For example, the software might be configured to point at the OS vendor's main repository, a repository of additional software available from the vendor, a repository of home-grown packages maintained by the organization, or something else. The client usually has some prioritization strategy such that if the same package appears in multiple repositories, the higher-priority repository is used first. A typical strategy is to assign a higher priority to the most authoritative source.

Some client software can point at just one repository. In this case the repository server may actually be a frontend that draws from many other repositories. This setup is called **federation**. The client sees one seamless merged view. In some systems each client receives a different subset of available repositories. Federation is applied to provide the same flexibility as a client that can access multiple repositories, while simultaneously ensuring central control over which repositories are viewed. It is commonly seen in Microsoft Windows and other enterprise-focused products.

45.7.1 Version Management

Some clients have sophisticated features for managing package versions and overrides. For example, if a particular package should not be upgraded to a

newer version, the package can be “pinned” at its current version and newer versions will be ignored.

Sometimes there is a need to install multiple versions of the same package on one machine. This is usually not allowed. Each version installs files into the same location, with the same filenames, which means it is not possible to have more than one version installed at any time. Most packages are built to assume they are installed in a particular location, and one cannot simply move the files and expect them to work.

The first strategy for dealing with this incompatibility is to simply ask the people involved if they can agree on a common version. The parties involved may be unaware of the conflict.

If that does not work, try to isolate the two use cases. Use two machines instead of one, where each machine uses a different version. Docker Containers can be used to isolate two different services on one machine, thereby eliminating the filename conflicts.

Some package systems eliminate this kind of conflict by having each package be self-contained. Two systems that use this strategy are the PC-BSD and Nix package managers. Rather than smaller packages that require additional (dependent) packages to be installed concurrently, each package contains all the files from all the dependencies. This has the benefit that conflicts cannot happen, and it also permits two versions of the same package to be installed on the same machine at the same time.

Self-contained packages are larger, resulting in longer download times and larger disk space requirements. However, bandwidth and disk space are now extremely inexpensive and plentiful compared to when the first generation of package systems were created.

Another tradeoff when using self-contained packages is the fact that security updates become more difficult. For example, suppose a security hole is found in the OpenSSL package, which is used by many products. Every self-contained package that embeds OpenSSL would need to be rebuilt, and SAs would have to install the upgraded packages on all affected hosts. Traditional systems that share one installation of OpenSSL could simply upgrade the OpenSSL package, and all services that depend on OpenSSL would then use the new version the next time they restart. Restarting services is a lot easier than waiting for, and installing, new packages on every affected host. Package management systems such as Nix and PC-BSD have automated build and test processes that assure fast delivery of new releases. Even so, installing them is still a burden for

the SA.

The last strategy can be labor intensive but has other benefits. If all else fails, the SA can rebuild each package from scratch in such a way that each version installs in a different location. This requires manual work for each new revision, but the additional effort is minimal if you are already maintaining a custom version of the software for other reasons.

Rebuild Package Versions with Unique Package Names

Once Tom needed to install HAProxy versions 1.5 and 1.6 on the same machine. The original package was called `haproxy` and was available in versions 1.5 and 1.6, which both installed in the same place. Tom created packages called `haproxy1-5` and `haproxy1-6`, which were installed in `opthaproxy1-5` and `opthaproxy1-6`, respectively. The version numbers on those packages started out as 1.0. If a package was rebuilt, the new package would still be called `haproxy1-6`, but it would be version 1.1. With this scheme, many versions of HAProxy could be present on the same machine.

45.7.2 Tracking Conflicts

In addition to tracking dependencies, some systems track conflicts. They may, for example, indicate that installing two particular packages at the same time causes one of them to be broken or unusable. Note that it is not considered a conflict if two packages have similar functionality or perform the same tasks.

Some clients perform **package staging**. This is a process that attempts to prevent installation failures by delaying the actual installation of files until all preparatory steps are complete. For example, all packages and dependent packages are downloaded and their digital signatures are verified before the installation starts. Before staging was implemented, systems would download one dependency and install it, then download the next and install it. If one particular package failed to download, packages would be partially installed and the host would be left in an inconsistent state.

45.8 Build Environment

Packages do not come just from vendors. You may find yourself building packages or working with developers who need assistance in building packages.

The first step is to build the software manually to ensure that you fully

understand the complete process. Then use the packaging system tools to repeat your steps and create the package. Document the steps required to build the package, and make sure the process can be repeated on another machine. This will harden the process.

Ultimately the package building process should be automated, and any configuration files, scripts, or other automation should be kept in a source code control system, preferably the same one as the source code itself. Software is not a static, unchanging thing. There will likely be updates, and the package will probably need to be rebuilt at some point. There are countless horror stories of companies that found themselves unable to build software packages after a key developer left the company. Making sure the entire process is automated and kept in the same source code system means that changes to the process will be tracked the same way as changes to the software.

45.8.1 Continuous Integration

The next level of automation is called continuous integration (CI). When CI is in place, a system notices changes being checked into the source code control system and builds the package, installs it, and runs a series of quality assurance tests. If the process is successful, this package is considered a release candidate. By detecting failures as soon as they happen, it becomes easier to fix problems. Imagine building the package just once a month and, on failure, needing to search through a month's worth of changes to find the one that caused the problem.

It is important that the build process is always done in the same environment. Software compilation and packaging can be affected by shell variable settings, directory and file permissions, compiler versions, and other factors. If all of these items are not reproduced the same way for each build, the results can be inconsistent. In one case, we experienced a file permission issue that made the build process think a particular module was unavailable. Rather than complain, it silently substituted a different module. In another situation, Tom was asked to debug a software package that would build correctly only when Tom logged in as his co-worker on his co-worker's machine. The package would not build if someone else logged into the same machine, or if the co-worker attempted to build the software on another machine. It turned out that the build process was dependent on unusual tools that the co-worker had installed on his workstation and on some unusual shell variables set in the co-worker's `.bashrc` file.

One of the advantages of CI is that the automated system starts each build in a clean environment, thereby improving repeatability. Developers need to be able

to build a package outside of the CI system, but those packages should not be distributed to users or outside the developer test environment. Official packages should come from the CI system.

45.8.2 Hermetic Build

A **hermetic build** is a build process that has zero untracked dependencies. A hermetic build system should be able to reproduce the exact same package, bit-for-bit (except timestamps), for any past software release. Achieving this goal requires the ability to re-create the build environment including the exact compiler version, OS version, shared libraries, and so on.

Imagine a customer demands a fix for a high-priority bug, but is using a software release that is two years old and cannot upgrade to the latest release. It would be risky to build the software from source as it existed two years ago with the current compiler. A new compiler release might introduce new bugs or just unexpected performance changes. If builds were hermetic, the system would know how to extract the exact files from the source code repository and how to create an environment with the same compiler, same OS release, and same set of patches, libraries, and other tools. A new package could then be built with confidence that the only change was the bug's fix. This may sound extreme, but some industries have such requirements, such as medical devices, telecommunications, NASA, and other systems that are long-lived and may have life-and-death safety consequences.

The Linux `mock` system makes it easy to create a near-hermetic environment for building packages. It makes building software convenient, and you get a high level of repeatability for free.

45.9 Repository Examples

In this section we describe three examples where software repositories were used as part of a larger system. A staged software repository uses different depots for different stages of the SDLC. An OS mirror depot conserves bandwidth and protects privacy. A controlled OS mirror uses depots to control when hosts get access to new packages.

45.9.1 Staged Software Repository

A software repository can be used as a way to distribute locally produced software. Such a system may be used to distribute software internally or externally. This is generally done by providing a few repositories, one for each

stage of the software development and testing process.

The **unstable repository** accepts packages from developers. They can then download the package to verify that their process for creating and uploading the package was a success. An automated CI system may upload all successfully built and tested packages to this repository.

Occasionally a package in the unstable repository will be selected as a release candidate. This package may receive additional testing. It is then promoted to the **beta repository**. Beta releases are candidates for general releases.

Only if no bugs are found does a package get promoted from the beta repository to the **general repository**. This promotion process includes saving a copy of the package being replaced, so that it will be easy to revert to the previous release if problems are discovered with the new release. If the repository system permits storing multiple versions of the same package, this may be as easy as tagging the old package as the one to revert back to. If the repository system stores only one version of a package at a time, alternative means are required: One might have an additional repository called “old general” that stores the previous “known good” package so that it is not lost in the promotion process.

Moving packages among four repositories might seem like a lot of bureaucracy just to push software out to users. Nevertheless, this extra effort is worthwhile because it prevents bad releases from reaching a wide audience. It also permits the organization to enforce quality control handoffs. For example, while all developers may be able to submit packages to the unstable repository, only designated quality assurance engineers may be able to promote packages after that.

A repository system such as this is a service and—like any other service—should be monitored. It should be monitored for disk space, and a lack of disk space should generate an alert. The amount of bandwidth used should be monitored so that future bandwidth needs can be predicted and accommodated ahead of time, not just when the service is found to be overloaded and slow. Access should be logged so that there is a record of who uploaded and promoted packages, thus affording accountability. There should also be a log of who or which machines downloaded packages. While this information shouldn’t be needed, it will come in handy when a bad release gets promoted and you find yourself in the unpleasant position of needing to notify the people who downloaded it. It is better to know who to warn than not.

Case Study: The Non-repository Repository

Tom observed this situation when he was at Bell Labs, but it is a story that seems to have happened at every company at one time or another.

A co-worker was good at maintaining a library of software. It contained both locally developed software plus software found on the Internet. The library was in his home directory, and everyone happily accessed it directly just by putting *homehelpfulperson/bin* in their PATH. But when this person left the company, everyone lost access to the software. To make matters worse, his home directory was hardcoded into scripts and configuration files all over the company. Systems broke, software had to be recompiled, and the lesson was learned: A person's home directory is not a viable software repository.

45.9.2 OS Mirror

A common application of software repositories is the OS mirror. For example, a site with many machines using the same operating system may make a mirror, or copy, of the vendor's repository and access it instead of directly accessing the vendor's server. This OS mirror is accessed for everything from installing a machine from scratch to downloading new applications, updates, and patches.

This approach can greatly speed up OS installation time. Access to a local server is faster than accessing a vendor's service over the Internet.

It also can save bandwidth. Internet access costs money. Hundreds or thousands of machines making the same requests for packages is a waste of bandwidth. In some countries Internet access is oppressively expensive and slow, so a mirror becomes a necessity.

A mirror also permits one to shift Internet utilization to off-peak hours. The process of updating the mirror can be a bandwidth hog that is better run when Internet use is lower.

45.9.3 Controlled OS Mirror

Another application of a software repository is the controlled OS mirror. A controlled mirror is similar to the previously discussed mirror but is delayed. That is, users do not see the new packages until they have been certified for use within an organization. This approach offers the benefits of a regular OS mirror but reduces the risk of failures and other surprises.

Bad releases happen. As an example, suppose a new release of a popular

application ships and every machine in the company installs it. While this release was sufficiently tested by the vendor, it has a bug that affects only your company. Now the application crashes, freezes, or possibly even refuses to start. Your entire company is unable to use the application. This could have been prevented with a controlled mirror.

A typical way to implement a controlled mirror is to create a few repositories. The first is the direct repository, which mirrors the vendor's repository directly. Only a few, specific test machines point at this repository. Packages are tested and, if approved, are copied into the main repository. This is the controlled mirror that all other machines point to.

There are many variations on this theme. First, there may be so many packages that testing them all would be impossible, unless the testing is automated. Also, not every package might be in use in your organization, so testing them all would be a waste of time. For that reason some packages may be automatically approved. One usually creates a whitelist, or list of packages that are automatically approved; all others require testing and approval. Alternatively, one can create a blacklist, a list of packages that require testing and approval; all others are automatically approved.

There may be additional repositories between the direct repository and the controlled one. Packages in the direct repository that are manually or automatically approved may first go to a repository that only early adopters point their machines at. These people are less risk averse and will report problems that may have been missed in the official testing. After a package has been in this repository for a certain number of days with no problems reported, the package is then automatically copied into the main repository.

Another variation is to randomly select which machines receive the new packages first. This is slightly more risky but requires less labor than manually tracking early adopters.

Note, however, that critical servers should not automatically receive new packages. For example, an email server should not just magically upgrade itself, but should be upgraded only after careful planning and testing.

45.10 Summary

A software repository used to make software packages available to many hosts.

In the old days, disk space on individual Unix clients was minimal, often less than 100 MB. Therefore clients would use NFS to mount *usr/local/bin* from a central file server. Updates were done “live,” and reverting old releases was

difficult or impossible. All machines had access to all software.

Today local disk space is plentiful, so modern systems install software locally. Installing software locally enables each machine to be custom configured with the software it needs. The software is downloaded from repositories that store prepackaged software.

Software and related files are combined into a package—a single file that is easier to transport than the individual files. They are stored in a software repository and served by various protocols, commonly HTTP/HTTPS or FTP. Clients contact the repositories and query them to determine which packages are available and which updates are available.

Repositories can contain original software or be a copy, or mirror, of another repository. Having a local mirror can improve performance and save bandwidth.

Repositories are a service and thus require planning and maintenance. They require capacity planning, monitoring, and disaster recovery just like any other service.

Exercises

1. What is a software repository?
2. What is a software package?
3. What are some of the differences between the various software package and repository systems available?
4. What are the benefits of using a software repository?
5. What are the parts of a software repository, package, and client?
6. If HTTPS provides end-to-end encryption and assures that no bits are changed during transport, why would digitally signed packages be needed as well?
7. Which access controls do software repositories need and why are they needed?
8. When providing a software repository service on behalf of others, what can be done to make it self-service?
9. Describe the software repository used in your environment, its benefits, and its deficiencies. If you do not have a repository, describe what is used instead, and list the pros and cons of adopting one for your environment.
10. What are the policies for your software repository? If you do not have any policies or do not have a repository, develop a set of policies for a

repository that you would use at your site. Justify the policy decisions made in your repository policy.

11. Compare your software repository to one of the examples described in this chapter. How is it better or worse?
12. A good software repository management system might also track usage. How might usage statistics be used to better manage the repository?
13. Why should a software release process be *source* → *test* → *production*, and not *source* → *test* followed later by *source* → *production*? What is the difference between these two processes?
14. If you are at a small site that does not require a complicated software repository, describe what kind of simple software depot you do have. At what point will your organization have grown enough to require a more complicated depot? What might that look like? How will you convert to this new system?

Chapter 46. Web Services

This chapter is about managing web-based services. Over the years applications have shifted from being installable software to web-based services that people access. Running a web server is more than just installing Apache or IIS and providing content.

The web is based on open standards, which are developed by an international committee, not a single corporation. You can use them without paying royalty or licensing fees. Web standards are defined by the World Wide Web Consortium (W3C), and the underlying Internet protocols are defined by the Internet Engineering Task Force (IETF).

The benefit of web-based applications is that one browser can access many web applications. The web browser is a **universal client**.

Web applications and small web servers are also present in firmware in many devices, such as small routers and switches, smart drive arrays, and network components.

This chapter covers the basics of how web servers work and how to manage small to medium-size web sites. The design and operation of large-scale web-based services are the topics of Volume 2 of this book series.

Open Standards Make the Web a Success

Imagine if instead of using open standards, one company set the standards and charged royalties to any web site or browser that used them. The web would never have been as successful. In fact, one attempt at just such a thing happened as the web was becoming popular. Network Notes was a product created out of a collaboration among AT&T, Novell, and Lotus, which at the time were the biggest players in network services, enterprise software, and application software, respectively. Network Notes was similar to the web but one had to use the Novell network protocol, Lotus server software, and a dedicated connection to AT&T's network. Network Notes users could communicate only with other Network Notes users.

People weren't willing to pay money for the software, because there were no sites to visit; nobody was willing to build a site, because nobody had the software to access it. The vendors weren't willing to give the software away for free, because they had to recoup their development costs.

Even if the service had become popular, innovation would have been stifled in other ways. Without the ability for vendors to create their own client software, nobody would be able to experiment with building browsers for other devices. Cellphone-based browsing might never have happened.

This example is relevant because even with the success of the Internet, companies continually—and bullheadedly—repeat these mistakes. There are continual attempts to extend web browsers in ways that lock customers in; rather, the creators of these schemes are locking potential customers out. Smartphone apps often try to reinvent the Internet as a closed ecosystem. There are constant attempts to change the HTML standard to permit proprietary DRM standards and other technologies that begin the slide down the slippery slope toward a closed web, controlled by a few. We must always be vigilant.

46.1 Simple Web Servers

Different types of content require different serving infrastructures. A single web server serving an unchanging document has different needs from a page serving dynamic pages. There are four basic web server types:

- **Static web server:** A static web server is one that serves only documents that don't change or that change only rarely. The documents are read directly from disk and are not altered in the process of being presented by the web server.
- **CGI servers:** CGI servers generate pages dynamically rather than simply loading them from disk. Dynamic page generation uses more CPU and memory than reading from disk.
- **Database-driven web sites:** Database-driven web sites generate each web page from information in a database. Each page is generated by reading information from a database and filling that information into a template.
- **Multimedia servers:** A multimedia server is primarily a web server that has content that includes media files, such as video or audio. Multimedia servers have high performance requirements for CPU, memory, storage, and network.

LAMP and Other Industry Terms

Certain technology combinations, or platforms, are common enough that

they have been named. These platforms usually include an OS, a web server, a database, and the programming language used for dynamic content. The most common combination is **LAMP**: Linux, Apache, MySQL, and Perl. LAMP can also stand for Linux, Apache, MySQL, and PHP; and for Linux, Apache, MySQL, and Python.

The benefit of naming a particular platform is that confusion is reduced when everyone can use one word to mean the same thing.

46.2 Multiple Web Servers on One Host

Small sites may want to start with a single web server that is used for all applications, because the amount of traffic that each application receives does not justify the cost and overhead of additional machines. Large sites that load-balance web services across multiple machines often want all the machines to be identical clones of each other, rather than having N machines dedicated to one application and M machines dedicated to another. Identical clones are easier to manage. There are many ways to host multiple web services on the same machine.

HTTP-based services usually listen for requests on TCP ports 80 (unencrypted) and 443 (encrypted via SSL, often called HTTPS). Multiple services cannot share a port. That is, only one process can be listening to port 80 at a time. The easiest way to resolve this is to configure multiple IP addresses on a machine. A less attractive option is to use different ports for each server, with URLs that list the port, such as `http://myservice:8000`.

Configuring multiple IP addresses on a machine does not scale well. This scheme is complex and not well supported in some OSs. Also, IPv4 addresses are scarce. URLs containing the port information look unprofessional.

46.2.1 Scalable Techniques

There are two techniques for permitting many services to share the same port on the same IP address. Both techniques provide a scalable solution with a professional look.

The first technique is called **virtual hosting**. Originally one machine could host only one domain. Virtual hosting permits one system—for example, the Apache HTTP server—to listen for incoming HTTP and HTTPS connections, but serve different web pages depending on which domain the request is for. This required a change to the HTTP protocol, but it is supported by all web

browsers.

The second technique is to use one process as a **dispatcher** for many individual services. A single process listens for connections and forwards them to the proper service. For example, Nginx (pronounced “EngineX”) is often used to listen for incoming HTTP and HTTPS connections, determine which domain they are intended for, and forward the request as appropriate. The individual services listen on different ports (8000, 8001, and so on), but this is hidden from the users. This has another benefit in that Nginx can decrypt SSL connections so that the individual services have to process only the more simple HTTP protocol. It also isolates all cryptographic issues to one place, where they can be managed separately.

46.2.2 HTTPS

Serving multiple domains on the same IP address is more complex for HTTPS (encrypted) connections due to a Catch-22 in how HTTPS works. Each domain has its own SSL certificate. The software needs to know which SSL certificate to use to decrypt the request, but the intended domain is in the encrypted portion of the request.

An early solution to this Catch-22 was to permit one certificate to apply to multiple domains. These multidomain certs are called subject alternative name (SAN) certificates. All but the oldest, most broken web browsers support it. Since there is one certificate, the expiration data and other information are the same for all domains, which isn’t usually a problem. However, the certificate must be reissued to add and remove domains, which is inconvenient.

A newer technique is called server name indication (SNI). This permits many individual certificates on the same machine, as one would expect. The way SNI works is that the client sends the intended domain unencrypted before sending the encrypted portion. The server can determine which certificate to use to decrypt the session. For safety reasons, the intended domain is still included in the encrypted portion and must match the unencrypted request. Sadly, SNI required a change to the SSL protocol, which means older devices do not support it. However, these devices have mostly disappeared.

46.3 Service Level Agreements

Like any other service, a web service needs an SLA. Many customers are used to thinking of the web as a 24/7 critical service, but the SLA of an individual web service might be quite different. Most internal web services will have the same

SLA as other office services, such as printing or storage.

If in doubt about the appropriate SLA for a web service, ask the customer group using it. Ideally, as with any SLA, the service level should be set by collaborating with the customer community. Resist setting any SLA that does not allow for periodic maintenance, unless the service is built out on redundant infrastructure. If the service is provided by a single host or a shared web host and is required to be available around the clock, it is time to have a discussion about increasing the redundancy of the service.

Metrics that are part of a web SLA should include the latency for a certain level of QPS. That is, how long should a typical query take when the system is under a particular load? Latency is usually measured as the time between receipt of the first byte of the request and sending of the last byte of the answer.

46.4 Monitoring

Monitoring your web services lets you find out how well it is scaling, which areas need improvement, and whether the service is meeting your SLA. [Chapter 38](#), “[Service Monitoring](#),” covers most of the material you will need for monitoring web services.

You may wish to add a few web-specific elements to your monitoring. Web server errors are most often related to problems with the site’s content and are often valuable for the web development team. Certain errors or patterns of repeating errors can be an indication of customer problems with the site’s scripts. Other errors may indicate an intrusion attempt. Such scenarios are worth investigating further.

Typically, web servers allow logging of the browser client type and of the URL of the page containing the link followed to your site (the referring URL). Some web servers may have server-specific information that would be useful as well, such as data on active threads and per-thread memory usage. Become familiar with any special support for extended monitoring available on your web server platform.

46.5 Scaling for Web Services

Mike O’Dell, founder of the world’s first ISP (UUNET), famously said, “Scaling is the only problem on the Internet. Everything else is a subproblem.” Volume 2 of this book series covers scaling for large-scale, highly reliable web services in much more detail. This section gives a short overview for enterprise environments.

If your web server is successful, it will get overloaded by requests. You may have heard of **the slashdot effect**. This phrase refers to a popular Internet news site with so many readers that any site mentioned in its articles often gets overloaded.

There are several methods of scaling. A small organization with basic needs could improve a web server's performance by simply upgrading the CPU, disks, memory, and network connection—individually or in combination. This single web server design is shown in [Figure 46.1](#).

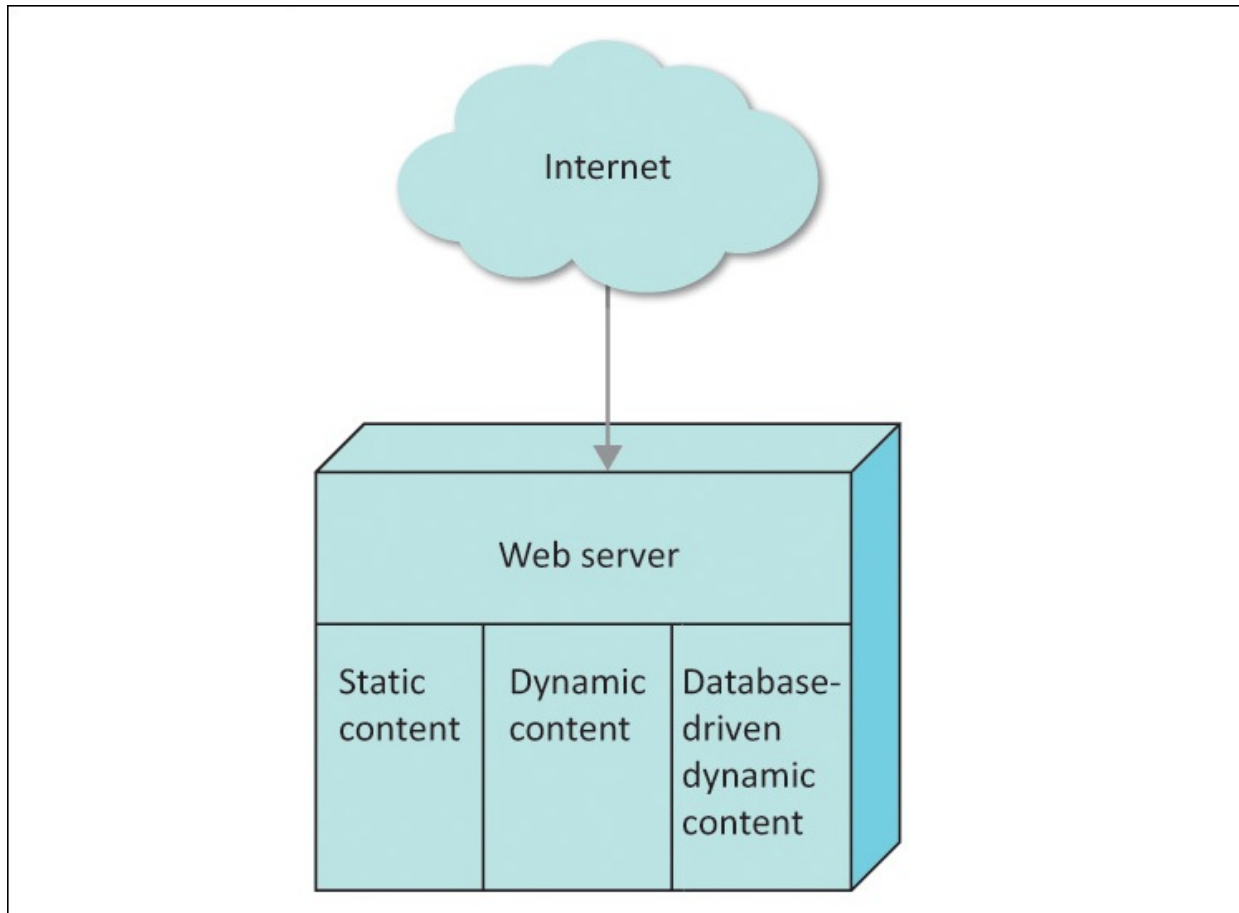


Figure 46.1: Single-machine web server

When multiple machines are involved, the two main types of scaling are *horizontal* and *vertical*. They get their names from web architecture diagrams. When drawing a representation of the web service cluster, the machines added for horizontal scaling tend to be in the same row, or level; for vertical scaling, they are in groups arranged vertically, as they follow a request flowing through different subsystems.

46.5.1 Horizontal Scaling

In **horizontal scaling**, a web server or web service resource is replicated and the load divided among the replicated resources. An example is two web servers with the same content, each getting approximately half the requests.

The typical solution is to use a device called a load balancer. A load balancer sits between the web browser and the servers. The browser connects to the IP address of the load balancer, which forwards the request transparently to one of the replicated servers, as shown in [Figure 46.2](#). The load balancer tracks which servers are down and stops directing traffic to a host until it returns to service. Other refinements, such as routing requests to the least-busy server, can be implemented as well.

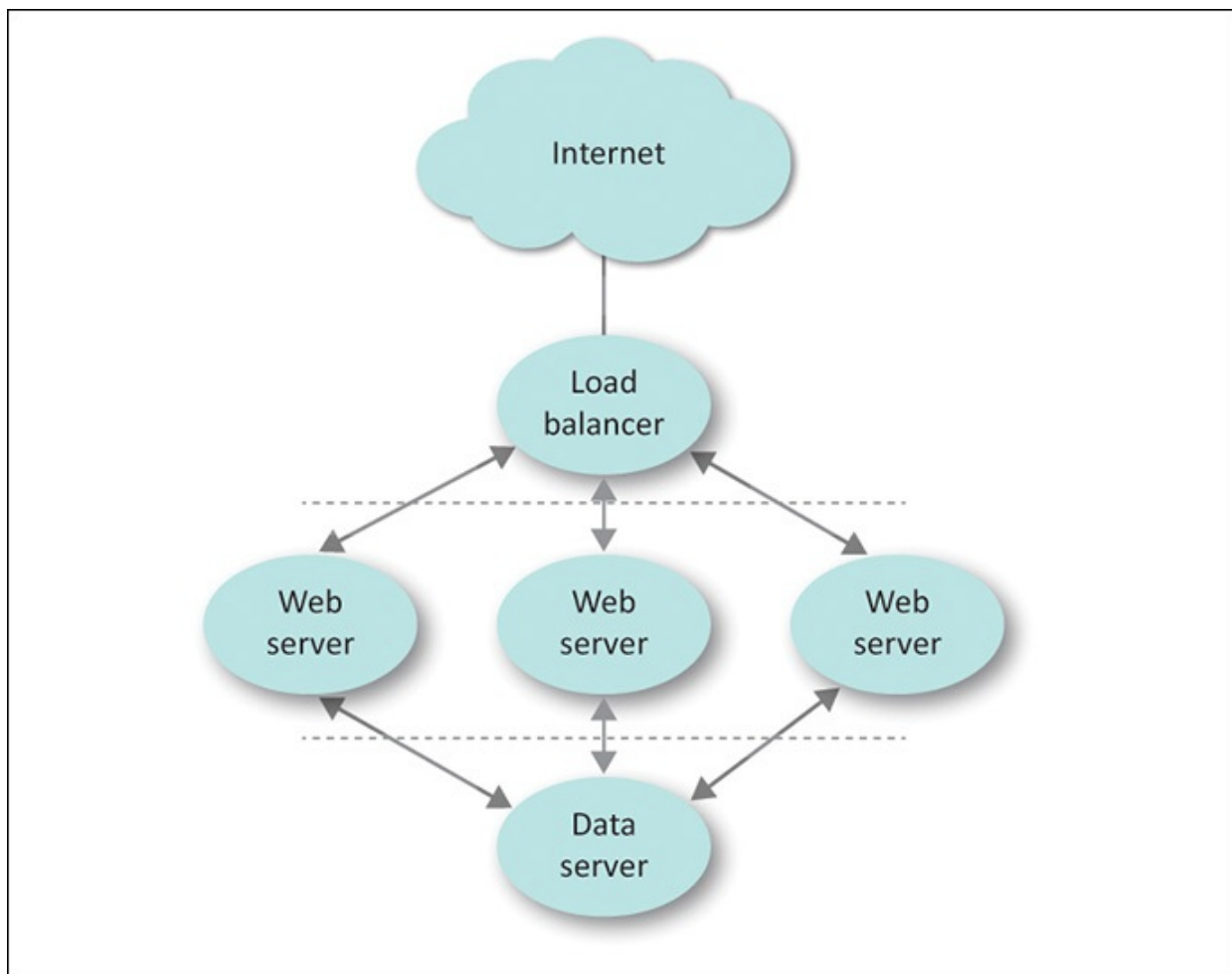


Figure 46.2: Horizontal scaling with a load balancer and shared database

Load balancers are often general-purpose protocol and traffic shapers, routing not only HTTP but also other protocol requests, as required. This allows much more flexibility in creating a web services architecture. Almost anything can be load balanced, and this approach can be an excellent way to improve both

performance and reliability.

46.5.2 Vertical Scaling

Another way to scale is to separate out the various kinds of subservices used in creating a web page, rather than duplicating a whole machine. Such **vertical scaling** allows you to create an architecture with finer granularity, so that you can put more resources at the most intensively used stages of page creation. It also keeps different types of requests from competing for resources on the same system. For example, you might take the single web server that is depicted in [Figure 46.1](#), split the different types of content onto separate machines, and change the web pages to refer to those other machines for that content.

A good example of this might be a site containing a number of large video clips and an application to fill out a brief survey about a clip after it has been viewed. Reading large video files from the same disk while trying to write many small database updates is not an efficient way to use a system. Most operating systems have caching algorithms that are automatically tuned for one or the other, but perform badly when both happen. In this case, all the video clips might be put on a separate web server, perhaps one with a storage array customized for retrieving large files. The rest of the web site would remain on the original server. Now that the large video clips are on a separate server, the original server can handle many more requests.

As you might guess, horizontal and vertical scaling can be combined. The video survey web site might need to add another video clip server before it would need to scale the survey form application. Scaling web services, especially for large sites, is covered in much more detail, and with more approaches, in Volume 2 of this book series.

46.5.3 Choosing a Scaling Method

Your site may need horizontal or vertical scaling or some combination of both. To know which you need, classify the various components that are used in conjunction with your web server according to the resources they use most heavily. Then look at which components compete with one another and whether one component interferes with the function of other components.

A site may include static files, CGI programs, and a database. Static files can range from comparatively small documents to large multimedia files. CGI programs can be memory-intensive or CPU-intensive processes, and can produce large amounts of output. Databases usually require the lion's share of system

resources.

Use system diagnostics and logs to see which kinds of resources are being used by these components. In some cases, such as the video survey site, you might choose to move part of the service to another server. Another example is an IS department web server that is also being used to create graphs of system logs. This can be a very CPU-intensive process, so the graphing scripts and the log data can be moved to another machine, leaving the other scripts and data in place.

A nice thing about scaling is that it can be done one piece at a time. You can improve overall performance with each iteration, but don't necessarily have to figure out your exact resource profile the first time you attempt it.

It is tempting to optimize many parts at once. We recommend the opposite. Determine the most overloaded component, and separate it out or replicate it. Then, if there is still a problem, repeat the process for the next overloaded component. Doing this one component at a time yields better results and makes testing much easier. It can also be easier to obtain budget allocations for incremental improvements than for one large upgrade.

The Importance of Scaling

Everyone thinks that scaling isn't important until it is too late. The Florida Election Board web site contained very little information and therefore drew very little traffic. During the 2000 U.S. national elections, the site was overloaded. Since the web site was on the same network as the entire department, the entire department was unable to access the Internet because the connection was overloaded by people trying to find updates on the election. This prevented the staff from updating the web site.

In summary, here is the general progression of scaling a typical web site that serves static and dynamic content and includes a database: Initially, these three components are on the same machine. As the workload grows, we typically move each of these functions to a separate machine. As each of these components becomes overloaded, it can be scaled individually. The static content is easy to replicate. Often, many static content servers receive their content from a large, scalable network storage device: NFS server or SAN. The dynamic content servers can be specialized and/or replicated. For example, the dynamic pages related to credit card processing may be moved to a dedicated machine; the dynamic pages related to a particular application, such as

displaying pages of a catalog, may be moved to another dedicated machine. These machines can then each be upgraded or replicated to handle greater loads.

46.6 Web Service Security

Implementing security measures is a vital part of providing web services. Security is a problem because people you don't know are accessing your server. Some sites believe that security is not an issue for them, since they do not have confidential documents or access to financial information or similar sensitive data. However, the use of the web server itself and the bandwidth it can access are, in fact, a valuable commodity.

Intruders often break into hosts to use them for money-making purposes, and sometimes they do so simply for entertainment. Intruders usually do not deface or alter a web site, since doing so would lead to their discovery. Instead, they simply use the site's resources. Common uses of hijacked sites and bandwidth include distributing pirated software ("warez"), generating advertising email ("spam"), launching automated systems to try to compromise other systems, and even competing with other intruders to see who can run the largest farm of machines ("bot" farm) to launch all the previously mentioned operations. (Bot farms are often used to perform fee-for-service attacks and are very common.)

Many companies subscribe to cloud-based platform as a service (PaaS) offerings for their customer-facing web sites. The PaaS provider takes care of the machines, the OS, and the web server software, including security and vulnerability management. The PaaS provider also takes care of scaling, for a price.

Internal web services should be secured as well. Although you may trust employees of your organization, there are still several reasons to practice good web security internally:

- Many viruses transmit themselves from machine to machine via email and then compromise internal servers.
- Intranet sites may contain privileged information that requires authentication to view, such as human resources or finance information.
- Most organizations have visitors—temps, contractors, vendors, speakers, interviewees—who may be able to access your web site via conference room network ports or with a laptop while on site.
- If your network is compromised, whether by malicious intent or accidentally by a well-meaning person setting up a wireless access point reachable from outside the building, you need to minimize the potential

damage that could occur.

- Security and reliability go hand-in-hand. Some security patches or configuration fixes also protect against accidental denial-of-service attacks that could occur and will make your web server more reliable.

Large companies that build their own private cloud will usually have basic web services as their first offering. The internal PaaS group handles internal web site security and streamlines its build and patch process for fast roll-out.

In addition to problems that can be caused on your web server by intrusion attempts, a number of web-based intrusion techniques can reach your customers via their desktop browsers. We talk about these sorts of attacks later in this chapter.

New security exploits are frequently discovered and announced, so the most important part of security is staying up-to-date on new threats.

46.6.1 Secure Connections and Certificates

Usually web sites are accessed using unencrypted, plaintext communication. The privacy and authenticity of the transmission can be protected by using HTTP over Secure Sockets Layer (SSL) to encrypt the web traffic. SSL 4.0 is also known as Transport Layer Security (TLS) 1.0. Confusingly, SSL 2.0 and 3.0 predate TLS 1.0. We use encryption to prevent casual eavesdropping on our customers' web sessions even if they are connecting via a wireless network in a public place, such as a coffeeshop. URLs using `https://` instead of `http://` are using SSL encryption.

Implementing HTTPS on a web server is relatively simple, depending on the web server software being deployed. Properly managing the cryptographic certificates is not so easy.

SSL depends on cryptographic certificates, which are strings of bits used in the encryption process. A certificate has two parts: the private half and the public half. The public half can be revealed to anyone. In fact, it is given out to anyone who tries to connect to the server. The private part, however, must be kept secret. If it is leaked to outsiders, they can use it to pretend to be your site. Therefore, one role of the web system administrator is to maintain a repository, or key escrow, of certificates for disaster-recovery purposes. Treat this data the same way as you manage other important secrets, such as root or administrator passwords.

One dangerous place to store the private half is on the web server that will be using it. Web servers are generally at a higher exposure risk than other servers.

Storing an important bit of information on the machine that is most likely to be broken into is a bad idea. However, the web server needs to read the private key to use it. There are a number of ways to solve this issue:

- **Isolated storage:** Sites with extremely high security requirements store the key on a USB stick that is inserted into the machine only when the key is needed. This strategy requires someone to temporarily insert the USB device into the machine anytime the software is restarted.
- **Encrypted storage:** Sites with fewer security requirements simply store the key on the machine in encrypted form. A person is required to enter the decryption passphrase anytime the software is restarted.
- **Network-accessible key repository:** Some sites have a key repository service that is cryptographically secure. The machine can securely request the key over the network, and the key server can authenticate that the request is really coming from the machine that it claims to be.
- **Protected file:** Sites with lower security requirements simply store the file with permissions that can be read only by the specific processes that will need the data. Some systems read the file as `root`, then switch to a lower-privileged user to run the service. With this approach, if there is a security intrusion, the invader cannot steal the private key.

A cryptographic certificate is created by the web system administrator using software that comes with the encryption package; OpenSSL is one popular system. The certificate at that point is “self-signed,” which means that when someone connects to the web server using HTTPS, the communication will be encrypted, but the client that connects has no way to know that it has connected to the right machine. Anyone can generate a certificate for any domain. It is easy to trick a web client into connecting to a “man in the middle” attacker instead of to the real server; the client won’t know the difference. This is why most web browsers, when connecting to such a web site, display a warning stating that a self-signed certificate is in use.

What’s to stop someone pretending to be a big e-commerce site from gathering people’s login information by setting up a fake site? The solution to this dilemma is to use an externally signed cryptographic certificate from a registered certification authority (CA). The public half of the self-signed certificate is encrypted and sent to a trusted CA, which signs it and returns the signed certificate. The certificate now contains information that clients can use to verify that the certificate has been certified by a higher authority. When it connects to the web site, a client reads the signed certificate and knows that the

site's certificate can be trusted because the CA says that it can be trusted. Through cryptographic techniques beyond what can be explained here, the information required to verify such claims is stored in certificates that come built into the browser, so that the browser does not need to contact the CA for every web site using encryption.

A large company may create its own CA. It can then sign as many certificates as it wishes at no additional cost. Browsers will know to trust these certificates one of two ways. The company can configure browsers to trust this CA by adding the appropriate CA certificate information to the browser. This is labor intensive and will cause problems for any browser or other system that does not receive the modification. Alternatively, the private CA's certificate can be signed by one of the CAs that is already trusted by the browsers: These are called the *root CAs*. The browser then does not require special configuration because it will automatically trust the company's CA, since it is trusted by a CA that trusts it. This is called the *hierarchy of trust*.

46.6.2 Protecting the Web Server Application

A variety of malicious efforts can be directed against the web server itself in an attempt to get login access to the machine or administrative access to the service. Any vulnerabilities present in the operating system can be addressed by standard security methods. Web-specific vulnerabilities can be present in multiple layers of the web server implementation: the HTTP server, modules or plug-ins that extend the server, and web development frameworks running as programs on the server. This last category is distinct from generic applications on the server, as the web development framework is serving as a system software layer for the web server.

The best way to stay up-to-date on web server security at those layers is through vendor support. The various HTTP servers, modules, and web development environments often have active mailing lists or discussion groups and almost always have an announcements-only list for broadcasting security exploits, as well as available upgrades.

Implementing service monitoring can make exploit attempts easier to detect, as unusual log entries are likely to be discovered with automated log reviews. (See [Section 17.6.1](#) and [Chapter 38](#), "[Service Monitoring](#).")

46.6.3 Protecting the Content

Some web-intrusion attempts are directed toward gaining access to the content

or service rather than to the server. There are too many types of web content security exploits to list them all here, and new ones are always being invented. We discuss a few common techniques as an overview.

It is strongly recommended that SAs who are responsible for web content security educate themselves on current exploits via Internet security resources. To properly evaluate a server for complex threats is a significant undertaking. Fortunately, open source and commercial packages to assist you are available.

Software Vulnerabilities

The single largest threat is security vulnerabilities in the software and frameworks used by your application. Your code may be perfect, but if it relies on an old version of a framework or system, it may be riddled with security vulnerabilities. Studies have found that the majority of security incidents are not due to the newest security threats, but rather are attributable to unfixed vulnerabilities that are very old.

Directory Traversal

A technique generally used to obtain data that would otherwise be unavailable is directory traversal. The data may be of interest in itself or may be obtained to enable some method of direct intrusion on the machine. This technique generally takes the form of using the directory hierarchy to request files directly, such as `../../../../some-file`. When used on a web server that automatically generates a directory index, directory traversal can be used with great efficiency. Most modern web servers protect against this technique by implementing special defenses around the document root directory and refusing to serve any directories not explicitly listed by their full pathnames in a configuration file. Older web implementations may be vulnerable to this technique, along with new, lightweight, or experimental web implementations, such as those in equipment firmware.

A common variation of directory traversal uses a CGI query that specifies information to be retrieved, which internally is a filename. A request for `q=maindoc` returns the contents of `repositorymaindoc.data`. If the system does not do proper checking, a user requesting `../../../../paidcontent/prize` would be able to gain free but improper access to that file.

Form-Field Corruption

Another technique used by invaders is form-field corruption, which uses a site's

own web forms, which contain field or variable names that correspond to customer input. These names are visible in the source HTML of the web form. The intruder copies a legitimate web form and alters the form fields to gain access to data or services. If the program being invoked by the form is validating input strictly, the intruder may be easily foiled. Unfortunately, intruders can be inventively clever and may think of ways around restrictions.

For example, suppose that a shopping cart form has a hidden variable that stores the price of the item being purchased. When the customer submits the form, the quantities chosen by the customer are used, with the hidden prices in the form, to compute the checkout total and cause a credit card transaction to be run. An intruder modifying the form could set any prices arbitrarily. There are cases of intruders changing prices to a negative amount and receiving what amounts to a refund for items not purchased.

This example brings up a good point about form data. Suppose that the intruder changed the price of a \$50 item to be only \$0.25. A validation program cannot know this in the general case. It is better for the form to store a product's ID and have the system refer to a price database to determine the actual amount to be charged.

SQL Injection

A variant of form-field corruption is SQL injection. In its simplest form, an intruder constructs a piece of SQL that will always be interpreted as “true” by a database when appended to a legitimate input field. On data-driven web sites or those with applications powered by a database backend, this technique lets intruders do a wide range of mischief. Depending on the operating system involved, intruders can access privileged data without a password, and can create privileged database or system accounts or even run arbitrary system commands. Intruders can even perform entire SQL queries, updates, and deletions! Some database systems include debugging options that permit running arbitrary commands on the operating system.

46.6.4 Application Security

The efforts of malicious people can be rendered less likely to succeed by following good security practices when developing applications. For valuable further reading, we recommend the book *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services* by Andrews & Whittaker ([2006](#)). The following subsections describe some of the fundamental practices to use when writing web code or extending server

capabilities.

Limit the Potential Damage

One of the best protections one can implement is to limit the amount of damage an intruder can do. Suppose that the content and programs are stored on an internal golden master environment and merely copied to the web server when changes are made and tested. An intruder defacing the web site would accomplish very little, as the machine could be easily re-imaged with the required information from the untouched internal system.

If the web server is isolated on a network of its own, with no ability to initiate connections to other machines and internal network resources, the intruder will not be able to use the system as a stepping-stone toward control of other local machines. Necessary connections, such as those performing backups, collecting log information, and installing content upgrades, can be set up so that they are always initiated from within the organization's internal network. Connections from the web server to the inside would be refused.

Validate Input

It is crucial to validate the input provided to interactive web applications so as to maximize security. Input should be checked for length, to prevent buffer overflows where executable commands could be deposited into memory. User input, even of the correct length, may hide attempts to run commands, sometimes by using quotes or escape characters.

Enclosing user input in so-called safe quotes or disallowing certain characters can work in some cases to prevent intrusion but can also cause problems with legitimate data. For example, filtering out or rejecting characters, such as a single quote mark or a dash, might prevent Patrick O'Brien or Edward Bulwer-Lytton from being registered as users.

It is better to validate input by inclusion than by exclusion. That is, rather than trying to pick out characters that should be removed, remove all characters that are not in a particular set. Even better, adopt programming paradigms that do not reinterpret or reparse data for you. For example, use binary APIs rather than ASCII, which will be parsed by lower-level systems.

Automate Data Access

Programs that access the database should be as specific as possible. If a web application needs to read data only from the database, have it open the database in a read-only mode or run as a user with read-only access. If your database

supports stored procedures—essentially, precompiled queries—develop ones to do what you require, and use them instead of executing SQL input.

Many databases and scripting languages include a **preparation function** that can be used to convert potentially executable input into a form that will not be interpreted by the database. As a result, the input cannot be subverted into an intrusion attempt.

Use Permissions and Privileges

Web servers generally interface well with the authentication methods available on the operating system, and have options for permissions and privileges local to the web server itself. Use these features to avoid giving any unnecessary privileges to web programs. Apply the least-privilege security principle to web servers and web applications. Then, if someone manages to compromise the web server, it cannot be used as a springboard for compromising the next application or server.

Use Logging

Logging is an important protection of last resort. After an intrusion attempt, detailed logs will permit more complete diagnostics and recovery. To hide their activities, smart intruders will attempt to remove log entries related to the intrusion or to truncate or delete the log files entirely. Logs should be stored on other machines or in nonstandard places to make them difficult to tamper with. For example, intruders know about the Unix *var* log directory and will delete files in it. Many sites have been able to recover from intrusions more easily by simply storing logs outside that directory.

Another way of storing logs in a nonstandard place is to use network logging. Few web servers support network logging directly, but most can be set up to use the operating system's logging facilities. Most OS-level logging includes an option to route logs over the network onto a centralized log host.

46.7 Content Management

Providing a content-management system (CMS) empowers users to perform self-service updates of the web content. A CMS lets you create privileged accounts for those users who are permitted to perform these updates, often in some kind of controlled manner.

It is not a good idea for an SA to be directly involved with content updates. Assuming this responsibility not only adds to the usually lengthy to-do list of the

SA, but also creates a bottleneck between the creators of the content and the publishing process.

Instituting a web council makes attaching domains of responsibility for web site content much easier, because the primary “voices” from each group are already working with the webmaster or the SA who is being a temporary webmaster. The web council is the natural owner of the change control process.

This process should have a specific policy on updates. Ideally, the policy should distinguish three types of alterations that might have different processes associated with them:

- **Update:** Adding new material or replacing one version of a document with a newer one
- **Change:** Altering the structure of the site, such as adding a new directory or redirecting links
- **Fix:** Correcting document contents or site behavior that does not meet the standards

The process for making a fix might involve a trouble ticket or bug report being open and the fix passing quality assurance (QA) checks. The process for making an update might involve having a member of the web council request the update, with QA then approving the update (and keeping an approval document on file) before it is pushed to the site. A similar methodology is used in many engineering scenarios, where changes are classified as bug fixes, feature requests, and code (or spec) items.

Policy and Automation Mean Less Politics

When Tom worked at a small start-up, the debate over pushing updates to the external web site became a big political issue. Marketing wanted to be able to control everything, quality assurance wanted to be able to test things before going live, engineering wanted to make sure everything was secure, and management wanted everyone to stop bickering.

The web site contained mostly static content and wouldn't be updated more than once a week. Tom and a co-worker established the following process. First, they set up three web servers:

- www-draft.example.com: The work area for the web designer, not accessible to the outside world
- www-qa.example.com: The web site as it would be viewed by quality assurance and anyone proofing a site update, not accessible to

the outside world

- www.example.com: The live web server, visible from the Internet

The web designer edited `www-draft` directly. When ready, the content was pushed to `www-qa`, where people checked it. Once approved, it was pushed to the live site.

Initially, the SAs were involved in pushing the content from one step to the next. This put them in the middle of the political bickering. Someone would tell the SAs to push the current QA content to the live site, then a mistake would be found in the content, and everyone would blame the SAs. They would be asked to push a single file to the live site to fix a problem, and the QA people would be upset that they hadn't been consulted.

The solution was to create a list of people and the systems those people were allowed to move data from, and to automate the functions to make them self-service, thereby taking the SAs out of the loop. Small programs were created to push data to each stage, and permissions were set using the Unix `sudo` command so that only the appropriate people could execute particular commands.

Soon, the SAs had extricated themselves from the entire process. Yes, the web site got messed up in the beginning. Yes, the first time marketing used its emergency-only power to push from draft directly to live, it was, well, the last time it ever used that command. But over time everyone learned to be careful.

But most important, the process was automated in a way that removed the SAs from the updates and the politics.

46.8 Summary

The web is a universal service delivery system in every organization, with the web browser client providing a common interface to multiple applications. Web services should be designed with the primary applications in mind so that they can be appropriately scaled, either by adding instances (horizontally) or by balancing the load among different service layers (vertically). Installing a simple web site as a base for various uses is relatively easy, but making sure that the systems staff doesn't end up tasked with content maintenance is more difficult. Forming a council of web stakeholders, ideally with a designated team of webmasters, can solve this problem and scale as the organization's web usage

grows. Another form of scaling is to outsource services that are simple enough or resource-intensive enough that the system staff's time would be better spent doing other things.

Exercises

1. Which issues need to be considered differently when providing external web services versus internal services to your organization?
2. How many cryptographic certificates are in use by your organization? How are they managed? Which improvements would you make to how they are managed?
3. Which methods does your organization use to provide multiple web sites on the same machine?
4. Give an example of a type of web-specific logging that can be done and what that information is used for.
5. Pick a web service in your organization, and develop a plan for scaling it, assuming that it will receive 5 times as many queries. Now do the same while assuming it will receive 100 times as many queries.
6. Is your organization's external web site hosted by a third-party hosting provider? Why or why not? Evaluate the pros and cons of moving it to a third-party hosting provider or bringing it internal to your organization.

Part IX: Management Practices

Chapter 47. Ethics

Which policies related to ethics should a site have? What are the additional ethical responsibilities of SAs and others with privileged technical access? This chapter discusses both of these issues.

Ethics, the principles of conduct that govern a group of people, are different from morals. **Morals** are a proclamation of what is right and good, a topic of discussion that is beyond the scope of this book.

Whether your organization involves you in drafting the ethics guidelines for all users or only the SAs, bring this chapter along. It provides you with the tools you need to get the job done.

Organizations usually have various ethics-related policies for their employees and other affiliates. Ethics policies concerning network use fall into two categories: policies applying to all users and policies applying only to privileged users, such as SAs and database administrators. In general, an SA needs to carefully follow company policies, and to set a higher standard. Through elevated system privileges, SAs have access to confidential information that is normally restricted on a need-to-know basis; as a result, SAs have special responsibilities.

In recent years, U.S. and European Union (EU) legislation has mandated enhanced corporate responsibility for ethical policies and controls with respect to IT. Laws such as the Sarbanes-Oxley Act (SOX), the Family Educational Rights and Privacy Act (FERPA), and the Health Insurance Portability and Accountability Act (HIPAA) have changed the way companies think about these issues. The occupation of system administration has been directly affected.

47.1 Informed Consent

The principle of **informed consent** comes from the the field of medical ethics. All medical procedures carry some level of risk. Therefore, to ethically perform them on a patient, the patient should be informed about the risk, and should consent to the procedure. Being informed means the patient should be fully educated as to all the treatment options, all the possible benefits and drawbacks of those options, and the various probabilities of success. The information should be explained in whatever way the person is competent to understand. Consent means the patient must be given the opportunity to permit the treatment or refuse it, without coercion of any sort.

Consent is not possible if someone is legally incompetent (unable to understand the ramifications) or unable to give consent—for example, a person in a coma who has no next of kin. In such cases the accepted standard is to fully satisfy all three of the following conditions. First, the procedure must have a high likelihood of success. Second, it must be in the *patient's* best interest, such that if the procedure is successful, the person would likely be thankful in retrospect. Third, all other avenues of obtaining consent must have been attempted first. In other words, violating the requirement for informed consent must be a last resort.

These principles can be applied to many SA tasks. People should understand the rules under which they operate. For example, if an acceptable use policy (AUP) is enforced upon users, they should know what that policy is. If an SLA specifies that intrusive maintenance will happen during certain hours, those hours should be published.

Users should consent to changes being made, especially in extraordinary situations. Suppose a compute server is used for long-running batch jobs such as simulations. If the simulation software doesn't have a checkpointing feature, a reboot might lose days or weeks of work. If a reboot is absolutely unavoidable, the SLA might specify that the current users of the machine will be involved in the decision to reboot. In contrast, compute servers for short-term batch jobs might have a blanket SLA that specifies little to no warning is required for reboots.

The Right Reminder at the Right Time

Systems should remind people of a policy at the moment that the policy applies directly to them.

The popular Unix utility `sudo` prints this message when it is about to grant elevated privileges to a user:

[Click here to view code image](#)

```
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these two things:
```

- ```
1) Respect the privacy of others.
2) Think before you type.
```

---

## **47.2 Code of Ethics**

USENIX and other organizations have jointly published a System



Administrators' Code of Ethics. It does an excellent job of expressing the need for SAs to maintain a high level of professionalism. We encourage you to adopt it as your own code of ethics, and use it as the basis of your organization's SA code of conduct. It is intentionally *not* a set of enforceable laws, an enumeration of procedures, proposed responses to situations, or an inventory of sanctions and punishments. Rather, it is a framework that can be used as the basis for all those things. It is included on the next page and is also available as a PDF suitable for printing as a poster at <https://www.usenix.org/lisa/system-administrators-code-ethics>.

### 47.3 Customer Usage Guidelines

Every organization needs a set of guidelines for acceptable uses of that organization's computers. Companies and Internet service providers often refer to these agreements as an acceptable use policy (AUP); colleges often call them a user code of conduct (UCC). The guidelines address some of the following points: What is considered acceptable and unacceptable use? Under which circumstances is personal use of employer equipment permitted? Which types of personal use are forbidden? Can an employee run a fledgling dot-com out of his or her cubicle? Can an employee post to a blog, Facebook, or Twitter from work? How about using the organization's computer for surfing "adult" web sites? How do the rules change if an employee is using company equipment at home?

Network users have to understand that monitoring may happen as part of running the network, either as part of a AUP, or in a separate monitoring and privacy policy.

The AUP or UCC should define and forbid threatening or harassing communications, explain how to report them, and explain how complaints are processed.

Codes of conduct at academic institutions are usually very different from codes of conduct in industry. The differences stem from the requirements for academic freedom and the fact that, for many students, the campus *is* home.

You can find sample policies through various industry and academic consortia, which often have a web site with a collection of policies from various organizations. Dijker (1999) is one such archive. The best way to write a policy is to use an archive to find a policy whose philosophy is close to your own and then use it as a base document.

---

## **The System Administrators' Code of Ethics**

*We as professional System Administrators do hereby commit ourselves to the highest standards of ethical and professional conduct, and agree to be guided by this code of ethics, and encourage every System Administrator to do the same.*

**Professionalism** I will maintain professional conduct in the workplace and will not allow personal feelings or beliefs to cause me to treat people unfairly or unprofessionally.

**Personal Integrity** I will be honest in my professional dealings and forthcoming about my competence and the impact of my mistakes. I will seek assistance from others when required.

I will avoid conflicts of interest and biases whenever possible. When my advice is sought, if I have a conflict of interest or bias, I will declare it if appropriate, and recuse myself if necessary.

**Privacy** I will access private information on computer systems only when it is necessary in the course of my technical duties. I will maintain and protect the confidentiality of any information to which I may have access, regardless of the method by which I came into knowledge of it.

**Laws and Policies** I will educate myself and others on relevant laws, regulations, and policies regarding the performance of my duties.

**Communication** I will communicate with management, users, and colleagues about computer matters of mutual interest. I will strive to listen to and understand the needs of all parties.

**System Integrity** I will strive to ensure the necessary integrity, reliability, and availability of the systems for which I am responsible.

I will design and maintain each system in a manner to support the purpose of the system to the organization.

**Education** I will continue to update and enhance my technical knowledge and other work-related skills. I will share my knowledge and experience with others.

**Responsibility to Computing Community** I will cooperate with the larger computing community to maintain the integrity of network and computing resources.

**Social Responsibility** As an informed professional, I will encourage the writing and adoption of relevant policies and laws consistent with these

ethical principles.

**Ethical Responsibility** I will strive to build and maintain a safe, healthy, and productive workplace.

I will do my best to make decisions consistent with the safety, privacy, and well-being of my community and the public, and to disclose promptly factors that might pose unexamined risks or dangers.

I will accept and offer honest criticism of technical work as appropriate and will credit properly the contributions of others.

I will lead by example, maintaining a high ethical standard and degree of professionalism in the performance of all my duties. I will support colleagues and co-workers in following this code of ethics.



Copyright © 2014 The USENIX Association. USENIX grants permission to reproduce this Code in any format, provided that the wording is not changed in any way, that signatories USENIX, LISA, and LOPSA are included, and that no other signatory or logo is added without explicit permission from the copyright holder(s).

---

## 47.4 Privileged-Access Code of Conduct

Some users need privileged access to do their jobs. The ability to write and debug device drivers, install software for other people, and perform many other tasks all require `root` or administrator access. Organizations need special codes of conduct for these people; as we all know, privileges can be abused. This code of conduct should include the following points:

- The individual acknowledges that privileged access comes with a responsibility to use it properly.
- The individual promises to use elevated access privileges solely for necessary work-related uses. Management should explicitly describe these uses.
- The company acknowledges that mistakes happen and encourages procedures for minimizing the damage a mistake may cause. For example, SAs should make backups before making any changes.
- Procedures should be defined for what to do if privileged access gives someone information about something that wouldn't have otherwise been

made public. For example, what should an SA do if while fixing someone's email, the SA observes illegal behavior (running a gambling operation from the organization's network) or something less nefarious but just as important (a message about a pending merger).

- The consequences of making a mistake should be specified. The best policy is to state that there will be no punishment for making an honest mistake as long as it is reported quickly and completely. The sooner a mistake is reported, the sooner it can be fixed and the less domino-effect damage that will occur.
- A warning should be given about the possible penalties, including termination, for violating the policy.

People with privileged access should sign a statement saying that they have read the code of conduct for privileged users. The original should be filed with the person's manager or HR department, whichever is customary at the organization. Both the individual and that person's manager should be given a copy of it for their files.

As a good security measure, the SA team should track who has privileged access to which systems. This practice is especially useful for alerting the SAs to remove access privileges when a privileged user leaves the organization. Some organizations have a policy that privileged access expires every 12 months unless the form is re-signed. This practice encourages regular policy reviews. Automatic reminders are another good tool.

Sometimes, these policies are governed by federal law. For example, the Securities and Exchange Commission (SEC) has rules against monitoring networks used to trade stock, which can make debugging a network problem on Wall Street very difficult. Also, the Federal Communications Commission (FCC) has rules about how telephone operators and technicians can use information accidentally obtained while doing their jobs. These people can discuss the information only with its source and can't use it for personal gain.

---

### **Guidance for Junior SAs**

Tom gives these instructions to junior SAs that he hires. The point of this guidance is that we are all human and make mistakes. Being open about your mistakes is the best policy.

**Tom's Three Rules of Privileged Access (1) Be careful. (2) Respect privacy. (3) If you mess up, tell me.**

**Rule 1: Be careful.** You can do a lot of damage as `root/Administrator`,

database admin, and so on, so be careful. Make backups. Pause before you press ENTER. Make backups. Test wildcards before you use them. Make backups. Pay attention to what you are doing. Make backups. Don't drink and compute. Make backups.

**Rule 2: Respect privacy.** Don't look at anything that isn't required for a task. Don't "browse." Don't look at anything if you wouldn't want someone else looking at it if it were yours.

**Rule 3: If you mess up, tell me right away.** You will make mistakes. That's okay. You will never be punished for an honest mistake. Tell me as soon as you realize it's beyond what you can fix. It's most likely my job to fix your mistake, and you will have to watch as I do so. The sooner you tell me, the happier I will be, because the less I will have to fix. However, if you hide a mistake and I have to discover the issue on my own, I will figure out what the mistake was, and who made it, and there will be negative consequences.

---

## 47.5 Copyright Adherence

Organizations should have policies stating that their members abide by copyright laws. For example, software piracy is pervasive, and many people don't realize that "borrowing" software that is not freely redistributable is actually *illegal*.

Companies are very concerned about being caught using pirated software. The financial liabilities and bad publicity don't make executives or shareholders very happy. Add this fact to the highly publicized raids conducted by anti-software-piracy organizations, and you have a recipe for disaster. Pirated software is also a vector for spreading computer viruses. The bottom line: Don't use pirated software on company equipment, and don't let users sneak it past you.

Telling people not to pirate software isn't all that effective, because they're usually convinced that what they're doing is not software piracy. Many people don't understand what constitutes software piracy. Even if they do, they will try to plead ignorance when caught. "I thought we had a site license." "I didn't know it was also installed on another machine." "Someone told me it was okay."

The best copyright-adherence policy is clear and uses plain language. State that individually licensed PC software packages should be purchased for individual PCs, and that single-use installation media should not be used on multiple machines. Give three or four examples of the most common violations so people can recognize them.

Some companies bar employees from installing *any* software without explicit management approval. Alternatively, and for the sake of simplicity, a policy statement might specify the kinds of software that employees may download at will, such as new versions of Adobe Acrobat Reader or new versions of web browsers. Installing software not on the list would require management approval.

Finally, a statement along the following lines might be useful: “We are always striving to reduce overhead costs, and we appreciate your efforts in this area. That said, software piracy is a recipe for trouble, and we do not consider it to be a valid cost-saving measure. No one at this company is authorized to pirate software; if anyone pirates software or asks you to do so, please follow this procedure.”

---

### **Have a Witness**

A company’s buggy email server was mangling people’s mailboxes. While waiting for a software patch, the SAs discovered that the mailboxes could be fixed manually. However, an SA could see someone’s messages while fixing the mailbox. When the CEO’s mailbox was mangled, the SAs faced a challenge. The industry was experiencing a high level of merger activity, and the SAs didn’t want the responsibility that would come with accidentally seeing a critical email message in the CEO’s mailbox.

They decided that the CEO’s assistant would watch the SA while he fixed the CEO’s mailbox. That way, the assistant would see that the SA wasn’t nosing around for confidential information and would also understand how much exposure the CEO’s confidential email had received. This protected the CEO and the SA alike.

---

The easiest way to ensure policy compliance is to mold your policy into the path of least resistance: Bulk-license popular packages for all workstations. You can’t be out of compliance if you have a site license. Install it as part of the standard set of software delivered with all workstations. People are less likely to seek out alternative software packages if there is no effort required to use the one you have licensed. If this is not feasible, another approach is to require that new workstation or server requests also include necessary OS and application software or licenses.

---

### **Super-Simple License Management**

Managing bulk licenses need not be complicated. A small number of

licenses can be tracked on paper. If you have 50 right-to-use licenses for a software package, number a piece of paper with 50 lines and write the name of the person allocated to each license. Tape the paper inside the installation manual of the software. You can also use a spreadsheet or a wiki for this purpose.

You can also use group permissions as a simple way to track software licenses electronically. Suppose that you've licensed 50 copies of a software package to be given out to people as they request it. Create a group in Microsoft ActiveDirectory or LDAP named after the software package (or maybe `license_NameOfPackage`). Anyone with access to the software is then added to the group. You now know how many licenses are allocated and who has access. You can even create a mailing list based on the group if you need to communicate with these users. Best of all, when someone is terminated and his or her account is deleted, the person will be removed from the group, and the license will be freed up. This, of course, doesn't uninstall the software from someone's PC if that person leaves the company, but if machines are wiped and reinstalled between users, this is not an issue.

---

A benefit of free and open source software is that the license permits and encourages copying. There is still a license that must be adhered to, but normal use rarely creates a problem. Conversely, if employees are modifying the source code or using the source code as part of a product, the license should be carefully studied. Some large companies have a designated group to globally manage their free/open source compliance and to look for ways to better leverage their involvement in the open source community.

It is important to inform people of this sad bit of reality: When faced with a copyright-violation lawsuit, companies rarely accept the blame. Instead, they implicate whoever let the violation happen and relay the damages to that person. Document this in the policy, and make sure that the policy is communicated to all.

It is especially important for SAs to realize their vulnerability in this area. It is much more likely that the person to be blamed will be the hapless SA who used a development OS license to get a customer up and running than the manager who refused to sign the purchase order for new licenses in a timely manner. If your management tells you to break the law, refuse politely, in writing or email.



## 47.6 Working with Law Enforcement

Organizations should have a policy on working with law enforcement agencies so that SAs know what to do if one contacts them. Law enforcement officials sometimes contact SAs to help with investigations in areas of computer-related crime, as well as with harassment issues or other instances in which evidence is needed.

To protect people's privacy, your goal should be to comply with your legal responsibilities but hand over the absolute minimum amount of information as required by law. In fact, companies should have a policy of first verifying that a warrant exists and then turning over only the exact information specified within that warrant, and nothing more, and should appeal requests that are overly broad.

We feel strongly that companies should hold the privacy of users as a high priority, even if this means pushing back on overly broad requests. We are disappointed that some companies reveal much more than what is required by law, violating their users' privacy. We believe that eventually such behavior will become public and will damage the company's reputation.

An organization *must* verify the identity of a person claiming to be from a law enforcement agency before telling him or her *anything*. This includes giving out your manager's name and contact information, as it may be used for social engineering purposes. Perform this verification even before you admit that you are an SA. Explain that you'd like to verify the person's credentials and ask for the person's phone number and the agency's switchboard number. Call the switchboard and ask for the person. If you question whether the switchboard number is real, look it up online or in a phone book. Yes, the FBI, the CIA, and even the NSA are all listed!

Many SAs panic when they are contacted by law enforcement. Such a reaction is a natural response. For this reason, and to avoid violating the law or company policy, SAs need a written policy guiding them through the process. One company had the following procedure:

- Relax. Be calm.
- Be polite. (*This is included because SAs often have problems with authority and need to be reminded that being rude to an investigator is a bad thing.*)
- Refer the issue to your manager. A suggested response is "As a policy, we gladly cooperate with law enforcement. I need to refer this matter to my boss. Can I take your phone number and have her call you?" (*This is included because law enforcement will always give contact information.*)



*Pranksters and scam artists will not.)*

- If you are a manager, contact the legal department. They will authenticate the request and guide any follow-up work.
- Keep a log of all requests, all related phone calls, and any commands typed.
- The SA who collects evidence should give it to the legal department, which will give it to law enforcement unless the manager directs otherwise. *(This is included to protect the SA.)*
- If the caller is with the internal corporate security department, the evidence should be given to the SA's manager, who will give it to corporate security. Again, be polite when explaining this policy to corporate security: "We always comply with requests from your group. However, it is department policy for me to collect these logs and give them to my boss, who will then give them to you. My boss can be contacted as follows . . . ."

If you run your organization's Internet gateway, you are much more likely to be contacted by law enforcement. If law enforcement agencies contact you regularly, it's time to consider streamlining your procedures for dealing with them to avoid making mistakes or becoming a target. You might get training from the legal department and establish a procedure that lets you handle a standard police inquiry yourself and simply notify the legal department about the existence of the inquiry. This way, the legal department won't need to hold your hand every step of the way. Of course, exceptional cases should still be referred to the legal department. In the best case, a problem is quickly fixed, and no future reports are generated. However, Internet service providers and web-hosting companies may have a continual stream of problems.

---

### **Don't Be Too Accommodating**

Once upon a time, a company offered a demo of a web-based service that let people surf the web anonymously. Crackers used this service to harass other sites. This activity, unfortunately, was traced back to the anonymizing service. This was *not* good. Whenever the problem arose, law enforcement contacted the SA, who passed the message along to the people running the service. The SA would never hear of it again and assumed that appropriate action was being taken. Many services shared his Internet connection and, being a typically overworked SA, it took him a while to notice the trend: multiple requests from law enforcement all concerning the same service.

The SA was worried about the negative effects of the service but also wanted to be accommodating to his customer. He advised the group how to change the service to make it less susceptible to abuse, but the group ignored him. Soon the requests from law enforcement started taking up more of his time, as he was being listed on subpoenas. Not surprisingly, the SA became extremely frustrated and burned out.

Eventually, the SA realized that he was trying to fix this problem at the wrong level. He brought the issue to his manager, and they agreed that he shouldn't take on the responsibility of problems caused by one of his customers, especially since he had made valid suggestions on how to fix the service. At the manager's level, he could require the customer to fix its software or be cut off in 30 days. The manager also decided that it was more appropriate for the requests from law enforcement to be forwarded to the company's legal department, to make sure that they were handled properly.

The legal department shut down the service within minutes of being told that the situation existed. They had no interest in waiting the entire 30 days. The legal department was shocked that the problem had been permitted to persist at all.

In hindsight, the SA should have been tougher with the customer from the beginning. If he didn't have the backbone or the authority to shut the customer down, he should have turned the problem over to the legal department, which would have been much firmer. The moral of this story is simple: Don't try to solve every problem yourself. Be firm with people who are harming your company. If they turn into bullies, it's better to find a bigger bully to fight for you.

---

Real law enforcement personnel and real employees will provide information to verify their identities, and they should not be offended when you ask them for this information.

Failing to verify the identity of someone claiming to be a law enforcement official can turn out to be a disaster. Unfortunately, some of society's seamier characters pretend to be officials when they steal company information, using a tactic termed **social engineering**. It works like this. The perpetrator starts with a small piece of information. He or she makes telephone calls while pretending to be an official or a new employee. The offender leverages one piece of information to gain more information. This is repeated until the perpetrator gains

the information he or she needs. Social engineering has been used to get people to reveal their passwords, to explain how to access corporate proprietary information, and even to change security settings on corporate firewalls.

Sometimes, would-be social engineers hatch their plans by starting with the information they find by rooting through dumpsters and rubbish bags, a technique called dumpster diving. They look for anything that might help them harm your company: names, phone numbers, or project information. At other times, they use search engines to find information about your company. Information leaks onto the public Internet in many ways. It is truly sobering to do a search on the phrase “confidential, do not distribute” and see how many documents are out there. (We do not, of course, encourage you to actually read them.)

Suppose that while dumpster diving a malicious person finds a memo on company letterhead about a mysterious Project Zed in the R&D department. Attached to it is a list of people involved in the project and their phone numbers. The perpetrator may use this starting material and telephone tactics to get as much information as possible out of unsuspecting employees. These people can sound very smooth on the phone and can succeed if employees aren’t on their toes. The caller may start by claiming to be new on Project Zed, working with [insert the name of someone listed on the memo], and trying to find out how to have an account created, get details for remote access, and so on. Once the account is created, the person can walk right into your systems. The moral of this story is to tell people to be careful about what they say on the phone, and to shred documents that may contain sensitive information—even if they feel silly doing it!

---

### **A Failed Attempt at Social Engineering**

A naive young SA once received a phone call from someone claiming to be from the local police. The guy claimed that he was trying to understand how local companies secured their computer networks as part of a community assistance program. He asked several pointed questions, which the SA happily answered.

Over the next few days, other people at the company received phone calls from the same guy, this time claiming that he was a new member of their computer security group. He certainly sounded knowledgeable about the system. Luckily, one woman tried to verify his identity; when she couldn’t, she alerted the SA team’s manager. As a result, an executive warned everyone in the company that a scam was afoot, that no one

should reveal sensitive information over the phone, and that any unusual requests for sensitive information should be reported to a manager. These actions stopped the guy in his tracks.

If the perpetrator had continued, he might have used his techniques to gain access to the corporate network. For example, when he claimed to be from the security group, he sounded authentic because he had learned so much about the company's security system from the naive SA. Eventually he could have collected enough small bits of information to leverage them into full system access.

---

No matter how you feel about a policy, your duty is to follow the requests from corporate security. If you feel uncomfortable with those requests, take the issue to your manager; don't take the situation into your own hands.

---

### **Printer Log Panic**

A young SA who ran the print system for a large company was contacted by corporate security. As part of an investigation, security wanted logs related to what had been printed on a particular color printer. This printer was in the SA's building, which meant that he might know the suspect.

The SA panicked. He collected all the logs from that printer and copied them to his own computer at home. Next, he deleted the logs at work. Finally, he sought advice from two friends. "Someone could be fired! What should I do?" They both gave him the same advice: To avoid getting fired himself, he should restore the logs and give corporate security what had been requested. By hiding evidence, he had endangered his own position and made himself look like an accomplice to the accused.

---

## **47.7 Setting Expectations on Privacy and Monitoring**

Establishing a policy on privacy and monitoring is a fundamental ethical issue. This section emphasizes the need to remind the customers time and time again about this policy and its implications.

Setting employee expectations on privacy is important because putting people in situations in which they don't know the laws governing them is unfair. Punishing people for violating a rule they've never been told about is abusive.

There are many ways to set expectations. When hired, employees should be required to sign a statement that they have read the privacy and monitoring

guidelines. Companies can also require employees to sign these statements annually. To avoid confusion about whether a policy has changed or is merely being reprinted as a reminder, include the date when it was last revised.

Having a summary of the policy, or even the sentence “All sessions are open to monitoring,” displayed at every login screen is more effective than having a long policy stored on a web server that nobody visits.

Allowing employees to remain uninformed about privacy policies can be dangerous for business reasons. System users who do not realize which risks their actions involve cannot manage those risks. Suppose that customers discuss proprietary business details via email, which they think is secure. If it is not, the information could be leaked. Because they are uninformed, they may expose the company to unnecessary risk.

In the financial community, email is regularly monitored for SEC violations, such as insider trading. The threat of monitoring may be enough to prevent illegal exchanges of insider information via email. Of course, it also could simply push insider trading back to channels that are more difficult to monitor, such as the telephone. However, that decision is for the SEC to make, not you.

E-commerce sites, and any site doing international business, must be concerned with privacy laws, as they vary from country to country. For example, if you do business with EU citizens, there are strict regulations on how you must protect private information. U.S. laws add similar responsibilities.

Setting expectations also protects SAs' reputations because a lack of information will result in customers assuming the worst. Tom once had a customer who had previously been at a site that fired its SA for reading other people's email. The customer now assumed that all SAs read other people's email. Other customers believe that email is somehow magically private in all situations and will take unreasonable risks, such as sending salary data via email.

Removable media represent an easy way for data to leave company premises, or be lost or stolen during legitimate transport, so policies should address these issues as well. For example, it is common practice for salespeople to retain a copy of their address book and contacts list when they leave a company despite policies prohibiting it. Salespeople will tell you “everyone does it.” You probably can't stop them. If your policy sets different expectations, make sure that management is willing to stand behind their enforcement. Sites with strict confidentiality requirements may implement technical precautions to prevent data leakage in addition to their policies.

---

## **Email Forwarding**

A company had a liberal policy permitting email to ex-employees to be forwarded to their new email addresses. The policy created problems because proprietary business information was often bulk-emailed to lists that hadn't been updated to remove terminated employees. Most employees weren't aware of the risk of information leaks until an incident occurred.

A better policy is to set up an auto-reply system that sends a notification that the message will not be delivered.

---

## **47.8 Being Told to Do Something Illegal/Unethical**

No chapter on ethics would be complete without a discussion of what to do if your manager tells you to do something illegal, unethical, or against company rules. We hope that you will never need the information in this section, but it is better to be prepared with an understanding of the issues than to be facing them cold.

The most important thing to remember in this situation is to maintain a record of events. Keep logs of when such requests are made, when related phone calls happen, which commands you type to execute such requests, and so on. Logs are your friend. We recommend a simple process:

1. Verify the request—maybe you heard it wrong.
2. Verify that it's illegal or against company policy—check the written policy or ask someone for advice. Perhaps there is an ombudsperson you can talk with confidentially and without giving too many specifics.
3. If the request is against policy, assert yourself politely and reject the request explicitly.

If the manager persists, you can go along with it, go to a higher authority, or both. Highly regulated industries, such as financial institutions, have clearly established guidelines for what to do next. Even in a small firm, you have the option of approaching the HR or legal department and letting them know that you have a situation where you require guidance.

A useful technique is to ask that the request be made in writing or via email. That gives you a paper trail and requires the person to restate the request. If the request was an innocent request that was misinterpreted, seeing the request in email may clarify the situation.

Someone who is requesting something unethical and knows it won't put it in

writing. This can put an end to the request. However, asking that a request be put in writing without sounding confrontational is difficult. It can even sound a lot like insubordination. Instead, you can ask, “Can you email me a reminder so I’ll see the request after lunch?” If the person doesn’t, you can say that you need the email to be sure that you understand the request. Finally, you have to put your cards on the table: “Either I’m misunderstanding the request, or you are asking me to do something that I’m uncomfortable doing.” Then suggest that you add someone to the conversation, such as your manager, your manager’s manager, or someone else directly involved.

Let’s follow this through with a fictional example: Department head Bob asks you to read department head Alice’s email to see whether her department is planning on canceling a project on which Bob’s department relies. A good response would be to ask, in person, if you heard the request correctly: “My apologies. Could you say again what you want me to do?”

If the request is confirmed, verify that it’s against company policy by finding the appropriate paragraph in your organization’s privacy and monitoring guidelines. Gently verify that the person realizes that what is being asked is unethical by stating, “I’m not sure I heard you correctly” or “I think I must be confused or misunderstanding your request.” Explain what you think you are being asked to do, and point out that, as requested, it is against policy.

Use the policy to provide a gentle reminder to Bob. He might rationalize his request, explaining that Alice has canceled other commitments and that he’s only trying to help you because he knows that you also rely on this project.

At this point, you have a decision to make. You can stall and use the time to talk with an ombudsperson, corporate security, or Bob’s manager. You could go along with the request, but doing so would make you an accessory. Bob might make more requests of you, possibly pointing out that if you don’t comply, he’ll reveal the previous incident, claiming that you did it on your own initiative. He might also claim that if you do not comply, he’ll simply find someone else who will do it. This tactic is a very threatening way for someone to convince a person to do something.

Obviously, we cannot make this decision for you. However, we can give you this advice: When you aren’t sure whether you should do something, get the request in writing and keep a log of exactly what you do. Never act on verbal instructions that you find questionable. Even if you think that it might be okay, get it in writing. This is critical not only to cover yourself, but also to assist the person making the request to be sure that he or she really wants it done.



Someone who is not willing to put a request in writing is not willing to take responsibility for the action. Your log should note the time, the request, who made it, why it was done, and what was done. Also note anything unusual about the request. Not creating a log is something that people regret when it is too late. Automatically timestamped logs provide better accountability and take the tedium out of record keeping.

## **47.9 Observing Illegal Activity**

Often SAs are in a position to observe illegal behavior committed by the organization. It is an SA's responsibility to report such activity. Common examples include observing insider trading in a financial market, unauthorized public surveillance, or organizations making public statements that are known to be untrue within the company.

Reporting such activity is called being a whistleblower. The ethics of whistleblowing are complex and controversial. In fact, there is more to say on this topic than we can cover in this book. Some countries protect whistleblowers who follow carefully defined processes. However, being a whistleblower usually means years of legal wrangling and often ends the person's career. So tread carefully and, if you choose to take a stand, follow your organization's reporting process carefully. If that fails and you still see legal malfeasance, seek legal advice about involving regulatory or external law enforcement.

To stay up-to-date on whistleblower laws, check out the NPR show *On the Media*. The November 30, 2012 and February 13, 2015 episodes are particularly insightful ([Gladstone & Garfield 2012](#), [2015](#)).

## **47.10 Summary**

Ethics are the principles of conduct that govern people's actions. Many people find the word *ethics* to be scary and vague. This chapter laid down some guiding principles for you to consider, but left things open-ended enough that you can make your own choices.

The System Administrators' Code of Ethics seeks to enhance the professionalism and image of SAs by establishing a standard for conduct. The policies that an organization should create include a network/computer user code of conduct, a privileged-access code of conduct, a copyright-adherence policy, and a policy on working with law enforcement. Informed consent decrees that we have a monitoring and privacy policy that is clearly communicated to all customers. Unless all these policies include a penalty for violations and are



enforced consistently, they have no teeth.

Having thought about potential ethically challenging situations greatly prepares you for dealing with such situations when they arise. Try to think about potential ethical dilemmas you might encounter and what you would do in those instances. This might be a good thing to discuss occasionally with your team, perhaps at staff meetings or during lunch. This should be done with your manager present so that you can develop an understanding of the official policy.

If you learned one thing in this chapter, we hope it was this: When you are in a gray area, the best way to protect yourself is to keep logs. Ask for the request in writing to create a log of the request, log when you receive phone calls, log what you are asked to do, and log what you do. Log everything!

## **Exercises**

1. Describe an experience in which the System Administrators' Code of Ethics or the informed-consent rule would (or did) affect your actions.
2. Give examples of instances in which you or your team enforced policies that customers were not aware of. In particular, consider conditions of acceptable use, sharing and availability of common resources, and system monitoring. How would you improve your performance in these areas?
3. Think of an incident in which you or another SA were not acting in a fully professional manner. How would you have handled the incident with the System Administrators' Code of Ethics in mind?
4. Of the policies discussed in this chapter, which does your site have? If you are at a large site with corporate policies, are there policies specific to your division?
5. Describe the policies detailed in the previous question as being easygoing or strict. Give examples. How would you change things, and why?
6. Ask three customers whether they know where to find any of the policy documents mentioned in this chapter.
7. Suppose that you were debugging a problem and as a result accidentally learned that a co-worker was dealing drugs from the office. What would you do? What if the person was instead planning to sabotage the company? Stealing office supplies? Stealing pieces of equipment for resale on an Internet auction site? Having an affair with the boss? What if the person was not your co-worker but a high-level executive?
8. How long do you keep various logs on your system? If you had been

involved in the printing log anecdote in [Section 47.6](#), how long would you now retain such logs? Why?

- 9.** Suppose that you work at a web-based e-commerce site. An engineer who doesn't have the skill or patience to properly test his code in the development environment asks you to let him look at the logs on a production server. Soon he is pressuring you to let him temporarily change a logging parameter so he gets more information. Before you know it, you have an engineer doing development on the live production hosts. How would you handle this situation? Realistically, how could you have prevented it?
- 10.** An executive asks you to allocate additional disk space for someone. You respond that you don't have the space but are told, "Make space—this person is important." Soon the same request is made for another person. You are told, "You did it for the last person. This one is just as important." How would you handle this situation? Realistically, how could you have prevented it?
- 11.** A person who is not an SA has privileged access to her workstation because the software she uses requires it. Friends ask for accounts on her system. It's against policy, but she does it anyway. One of her friends starts engaging in illegal activity on the workstation. How would you handle this situation? What if the employee who violated policy was above you in management? A peer? Realistically, how could you have prevented the problem?

## Chapter 48. Organizational Structures

How an SA team is structured, and how this structure fits into the larger organization, are major factors in the success or failure of the team. This chapter examines some of the issues that every site should consider in building the system administration team and covers what is practiced in a variety of companies. It concludes with some sample SA organizational structures for various sites.

Communication is an area that can be greatly influenced by the organizational structure of both the SA team and the company in general. The structure of the organization defines the primary communication paths among the SAs, as well as between the SAs and their customers. Both sets of communication are key to the SA team's success. The SAs at a site need to cooperate so that they can build a solid, coherent computing infrastructure for the rest of the company to use. However, they must do so in a way that meets the needs of the customers and provides them with solid support and good customer service.

Management and individuals should work hard to avoid an us-versus-them attitude, regardless of how the organization is structured. Some organizational structures may foster that attitude more than others, but poor communication channels are always at the heart of such problems.

Creating an effective system administration organization that is free of conflicts both internally and with the customer base is challenging. Sizing and funding the SA function appropriately so that the team can provide good service levels without seeming to be a financial burden to the company is another tricky area. We also examine the effects the management chain can have on that issue.

The ideal SA team is one that can provide the right level of service at the lowest possible cost. Part of providing good service to your company is keeping your costs as low as possible without adversely affecting service levels. To do that, you need to have the right SAs with the right set of skills doing the right jobs. Throwing more people into the SA team doesn't help as much as getting the right people into the SA team. A good SA team has a comprehensive set of technical skills and is staffed with people who communicate and work well with others.

Small SA groups need well-rounded SAs with broad skill sets. Larger SA groups need to be divided into functional areas. We identify which functions should be provided by a central group and which are better served by small

distributed teams of SAs. We explain the mechanisms of the centralized versus decentralized models for system administration and describe the advantages and disadvantages of each approach.

## **48.1 Sizing**

Making your SA team the correct size is difficult. If it is too small, it will be ineffective, and the rest of the company will suffer through unreliable infrastructure and poor customer service. If the team is too large, the company will incur unnecessary costs, and communication among the SAs will be more difficult. In practice, SA teams are more often understaffed. It is unusual, though not unheard of, to see an overstaffed SA team. Overstaffing typically is related to not having the right set of skills in the organization. If the SA team is having trouble supporting the customers and providing the required level of service and reliability, simply adding more people may not be the answer. Consider adding the missing skills through training or consultants before hiring new people.

When deciding on the size of the SA team, the management of the organization should take into account several factors: the number and variety of people and machines in the company, the complexity of the environment, the type of work that the company does, the service levels required by various groups, and how mission-critical the various computer services are. It is a good idea to survey the SAs to find out approximately how much time each is spending supporting the customers and machines in each group, as well as the central infrastructure machines.

Ideally, your trouble-ticket system should be able to give you this information for a given period quite quickly. If it cannot, you can have each SA fill out a short form, such as the one in [Figure 48.1](#), with approximate numbers. This information will provide a basis for deriving the SA team's growth rate that should keep the service levels approximately constant. It should also give you insight into areas that consume a lot of SA time and allow you to look for ways to reduce support overhead.

Fill in the *approximate* percentage of time spent on each category.  
Please make sure that they add up to 100 percent.

| Time spent on ...        | Percentage | Quantity supported      | Number |
|--------------------------|------------|-------------------------|--------|
| Customer/desktop support |            | Customers               |        |
| Customer server support  |            | Customer servers        |        |
| Infrastructure support   |            | Infrastructure machines |        |

Figure 48.1: Form for gathering data to predict growth rate

### Case Study: High Support Costs

When Synopsys did a survey of where the SAs were spending their time, the managers discovered that SAs were spending a lot of time supporting old, ailing equipment that also had high maintenance contract costs. Replacing the equipment with new, faster hardware would yield continuing savings in machine room space and labor. The managers used this information to persuade the group that owned the equipment to retire it and replace it with new machines. This enabled the SAs to use their time more effectively and to provide better customer service.

There is no magic customer-to-SA ratio that works for every company, because different customers have different needs. For example, a university campus may have 500 or 1,000 customers for every SA, because most of these customers do not use the machines all day, every day, are reasonably tolerant of small outages, and generally do not push the hardware to its limits. In contrast, high-tech businesses, such as hardware design and gene sequencing, require more from their IT services and may require ratios closer to 60:1 or even 20:1. In software development, the range is even wider: We've seen each SA supporting as many as 50 customers or as few as 5. A nontechnical corporate environment will require about as many SAs as a technical one but with more focus on helpdesk, user interface, and environment training. Regardless, all organizations should have at least two SAs or at a bare minimum should have some way of providing suitable cover for their one SA if that person is ill or on vacation.

Machines themselves also require support time, independent of explicit customer requests. Servers require regular backups, software and OS upgrades and patches, monitoring, and hardware upgrades and maintenance. Some of this can be optimized and automated through techniques discussed elsewhere in this

book, but there is still significant server maintenance time. Even if desktops are easily replaceable clones of each other, they also require support time, though it is minimal because you can simply swap out a broken machine.

In any reasonably large organization, some people will spend their time primarily maintaining infrastructure services, such as email, printing, the network, authentication, and name service. Companies that provide e-commerce or other critical web-based services to their customers will also require a team to maintain the relevant systems.

All these areas must be taken into account when sizing the SA organization. Gather real data from your organization to see where SAs are spending their time. Use the data to look for places where automation and processes can be improved and to find services or systems that you may no longer want to support. Define SLAs with your customers, and use those SLAs to help size the SA team appropriately.

## **48.2 Funding Models**

Money is at the center of everything in every business. How and by whom system administration is funded is central to the success or failure of the SA team.

The primary reason that the SA function is generally understaffed is that it is viewed as a cost center rather than as a profit center. Simply put, the SA function does not bring in money; it is overhead. To maximize profits, a business must minimize overhead costs, which generally leads to restricting the size and growth of the SA team.

---

### **Case Study: Controlling Costs**

A midsize software company growing by about 30 percent annually was trying to control costs, so it restricted the growth of the budget for the SA team. The management of the SA team knew that the team was underfunded and that there would be more problems in the future, but it needed a way to quantify this and to express it to the upper management of the company.

The SA group performed a study to determine where the budget was being spent and highlighted factors that it could not control, such as per-person and per-server support costs. The group could not control the number of people other groups hired or the number of servers other groups bought, so the SA group could not control its budget for those

costs. If the budget did not keep pace with those costs, service levels would drop.

The most significant factor was how maintenance contracts were handled. After the first year, maintenance contract fees for machines under contract were billed to the central SA group, not to the departments that bought and owned the machines. Based on past trends, the group calculated its budget growth rate and determined that in 5 years, the entire system administration budget would be consumed by maintenance contracts alone. There would be no money left even for salaries. Last one out, turn off the lights (and sign the maintenance contract)!

Once the SA management was able to quantify and explain the group's budget problems, the CFO and his team devised a new funding model to fix the problems by making each department responsible for the system administration costs that it incurred.

You must be able to explain and justify the money that is spent on system administration if your team is to avoid being underfunded.

---

It is difficult to show how the SA team is saving the company money when everything is running smoothly. Unfortunately, it is easier to demonstrate where the company is losing money when system administration is understaffed and the infrastructure and support have deteriorated to the point that people working in the profit centers are losing significant amounts of time to computer and network problems. If a company reaches this stage, however, it is almost impossible to recover completely. The SA team will have lost the trust and cooperation of the customer base and will have a very difficult time regaining it, no matter how well funded it becomes.

You want to avoid reaching this state, which means figuring out a funding model that works. You need to be able to answer the following questions: Who pays? How does it scale? And what do customers get for their money?

The design of the funding model also has an impact on the organizational structure, because the people who are paying typically want significant control. Generally, SAs either are paid directly by business units and report into the business units, or are centrally funded by the company and form their own business unit. These are the **decentralized** and **centralized** models, respectively. It is not uncommon to see companies switch from one model to the other and back again every few years, because both models have strengths and weaknesses.

When a company changes from one model to the other, it is always stressful for the SAs. It is important for management to have frank, open meetings with the SAs. They need to hear management acknowledge the strengths of the existing structure and the problems that the group will face in maintaining those strengths. The SAs also need to be told what the weaknesses with the current structure are and how the new structure should address those weaknesses. The SAs should be given a chance to voice their concerns, ask questions, and suggest solutions. The SAs may have valuable insights for management on how their existing strengths can be preserved. If the SAs are genuinely involved in the process, it has a much higher chance of success. Representatives from the customer groups also should be involved in the process. It needs to be a team effort if the reorganization is to succeed.

The primary motivation for the decentralized model is to give the individual departments better or more customized service through having a stronger relationship with their SAs and more control over the work that they do. The primary motivation for centralizing system administration is to control costs through tracking them centrally and then reducing them by eliminating redundancy and taking advantage of economies of scale.

When it moves to a central system administration organization, a company looks for standardization and reduced duplication of services. However, the individual departments will be sensitive about losing their control and highly customized services, more alert to the smallest failure or drop in performance after centralization, and slow to trust the central group. Rather than work with the central group to try and fix the problems, the various departments may even hire their own rogue SAs to provide the support they had in the past, defeating the purpose of the centralization process and hiding the true system administration costs the company is incurring.

Changing from one model to the other is difficult for both the SAs and their customers. It is much better to get it right the first time or to work on incremental improvements, instead of hoping that a radical shift will fix all the problems without introducing new ones.

Funding of the SA team should be decentralized to a large degree, or it will become a black hole on the books into which vast sums of money seem to disappear. Decentralizing the funding can make the business units aware of the cost of maintaining old equipment and of other time sinks. It can also enable each business unit to still control its level of support and have a different level than other business units. A unit wanting better support should encourage its assigned SA to automate tasks or put forward the funding to hire more SAs.



However, when a business unit has only one SA, doubling that number by hiring another SA may seem a prohibitively large jump.

Given a time analysis of the SAs' work and any predefined SLAs, it is possible to produce a funding model in which each business unit pays a per-person and per-server fee, based on its chosen service level. This fee incorporates the infrastructure cost needed to support those people and machines, as well as the direct costs. Such an approach decentralizes the cost and has an added benefit that the business unit does not have to increase its SA head count by whole units. It does, however, require rigorous procedures to ensure that groups that are paying for higher service levels receive what they pay for.

Ideally, the beneficiaries of the services should pay directly for the services they receive. Systems based around a "tax" are open to abuse, with people trying to get as much as possible out of the services they are paying for, which can ultimately increase costs. However, cost tracking and billing can add so much overhead that it is cheaper to endure a little service abuse. A hybrid method, in which charges are rolled up to a higher level or groups exceeding certain limits incur additional charges, may be workable. For example, you might divide the cost of providing remote access proportionately across divisions rather than provide bills down to the level of single customers. Groups that exceed a predefined per-person level might also be charged for the excess.

Naturally, for budget planning and control reasons, the managers want to know in advance what the costs will be, or at least have a good estimate. That way, they can make sure that they don't run over budget by having unexpectedly high system administration costs. One way to do this is with monthly billing reports rather than an end-of-year surprise.

### **48.3 Management Chain's Influence**

The management chain can have considerable influence on how the system administration organization is run. Particularly in fast-paced companies, IT may come under the chief technical officer (CTO), who is also in charge of the engineering and research and development organizations. In other organizations, system administration is grouped with the facilities function and reports through the chief operating officer (COO) or the chief financial officer (CFO). These differences have important implications. If your CIO reports to your CTO, the company may view IT as something to invest in to increase profits. If your CIO reports to the CFO, the company may view IT as a cost that must be reduced.

When the system administration function reports through the CTO or the engineering organization, there are several beneficial effects and some potential problems. The most demanding customers are typically in that organization, so they have a closer relationship with the SAs. The group generally is quite well funded because it is part of a profit center that can directly see the results of its investment in system administration. However, other parts of the company may suffer because the people setting the priorities for the SAs will be biased toward the projects for the engineering group. As the company grows, the engineering function will likely be split into several business units, each with its own vice president. By this time, the system administration function either will be split into many groups supporting different business units or will report to some other part of the company because it will not be a part of a single “engineering” hierarchy.

In contrast, reporting through the COO or CFO means that the system administration function is a cost center and tends to receive less money. The people to whom the SA team reports usually have only the vaguest understanding of what the group does and which costs are involved. However, the COO or CFO typically has a broader view of the company as a whole and so will usually be more even-handed in allocating system administration resources. This reporting structure benefits from a strong management team that can communicate well with upper management to explain the budget, responsibilities, and priorities of the team. It can be advantageous to report through the CFO, because if the budget requirements of the SA team can be properly explained and justified, the CFO is in a position to determine the best way to have the company pay for the group.

Similarly, if SA groups report directly into the business units that fund them, the business units will usually invest as much into IT as they need, though quality may be uneven across the company.

Every reporting structure has strengths and weaknesses. There is no one right answer for every organization. The strengths and weaknesses also depend on the views and personalities of the people involved.

---

### **Friends in High Places**

Although being close to the CTO is often better in highly innovative companies, Tom experienced the opposite once at a company that was not as technically oriented. When Tom joined, he was concerned that he would be reporting to the COO, not the CTO, where he was usually more comfortable.

At this company, the COO was responsible for the production system that made the company money: Imagine a high-tech assembly line, but instead of manufacturing autos or other hard goods, the assembly line did financial transactions for other companies on a tight monthly schedule. The CTO office was a newly created department chartered with introducing some new innovations into the company. However, the CTO office hadn't yet established credibility.

Being part of the COO's organization turned out to be the best place for Tom's IT department. The COO held the most sway in the company because her part of the company was making the money that fueled the entire organization. As part of her organization, the IT group was able to work as a team with the COO to not only get proper funding for all the upgrades that the production part of the company needed, but also influence the CTO's organization by playing the role of customer of what the CTO was creating.

When the COO wanted something, she had the credibility to get the resources. When Tom wanted something, he had the COO's ear. When the CTO and Tom disagreed, he had the ear of the person who fundamentally controlled the funding.

---

## 48.4 Skill Selection

When building an SA team, the hiring managers need to assemble a well-rounded team with a variety of skill sets and roles. The various roles that SAs can take on are discussed in more detail in [Appendix B](#), “[The Many Roles of a System Administrator](#).”

The duties of the SAs can be divided into four primary categories. The first is to provide maintenance and customer support, which includes the helpdesk ([Chapter 27](#), “[Customer Support](#)”), server maintenance, second-tier helpdesk support, and whichever particular customers got specialized support on a day-to-day basis.

The second category is deployment of new services. These implementers should be dedicated to their current projects so that they are not affected by urgent customer requests that can push project completion dates back.

Third is the design of new service architectures. The design team is composed of systems architects who investigate new technologies and design and prototype new services for the customers and other SAs. The design team must keep in

touch with customers' needs through the other SA teams and is responsible for making sure that new services are planned and built in advance of when the customers need them.

If you have separate teams of SAs responsible for networks, databases, security, and so on, each of these groups must have staff who cover the range of categories that SAs encompass. There must be people to design, deploy, and maintain each supported area.

Finally, an SA team benefits greatly from senior generalists who understand, in depth, how all or most of the components work and interact. These SAs are able to solve complex, end-to-end problems whose resolutions may elude SAs who specialize in only one or two areas. The senior generalists are often referred to as integrators because they can integrate various technologies effectively to produce the best possible systems and services.

Overlap often occurs between the categories, particularly in smaller companies in which one or two SAs need to fulfill all roles to some degree. In larger companies, the roles are typically separated. Junior SAs are often hired to work at the helpdesk. As they gain experience, they move into second-tier support positions and later into deployment roles, day-to-day business unit support roles, or infrastructure maintenance roles. The most senior SAs typically fill the design and senior generalist roles.

In larger companies, finding ways to give the SAs a chance to spend some time in other roles, working with other teams, gives them growth and mentoring opportunities. This practice also educates the rest of the team when the returning SAs share their experiences and insights.

Rotating all the SAs through the helpdesk can give them valuable insight into the most common problems, some of which they may be able to fix permanently. It can also provide training opportunities for the junior SAs and faster call resolution for some more complicated calls. Helpdesk personnel can be rotated out of the helpdesk to give them a break and to enable them to spend some time being mentored through more senior work with customers or implementers.

Similarly, having implementers become involved in direct customer support should give them feedback on what customers need and how well their services are performing. Involving the implementers in design work should provide them with extra challenges and can help get their feedback to the designers.

SAs in roles that do not normally require much customer contact, such as server maintenance, design, and implementation roles, can lose touch with the customers' needs. Rotating them through roles that require a lot of customer

contact can keep them in touch with the company's direction.

---

### **When Things Are Dire, Hire Higher**

Being able to hire a mixture of skills and skill levels is a luxury for sites that are doing fairly well. When things are going well, it is possible to hire junior SAs who will be trained and groomed by the senior SAs. Being able to promote from within is less costly than recruiting outsiders, rewards the best workers, and retains institutional knowledge. Promoting from within lets you hire junior people to back-fill vacated positions.

However, if an SA team is not doing well—systems are crashing regularly, customers are dissatisfied, morale is low, and so on—a different tactic is required. When things aren't going well, one needs to hire reform-oriented people. Highly skilled people with experience at well-run sites are more likely to be reformers.

When reforming a failing organization, or even upgrading an organization that is performing adequately but needs improvement, start by hiring a few senior-level people who work well together. As things start to stabilize, give them the freedom to reorganize the team to use the more junior members as their experience dictates, or call for new junior members to be hired as described in the anecdote in [Section 1.5](#). Good people know good people; their professional contacts will help find the right people.

In other words, “[When things are dire, hire higher.](#)”

---

## **48.5 Infrastructure Teams**

When a company grows, it should have people dedicated to infrastructure support. An infrastructure team will look after centralized services, such as authentication, printing, email, name service, calendar service, networking, remote access, directory services, and security. This team also will be generally responsible for automated services for the SAs who are providing customer support, such as the automatic loading, configuration, and patching of new machines (as described in [Chapter 7](#), “[Workstation Software Life Cycle](#).”)

The infrastructure team must be a cohesive unit, even if it is spread over multiple locations. The company's infrastructure should be consistent and interoperable among all the sites. If different groups are running different pieces of the infrastructure, they may fail to agree on the protocols and interfaces

between the components, with adverse results.

---

### **Case Study: Distributed Network Support**

A large multinational computer manufacturing company implemented distributed management of networks and computers. Responsibilities were split among IT groups: The central IT department handled the WAN, remote sites had a local SA, and each department at headquarters had its own IT group to handle system and network administration. At the time, the remote administration tools we take for granted today were not available.

In the early days, with 20 sites, it seemed simple enough to merely give each site a subdomain and a class C network, and provide the SAs with tools to easily generate DNS zone files. However, when the company had 200 sites with different administrators wanting to manage their subdomains slightly differently, whenever an administrator left the company, the SA team had a training problem. In addition, many disputes went unresolved because, in some cases, the management chains of disputing sides intersected only at the CEO level. No one could reasonably be expected to arbitrate and decide what should be done.

Having so many subdomains also increased the amount of work the SAs had to do whenever someone moved to a different department. Mailboxes had to be moved; mail aliases, internal mailing lists, and NIS maps had to be changed; the modem pool authentication servers needed to be updated; and so on. With a proper, centrally managed system, only the mailbox might have to be moved, with a single corresponding mail alias change. But with so many changes needed, things were inevitably missed and mistakes were made.

The lack of a central system administration management structure was also expensive for the company in other ways: The numerous system administration groups were unable to agree on a network architecture or even on which network hardware to use. Overlapping duplicated networks were built with a variety of hardware. Incompatibilities among the hardware resulted in some communications being dropped. The network that resulted from this unarbitrated, decentralized process was unreliable and unsupportable, but each department continued to fund its own administration team and adamantly defend its territory.

Many years later, when all the tools for centralizing network

management, including namespace control, network addressing, and many other aspects of site maintenance became available, the company still used its outdated and ineffective distributed administration model.

---

Each piece of infrastructure has its own particular way in which it must be built and supported as a coherent unit. All infrastructure services fall into disarray when different parts of them are managed by different groups that do not cooperate with one another. Organizationally, this means that it is advisable to create a centralized infrastructure group and plan for its growth as needed. When other regions need their own infrastructure SAs, those SAs should report to the central infrastructure managers rather than to regional SA managers.

A very user-visible example of this is email, which is part of the company's interface to its customers and to the rest of the world. A consistent look and feel—one email format used consistently throughout the entire company, hiding server names, and so on—gives the company a professional image. Having a unified email architecture benefits SAs by making this system easier to debug and administer. Dedicated choke points where email enters and exits the company result in better email security, as filtering can be more comprehensive. All this requires cooperation between infrastructure teams across the globe.

Modern computing infrastructures rely heavily on networks and network services. If the network or network services architecture is badly designed or managed, it increases SA workload, cost, and customer dissatisfaction. If each site is permitted to be *creative*, the result is a management nightmare. Network design and implementation must be a cooperative effort; where it cannot be, well-defined demarcation lines must exist between the zones of the company that have internally consistent designs.

The security of the company as a whole is only as strong as the weakest link. If different groups have different standards and policies, it will be impossible to know how good the company's security really is.

## **48.6 Customer Support**

Customer support is a key part of the system administration function. Customer support means making sure that customers can work effectively. That, in turn, means being part of the solution for all their computing needs. Customer support generally works best in a more distributed model, in contrast to infrastructure support.

Having dedicated customer support personnel helps align your response time



with your customers' expectations. As discussed in [Section 49.1.3](#), having personnel dedicated to quickly responding to small requests and passing larger requests on to back-office personnel matches response time with customer expectations.

Customers like to know who their support people are. Customers build a relationship with their local SA and become familiar with that person's way of working and communication style. They feel more comfortable in asking that person for help. They can be more confident that their SA will know the context of what they are doing and which services the department uses and how, and that the SA will know the appropriate assumptions to be made about the customer and the environment.

We mentioned earlier that choosing between centralized and decentralized models for the system administration organization has an impact on customer support and the relationships both between the SAs and between the customers and the SAs. In the decentralized model, where each department or group has its own small SA team, and communication with the customers is typically strong, but communication between the SAs is generally weak. The customers will have greater familiarity with their SAs but will ultimately suffer because of the lack of communication between the SAs. For example, if the SAs' efforts are not well coordinated, many tasks may not be automated; there will not be consistent machines, OSs, patch levels, and applications across the company; and some services that should be centralized may be implemented multiple times. Collectively, these factors will result in inefficiency and lack of reliability that will ultimately be detrimental to the service the customers receive.

In the centralized model, communication between the SAs is generally strong and communication with the customer base is weak. The centralized model can lead to an us-versus-them attitude on both sides. That potential problem is balanced by a generally stronger infrastructure and a more consistent and robust site. However, customers will probably feel that they are not getting the attention they need, and projects for individual departments may be delayed unacceptably in favor of infrastructure projects or projects for other departments. Each department will suffer from not having its own SA paying attention to its specific needs.

A centralized support model can result in customers talking to a different person each time they place a support call. Because the SAs all support so many people, the SA who responds to a support call may not know the department well and probably won't know the customer. This is not a support model that satisfies customers. A team of at most five SAs should support a customer group.



Customers want to know their SAs and have confidence that the SAs are familiar with the department's setup and requirements. If a customer-visible SA team is larger than about five people, divide the team so that each half of the team supports half of that customer group. Another approach is to look at ways to reduce the number of SAs who provide direct customer support to that group by having some SAs work behind the scenes.

Some companies use hybrid models, in which each department or business unit has some dedicated SAs who report to the centralized system administration organization. This model is not perfect either, because some business unit SAs may be more vocal than others, causing some departments' needs not to be addressed properly by the automation and standardization performed centrally on behalf of the business units. Hybrid models have a lot of potential, but they can also introduce the worst features of both centralized and decentralized models.

Whichever model you use, be aware of its weaknesses and do what you can to mitigate them.

## **48.7 Helpdesk**

The first tier of a helpdesk function is the one that works best when centralized. This group receives initial problem reports and resolves basic issues, but otherwise dispatches the request to the appropriate area. Customers want one phone number, email address, and web page for support requests. Customers don't want to know which particular department owns a given problem, nor do they want to risk their request getting lost between departments because the one that received it doesn't know who should resolve it.

Large companies need well-coordinated regional helpdesks, with each reasonably large site having its own. With multiple helpdesks, escalation procedures and call handoff procedures become critical because some calls will cross regional boundaries. Large companies need to find the right balance between a tightly controlled centralized helpdesk that can handle all calls and a series of smaller distributed helpdesks that cannot handle all the calls but can put a friendlier, more personal face on the system administration organization. [Chapter 27, "Customer Support,"](#) discusses creation and management of helpdesks in detail.

## **48.8 Outsourcing**

Outsourcing the system administration function can seem like an appealing

option to some companies, because system administration is not usually considered a core competency of the company. If system administration is not a key part of the company's business, the company may prefer to outsource the entire area to an outsource contract company. That way, the company has to negotiate only a contract and not worry about SA recruiting, compensation packages, retention, and all the other issues that come with having employees.

For some companies, outsourcing makes sense. If a company is fairly small and has basic computing needs, negotiating a contract is easier than trying to hire an SA when the company has no one who can evaluate the SA's skill level or performance. If not satisfied with the outsourcing company, the company can renegotiate the contract or take the more difficult route of replacing that company with a new one. This step is easier than terminating an employee.

Other sorts of companies should not outsource their IT functions, or should do so very cautiously. For e-commerce sites, IT and system administration are part of the core business. Companies that rely on highly available computer systems will find that outsourcing contracts becomes prohibitively expensive when their reliability needs are factored in. Sites dependent on cutting-edge technology often need to run their own systems as well, because outsourcing companies have their own technology standards, which are difficult for individual customers to influence. In complex environments, getting rid of a poorly performing outsourcing provider is more difficult. It is not easy to switch out one SA team for another without suffering significantly downgraded service and outages while the new team becomes familiar with the site.

Security is another sensitive subject in the area of outsourcing. If security is important to a company, particularly in the areas of protecting information and handling customer-confidential information, the legal department should require strict confidentiality agreements and assurances about which companies the security contractors work for before, during, and after their work at that company. If a security breach can result in loss of customer confidence or otherwise severely affect the company's revenue stream, the security function should be one of the company's core competencies. In-house security staff have a stake in the company's success and failure, whereas the outsourcing company's liability will be limited to its contracts.

In companies that have a revenue-generating Internet presence or an e-commerce site, the SA function of maintaining this site is one of the core functions of the company. As a result, most of these companies are unwilling to outsource this area of their business. They want the people responsible for maintaining high availability on their Internet sites to have a personal stake in

their success or failure. For companies that receive only a small amount of revenue from their Internet presence and do not otherwise rely on high-availability computer systems, staffing the SA team for round-the-clock coverage can be prohibitively expensive, with outsourcing proving a more financially attractive option.

Many companies outsource their Internet presence to some degree by putting their Internet service machines into a colocation (colo) center. The main advantage of a colo is that the service provider has redundant high-bandwidth Internet connections and highly redundant power and cooling facilities that are prohibitively expensive for most companies to afford on their own. Colocation has better economies of scale.

Some high-end outsourcing companies also support the machines and the applications running on them. Others might have someone available to power-cycle a given machine on request. As with other forms of outsourcing, it is important to have service levels defined in the contract, with financial penalties specified if they are not met. The service levels should include guaranteed bandwidth, response times on service calls, power and cooling, uptime, physical security, and network availability.

## **48.9 Consultants and Contractors**

It can be very useful to use consultants and contractors to help your team grow and build new services and infrastructure, while still maintaining service levels. Used in the right way, consultants and contractors can give in-house staff opportunities to expand their abilities and gain experience. Used in the wrong way, they can offend, dishearten, and alienate the SA team.

We distinguish between consultants and contractors based on their skill level and the work they perform. A consultant brings significant new and in-depth skills to the table, usually for a specific project. A contractor brings skills that the company may already have and performs tasks that the SA group already performs.

Consultants who are experts in their fields can be useful temporary additions to a system administration team. A consultant should be engaged for a specific project with which in-house SAs may not currently have experience, such as designing and building a new service that needs to scale rapidly to support the whole company.

A successful relationship with a consultant will involve the SAs, particularly architects and implementers, in the project on which the consultant is working.

The consultant should be willing to share ideas, brainstorm, and generally work with the team, but not dictate to them. A good consultant can bring necessary expertise and experience to a new endeavor, resulting in a better design and increased knowledge for the in-house SAs.

In contrast, it is not good for the morale and success of an SA team to bring in consultants for the new and interesting projects while the in-house SAs are stuck with the day-to-day support and maintenance. The new service will not be as well supported as it would be if the in-house SAs were involved and understood the design. It also may not take into account local quirks or be as well integrated with other systems as it would be if local knowledge were used in the design. The in-house SAs will not get the opportunity to grow and learn new skills and will become dissatisfied with their positions, resulting in high turnover and unsatisfactory levels of customer support.

If new projects need to be implemented but the in-house SAs do not have time to participate, contractors should be brought in to back-fill for the in-house SAs' day-to-day tasks, not necessarily to implement the new service. The project will usually be more successful in terms of budget, features, and ongoing support and maintenance if the in-house SAs are involved, rather than giving it entirely to an outside team. Helping the SAs participate in new projects by relieving their day-to-day workload will also lead to a stronger, happier SA team.

Rotating the in-house SAs who participate in the interesting projects so that all the SAs have a chance to be involved in a project also strengthens the team.

---

### **Contract One-Shot Tasks**

Installing commercial data backup/restore systems is usually quite difficult. It's a lot easier the third or fourth time through, but most SAs never get to that point. A new data backup system is installed every few years.

Given this fact of life, it can be much more cost-effective to hire someone to do the installation. Find someone who has installed that particular system three or four times already.

Most backup software companies have a professional services group that will do the installation for you. These groups are invaluable. They set things up properly and train existing staff on daily operational details.

In one situation, Tom saw an SA take 6 months to install such a system. He was constantly interrupted with other project work, which extended and delayed the project. The company was happy that it didn't pay the

\$10,000 that the professional services contract would have cost. Tom felt that the money would have been well spent.

Many other similar one-shot tasks may benefit from hiring someone with experience.

---

## **48.10 Sample Organizational Structures**

How do the various roles and responsibilities fit into companies of different sizes? How does being an e-commerce site change the organization of the SA function? What's different in academic or nonprofit organizations? This section describes what the system administration organization should look like in small, medium, and large companies; an e-commerce site; and a university/nonprofit organization. For these examples, a small company is typically between 20 and 200 employees; a medium-size company, between 200 and 1,000 employees; and a large company, more than 1,000 employees. These are approximations; a small company with dozens of locations has many of the same needs as a large company.

### **48.10.1 Small Company**

A small company will have one or two SAs, who are expected to cover all the bases between them. There will be no formal helpdesk, though requests should be funneled through a request-tracking system. The SAs will be involved in customer support and infrastructure maintenance. However, usually only one of the SAs will be involved in designing and implementing new services as they are required.

As a small company moves beyond 200 employees and starts growing into a midsize company, its system administration organization will be formed. This intermediate time is when the big decisions about how to organize and fund system administration need to be made by the company's senior management. The formal helpdesk will be established during the transitional stages and initially will be staffed by a rotation of SAs in customer-support roles.

### **48.10.2 Medium-Size Company**

In a midsize company, the SAs start to specialize a bit more. A helpdesk team with dedicated personnel should have been formed before the company reached the 1,000-employee mark. The helpdesk will be expected to solve reasonably complex problems. SAs in customer-support roles may still rotate through the helpdesk to augment the dedicated staff. Some SAs will specialize in a specific

OS; others will take on networking or security specialties, initially as part-time responsibilities and later, as the company grows, as full-time positions. The architects generally will also be implementers, and ideally should be capable of solving end-to-end problems involving several technologies. Customer-support personnel also can be included in the design and implementation projects by designating days for project work and days for customer-support work.

### **48.10.3 Large Company**

A large company will have a high degree of specialization among the SA team, a well-staffed helpdesk with a clearly defined second-tier support team for the more difficult problems, small customer-support teams dedicated to various departments or business units, and a central infrastructure support group and a central security team. A large company will have at least one architect for each technology area and teams of implementers for each area. The company may have regional SA organizations or SA organizations dedicated to subsidiaries or large business divisions. These SA organizations will need formal communication paths to coordinate their work and clearly defined areas of responsibility. They should all be covered by the same company policies, if possible.

### **48.10.4 E-commerce Site**

An e-commerce site is different from other sites in that it has two sets of computers, networks, and services that require different levels of availability and are governed by different sets of priorities. A problem with a revenue-generating system will always take precedence over an internal customer-support call because the former is directly linked to the revenue stream.

To avoid this conflict of interest, a company with a large e-commerce site needs two separate SA teams: one to support only the Internet presence, and the other to support only the corporate systems. (Such dual teams are often also required for Sarbanes-Oxley Act compliance.) There should be a clean separation between the equipment used for each entity, because they have different availability requirements and different functions. Sites where parts of the Internet service rely on parts of the corporate infrastructure inevitably run into problems. A clean separation is also more readily maintained if different SA groups are in charge of each area.

The composition of the corporate SA team depends on the size of the company. For the Internet service, the SA team's composition is somewhat different. It may have a helpdesk that is part of the customer-support

organization (a helpdesk that supports external customers of the company, as opposed to the internal customers of the corporate SA team). Although that helpdesk may have second-tier support to which calls are escalated, there will not be small internal customer-support teams that serve each department. Instead, there will be only SAs who are responsible for the support and maintenance of the Internet service, and architects and implementers who design and deploy enhancements to the service and scale the service to meet customer demand.

---

### **Google's Two IT Teams**

Google has two different IT teams. The systems operations team maintains the internal IT systems. The site reliability engineering team maintains the systems involved in the web properties. Initially the IT functions were handled by one team, but its division has enabled the SAs to better meet business demands and creates an environment in which there is less conflict over priorities.

---

## **48.10.5 Universities and Nonprofit Organizations**

It is important for nonprofit organizations to control ongoing costs. Typically, they have very limited budgets, so it is vital that all the money available for computing be used properly, with organizations and departments working together with everyone's best interests in mind. It is easier to control costs when services are centralized. Universities, however, often have small fiefdoms built around the funding that individual research groups or professors receive. It is in the best interests of the university or nonprofit organization to centralize as much as possible and to work as a team. To do so requires strong leadership from the head of the organization and good service from the central SA team.

---

### **Case Study: Use Limited Resources Wisely**

Here's an example of what can happen if there is no budget for centralized services. A university department had a limited amount of money to spend on equipment. Several research groups within the department received additional money for equipment from their research projects. Equipment that was retired from use in a research group was given to the department to help support the undergraduate and infrastructure needs. The department had some glaring needs that it had been unable to fund. The backup system was able to accommodate only the machines that were

used by the undergraduates; there was no backup service for research computers. The servers that provided email, printing, software depot, and home directory services for the department were two unreliable machines that had been designed to be desktop workstations and were seven years old.

However, there was no coordination within the department on how best to spend the money from research contracts. One professor had enough money from a research project to buy four or five high-end PCs with plenty of disk space, monitors, and suitable graphics cards. Instead, he decided to buy two of the most expensive PCs possible with superfluous but flashy hardware and additional processors that could not be used by the computational code that ran on the machine. Both PCs were given to one Ph.D. student, who already had a high-end computer. They were largely unused. No machines were handed down to the department, and no consideration was given to purchasing a backup server for the research community in the department.

Professors were entitled to spend their own money, and none of them were willing to spend it on something that would benefit people outside their group in addition to their own group. Instead, money was blatantly wasted, and the whole department suffered. Ultimately, the head of department was responsible for failing to motivate the professors in the department to work as a team.

---

## **48.11 Summary**

The size and shape of an IT organization affects how effectively it can meet the needs of the company, as well as its cost and efficiency. The size should be based on pre-project needs, not simple ratios based on the number of customers or machines.

A small system administration team does a little of everything. Larger organizations should split into teams specializing in infrastructure, customer support, internal application development, and so on.

Some system administration teams are funded centrally, through charge-backs; others are funded through a decentralized model. If the SA team is centrally funded, people will take it for granted. If it is decentralized, it may drive customers to optimize locally, not globally.

Where the system administration organization reports into executive



management can have considerable impact on its influence and funding. Reporting to the CTO may mean new technology is prioritized. Reporting to the CFO can result in a self-defeating emphasis on cost cutting.

Consultants and contractors can be used effectively as short-term resources to advance particular projects that would otherwise take a long time to complete. However, they should be used in such a way as to give the permanent SAs the opportunity to get involved in these interesting projects, allowing them a chance to learn and advance their skills. Using only short-term resources to build systems for the permanent SAs to support is not a recipe for long-term success.

## Exercises

1. Is your system administration team centralized or decentralized?
2. Are there parts of your system administration team that are decentralized but that might work more effectively if they were centralized? Which problems do you think centralizing those groups would solve? Which problems might it create?
3. Are there parts of your system administration team that are centralized but that might work more effectively if they were decentralized? Which problems do you think decentralizing those groups would solve? Which problems might it create?
4. How have consultants or contractors been employed *effectively* by your organization?
5. How have consultants or contractors been employed *ineffectively* by your organization? How would you change what was done to make the relationship succeed and make sure goals were met?
6. How is your system administration organization funded? What are the problems with your current funding model, and what are the benefits? Can you think of ways that it might be improved? Which data would you need to present to upper management to get your ideas approved?
7. What are the relationships among the various components of your system administration organization? Can you think of ways to get SAs involved in other areas periodically? Which advantages and disadvantages would the team experience from implementing that idea?

## Chapter 49. Perception and Visibility

When done correctly, system administration is like good theater: The audience sees a wonderful show and never realizes how many months of planning were required to create the show or how much backstage work was happening during the performance. The majority of the work required for any performance is invisible to the audience. SAs tend to work behind the scenes. Even so, the audience appreciates and values the show *more* by understanding what went into creating the production. The audience is more invested in the success of the theater if it feels some kind of relationship to its people.

**Perception** is how people see you; it is a measure of quality. **Visibility** is how much people see of you; it is a measure of quantity. This chapter is about how customers see you and how to improve their perception of you. You are a great person, and we hope that this chapter will help you shine.

Customers' perceptions of you are their reality of you. If you are working hard but aren't perceived that way, people will assume that you are not. If they do not know you exist, you don't exist. If they know that you exist but have a vacuum of information about what you are doing, they will assume the worst. That's reality.

Many SAs believe that if they do the technical part of their jobs well, they will be perceived well and will be visible. This is not true. Many SAs feel that real SAs are concerned only with technical problems and that perception and visibility aren't their job, or that this is what managers are for, or that they have no control over how people perceive them, or that this is all just a hunk of baloney. We hope to change your mind. To that end, we have tried to keep our examples as real and down-to-earth as possible.

### 49.1 Perception

We've already established that you are a quality individual. Do people perceive you accurately? This chapter deals with improving how you are perceived and establishing your positive visibility. Every interaction with others is an opportunity to improve how you are perceived. The first impression that you make with customers dominates all future interactions with them. You must have a positive attitude about the people you support, because they will perceive you based on your attitude.

Customers perceive the efficiency of the work you do not by how hard you

work, but by how soon their requests are completed. Therefore, we discuss a technique that lets you match your priorities to customer expectations of completion time. This chapter ends with a discussion of what we call being a system advocate—that is, being proactive in meeting the needs of customers.

You are responsible for whether you are perceived positively or negatively. Take responsibility for improving how you are perceived. Nobody else will do it for you.

The following sections provide the key elements every SA must master to achieve positive perception. Some of these are more appropriate for management to initiate, but others must be done on the personal level. Later in the chapter, we discuss techniques SAs can use to increase the amount of visibility they receive.

### **49.1.1 A Good First Impression**

It is important to make a good first impression with your customers. If you get off on the wrong foot, it is difficult to regain their trust. Conversely, if you make a good first impression, the occasional mistake will be looked at as a fluke.

Try to remember back to grade school, particularly the first day of the school year. The kid who got in trouble that day was assumed to be “the bad kid” for the rest of the year. That kid was watched closely, was rarely trusted, and never got the benefit of the doubt. If anything went wrong, he or she was accused. Other students were on their best behavior the first week of school. They went out of their way to be especially nice to the teacher. For the rest of the year, they could get away with occasional infractions or ask for and receive permission to bend the rules.

Think of the impression you make as a goodwill bank account. Every good thing you do goes into the account. You can survive one bad thing for every five good things in the account, if you are lucky. Making a good first impression starts that bank account with a positive balance.

### **Outward Appearance**

If your first interaction with a customer is at an appointment, you can do many things to make a good first impression: Be on time or early, polite, friendly, and willing to listen. Humans are visual beings, so appearance and facial expression are two things people notice first. Smile.

---

#### **Chromatically Different Hair**

Dressing in a way that gives a positive first impression means different

things at different companies. An SA we know wears baggy overalls and dyes her hair bright pink, the current rave fashion. She's quite a sight! For a while, she was an SA at a major company in Silicon Valley. When she was assigned to directly support a web farm for a small group of people, they were extremely respectful of and enthusiastic about her. One day, a peer told her that the reason she had gained such quick acceptance from the customers was that they figured that anyone who dressed that way and got away with it must be technically excellent.

---

Pay attention to what you wear. Being well dressed has different definitions at different companies—a suit is respected at some, mocked at others. Just because we are not as concerned about how we dress does not mean that others feel the same way.

---

### **The First Time I Met You**

Tom ran into someone he hadn't seen for several years. The person casually recalled, "I still remember the silly T-shirt you were wearing the first time I met you!" First impressions are lasting impressions. Every day is a day you can potentially make a first impression.

---

What you wear on your face is also important. A non-exasperated, unruffled exterior is the greatest asset an SA can have. Smile now and then, or people will remember you as "the grumpy one."

### **Staying Calm**

Don't yell during a disagreement or because you are frustrated with someone in an otherwise calm situation. Yelling at people is bad. Really bad. The kind of bad that gets people demoted, fired, blackballed, or worse. When overheated or frustrated, it is nearly impossible to find something polite to say—and if you do say something polite, it won't come out sounding that way. It is simply best to remove yourself from the situation. If you are so frustrated that you are about to yell at someone, announce that you have to go to the restroom. It always works. Excusing oneself to go to the restroom is socially acceptable, even if you are in an important meeting, on a phone call, or standing in front of a down server with the CEO breathing down your neck. It's a convenient reason to take a break and cool off. Everyone understands what it means and why taking immediate action is required.

## **First Impression for New Hires**

You have an existing installed base of reputation with your current customers, but it is extremely important to make a good first impression with each new person who is hired. Eventually, these “new people” will be the majority of your customers. Making a good first impression on new hires begins before their first day at work. You need to make sure that when they arrive, they will find their computers in their offices, configured, accounts created, and everything working properly.

On a person’s first day, he or she can be nervous and afraid of the new surroundings. Having everything set up gives that person a warm and invited feeling that affects the impression of not just the SAs but the entire company. The SAs have an opportunity to establish a reputation for being organized and competent, and having the customer’s needs in mind.

Businesses are realizing that a new hire’s first day sets the tone for the employee’s entire time with the company. A person arriving for the first day of work usually is highly motivated. To maintain this motivation, the person must be productive right away. Every day that the person is delayed from being productive reduces motivation. If you want high performance, make sure that new hires can get started right away.

---

### **S-l-o-w PC Delivery**

A programmer in New Jersey told us that when she started working at a large insurance company, she didn’t receive a computer for her first month. This may be acceptable for nontechnical jobs, but she was a programmer! Members of her group were curious about why this upset her, because this was the status quo for PC deployment in their division. She was paid to do nothing for the first month, and everyone considered this “normal.” This was a waste of company resources. The fact that fellow employees had grown accustomed to such slow response indicates that the SAs of this company had been doing an unimpressive job for a long time.

---

It is much better to use new PC deployment as a time to create positive visibility.

---

### **Good PC Delivery**

At New York Institute of Technology, new PCs are delivered to faculty

and staff in a way that generates a lot of positive visibility. The PCs arrive on a cart decorated with jingle bells.

Everyone knows that the sound of bells means that someone is getting a new PC. As the cart rolls down the hallway, a crowd gathers to see who the recipient is. Finally, the cart reaches its destination. Everyone watches as the PC is lifted off the cart and placed on the person's desk. The "ooooos" and "aaaahs" sound like gifts being opened during a baby shower. Sometimes, people even applaud. (Of course, the PC has been preloaded and tested and is ready to be plugged in and used. Rehearsal makes everything perfect.)

---

To ensure that people have what they need on their first day, you should establish a process with the cooperation of other administrative personnel, often outside of the system administration team. Administrative assistants usually have a checklist of arrangements to make for all new hires. It is key to make sure that computing needs are on that checklist: finding out which kind of computer the person needs, procuring the computer, arranging the network jacks, finding out what the person's preferred login name is, determining which internal mailing lists the person needs to be on, finding out which software is needed, getting the accounts created, and so on. If there is a standard desktop computer configuration, preconfigured machines should be on hand and ready to be deployed. Otherwise, there has to be a process whereby the new hire is contacted weeks in advance to arrange for the proper equipment to be ordered, installed, and tested.

On the employee's first day, the friendliest member of your SA team should visit the person to do some kind of in-person orientation, answer questions, and personally deliver a printed "welcome to our network" guide. The orientation process puts a face to the SA team. This is comforting to the customer. Customers don't like faceless organizations. It is easier to get angry at a person whom you have never met. The orientation creates a good first impression with new employees, and is an investment that pays off with an improved relationship in the future.

---

### **Orientation Sessions**

If the SAs don't visit new hires to give them an orientation, someone else will. One company believed that doing such an orientation was a waste of time for the SAs and assumed that it would be an annoyance to new employees. Instead, a peer employee would brief the person about how to

log in and during the process take time to bad-mouth the SA team or recount the last unplanned system outage. It was a long time before the SA team realized that this was happening and even longer before the SAs could turn the situation around and repair their reputation.

---

### 49.1.2 Attitude, Perception, and Customers

How people perceive you is directly related to the attitude you project. It is important to have a positive attitude, because people pick up on your attitude very quickly.

The number-one attitude problem among SAs is a blatant disrespect for the people whom they are hired to serve. It is surprising how often we remind SAs that their users are not “lusers” or “pests with requests.” They are the reason SAs have jobs. SAs are there to serve but, more important, to advocate for these people. The SAs and the computer users are all on the same team. Use the term “customers” rather than “users” to promote a more positive attitude.

It is very enlightening to work as a consultant; the customers are directly paying you for your SA work and will replace you if you don’t meet their needs. It helps you to realize how much better things work when you treat your “users” as customers and really listen to what they need.

Conversely, an SA can get into trouble if he or she adopts the attitude that “the customer is always right.” That’s going too far. Part of an SA’s job is to (politely) say no when appropriate. An SA can end up doing the customer’s job if he or she does *everything* the customer requests. Instead, the SA must remember to help the customers help themselves. It is a balancing act. The SA management needs to establish a clear definition of where to draw the line (see [Section 27.5](#)). It also helps to adopt an attitude of “just because I *can* do it doesn’t mean I *should*.” Teach customers how to do things themselves, provide documentation, and make sure that they become good customers—the kind that don’t ask time and time again how to do something. It is also the responsibility of an SA to politely reject requests that are against policy.

As an SA moves from customer support to higher-level roles, such as system architect, the customer relationship often gives way to a more collaborative relationship whereby the SA and the customer work together as a team. As this happens, you are trying to develop a “business partners” relationship. You work together to do what is best for the company and define a scope of work that delineates what SAs should do and what customers should do, but the customer relationship still should remain.

Another attitude problem SAs can develop is becoming frustrated by the fact that customers do nothing but bring them problems. SAs can develop resentment toward customers and want to avoid them or moan every time customers come to their offices. This is a bad thing. We hear such comments as, “Oh, great! Here comes another problem.” This is an ironic situation because your job as an SA is to fix problems. If you get frustrated by the constant flood of problems, maybe you need a vacation (see [Section 52.4](#)). A more positive attitude is to address every “problem” as a puzzle, one that is a fun challenge to solve.

Related to that situation is frustration from believing that all your customers are stupid, or that they always call the help line with dumb questions. But remember that they wouldn’t be calling if they knew the answer, and you wouldn’t have been hired if you didn’t know more about this than they do. Don’t think your customers are idiots who barely know anything about computers. They know enough to work for a company that is smart enough to hire someone like you who can answer their questions, and they know a lot more about their own areas of expertise than you do.

Create opportunities to interact with your customers in which they won’t be coming to you with problems. Take the initiative to visit them occasionally. We’ll have more suggestions later in this chapter.

---

### **Venting About Customers**

An SA lost the respect of his customers and was later removed after an incident in which he loudly complained about them in the company cafeteria. His complaint was that although they were brilliant in topics outside of computers, their approach to computers was idiotic. He complained about them to no end while eating lunch and explained that he felt that it was beneath him to have to help them. One mistake he made was using the word *idiots* (and worse) to describe them. Another mistake he made was to misunderstand the hand signals all his coworkers were giving him to try to explain that the people he was talking about and their director were sitting at the table behind him. As you can guess, this issue was resolved by management at a very high level.

If your customers frustrate you, complain in private to your manager, and work on constructive solutions. Don’t gripe about customers in public, and remember that email can be forwarded, chat windows can be seen by passersby, and conversations can be heard over cubicle walls. Venting is a much-needed form of stress release: Find the right forum for it.



---

You should adopt an enlightened attitude toward trouble reports. Requests from customers can be exciting challenges and opportunities to do a fantastic job, one of which you can be proud. When you integrate this attitude into your life, your customers will notice the difference in the service you provide. When replying to a customer's trouble ticket, end with a sincere note of thanks for reporting the problem: "Thank you for reporting this. It helped us fix the problem and find ways to prevent it in the future." It can make all the difference. Your attitude shows through in everything you do.

### **49.1.3 Aligning Priorities with Customer Expectations**

The way you prioritize your tasks influences how customers perceive your effectiveness. You can make customers a lot happier if your priorities match their expectations. (Thanks to Ralph Loura for this technique.)

Customers expect small things to happen quickly and big things to take a reasonable amount of time. How they define *big* and *small* is based on their perception of your job. For example, resetting a password, which is perceived as taking a minute or two, should happen quickly. Installing a new computer is perceived as a bigger process, and it is reasonable that it takes a day or two. When a critical server is down, customers expect you to not be working on anything but handling the emergency. Therefore, you can make customers much happier if you prioritize requests so that emergencies are handled first, then quick requests, followed by longer requests.

In a given day, you or your group may complete 100, 500, or 5,000,000 requests. If you do them in the order that they arrive, your customers will not be happy with your work, even though their requests were completed. Instead, you can do the same amount of work in a day but in a different order, and customers will be delighted because their expectations were matched.

A customer will be upset if told that his small request has to wait until you have completed a larger request, such as installing a new computer. Imagine how upset you would get if you had to wait for a one-page print job because someone was printing a 400-page report that was ahead of you in the queue. It's very similar. It is important, however, to make sure that big jobs are not postponed endlessly by small ones.

Such tasks as resetting a password also have this expectation because of the domino effect. Not being able to log in delays other tasks. A person who can't log in can't get other work done. However, replacing an old PC with a newer one is less likely to have this same impact, since the person already has a PC.

Quick requests are often easy to automate so they become self-service. Establish a web page that will do the task for the customer. For example, set up another way for people to identify themselves so that you can implement an automated password-reset system. Set up a web site for people to order additional workstation software, and link it to inventory, and to your automated software installation system so that it can be installed immediately. Sometimes, the solution is as simple as keeping the spare paper and toner cartridges near the printer so that the time to install them is not dominated by the time to walk to a distant supply room.

In addition to using this reordering technique on the personal level, you can use it throughout an organization. You can divide the SA team so that frontline support people perform requests that customers expect to see done quickly. Requests that will take more time can be passed on to second-tier personnel. Senior SAs can be in charge of larger projects, such as the creation of services. This division of labor enables you to ensure that your priorities are aligned with customer expectations and shelters people who are working on long-term projects from continual interruptions. This may sound like something only large SA teams can afford to do, but even a team of two SAs can benefit from this technique. One SA can shield the other from interruptions in the morning, and vice versa in the afternoon. This is called the **mutual-interruption shield** technique.

#### **49.1.4 The System Advocate**

Customers perceive SAs as being somewhere between a clerk who reactively performs menial tasks and an advocate who proactively solves their problems and lobbies for their needs. This section is about becoming an advocate, which is the better position to be in.

At one end of the spectrum is what we call a system clerk, who is reactive rather than proactive. The clerk is told what to do on a schedule and does not participate in planning his work. Sometimes, the budget for the clerk comes from nontechnical budgets. A system clerk might spend the day installing software, performing backups, creating accounts by manually entering the various commands required to do so, and so on.

At the other end of the spectrum is the system advocate, who is proactive about her customers' technical needs, advocates those needs to management, automates the clerical tasks that she is asked to do, and is involved in the planning process for projects that affect her. Earlier, we discussed having new customers' machines and accounts set up by the day they arrive. To achieve that

goal, the SAs must be involved in the hiring process. Becoming involved is the kind of proactive step that a system advocate would take.

Between the clerk and the advocate are an infinite number of gradations. Try to be conscious of where you are and what you can do to move toward being an advocate.

Becoming an advocate is a slow evolution. It starts with one proactive project. Being proactive is an investment that can have a big payoff. It consumes time now but saves time later. It can be difficult to find time for proactive work if you are having trouble keeping your head above water. Select one thing that you feel is achievable and would have a significant impact, or ask your manager for suggestions. If your manager believes in the potential impact, he or she should be willing to reallocate some of your time to the project.

It is better to be the advocate, for many reasons. It's better for your company because it means that you are aligning your priorities with those of your customers. It puts your focus on where you can best serve the people who rate and review you. It is better for you because it develops your own reputation as a can-do person. People with such reputations have better chances during promotion opportunities. When raises are being calculated, it certainly helps to have a reputation for being the most helpful person in the group.

A large SA team usually will have the entire spectrum from clerks to advocates. The clerks are an important part of the team. The clerk role is where SAs get started and learn from others; the clerks provide the advocates with useful assistance. Even the newest SA should adopt a proactive can-do attitude, however, and should work toward becoming an advocate. There is a big difference between a clerk being managed by SAs versus by customers. Being managed by SAs means being told the right way to do things and provides opportunities to learn and grow. A clerk being managed by customers doesn't have the chance to learn from experts in the field and can feel isolated.

The biggest benefit comes from adopting the can-do attitude across the entire SA team. The team becomes an active, positive force for change within the organization. It becomes valued as a whole. Many companies live and die on whether they have the right IT infrastructure to achieve their business goals. Be part of that infrastructure.

Following are some examples that illustrate the differences between how a clerk and an advocate would handle various situations.

## **Installing Software**

You would think that installing software is fairly straightforward, but many subprocesses are involved. A clerk, for example, might receive the software to be installed from a customer who has purchased it. The customer may have ordered it weeks ago and is anxiously awaiting its installation. Often, in this situation, something goes wrong. The customer didn't know that there is a network license server and has licensed it to the customer's own workstation instead. This breaks the strategy of having a few well-maintained and monitored license servers. More often, the installation fails because of something simple that takes a long time to repair. For example, the customer was expecting the software to be installed on a machine that is overburdened and shouldn't have new applications added to it, the license is to a host that is being decommissioned soon, or the system runs out of disk space. It can take a long time to resolve such obstacles before a second installation attempt can succeed.

While the clerk is self-congratulatory about accomplishing the installation after overcoming so many obstacles, the customer is complaining to the SA's boss about having to wait weeks until the software was installed. The customer has no understanding that, for example, installing additional disk space is something that must be planned. Nor does the customer understand that the addition of a new license server requires more monitoring and reliability planning, and that by not leveraging the planned infrastructure, future work will require more effort for the SA team. There is no accounting for the real cost of the software, which should have included the cost of disk storage, CPU capacity, and the fact that the installation had to be performed twice because the first attempt failed owing to lack of disk space. The multi-week installation effort did not match the customer's expectation of a quick install after receiving the media. As discussed in [Section 49.1.3](#), matching the customer's expectations is critical.

A system advocate would be in a position to make the process go more smoothly. The customer would have known to involve the SA in the process from the beginning. The advocate would have interviewed the customer to get a basic understanding of the purpose of the software and performed capacity planning to allocate the proper disk, CPU, and network resources. A timetable agreed to by all parties involved would have avoided misunderstandings. The purchase would have included the software—possibly ordered by the SA to ensure correct licensing—as well as any additional disk, CPU, or network capacity. The problems, such as lack of disk space, could have been resolved while waiting for the software to arrive.

Such planning matches the software with the appropriate disk, CPU, and network capacity and gives the customer a better understanding of the SA

processes and the SA a better understanding of the customer's needs. People value what they understand and devalue what they don't understand. Through this process, both sides value the other person.

### **Solving a Performance Problem**

A customer complains about slow system performance. A clerk might be told to upgrade the customer's machine to a faster network card because all customers assume that systems are always slow because of slow networks. Little is improved after the installation, and the customer is not happy. The problem has not been properly diagnosed. In fact, many times in this situation the performance gets worse. The PC now has a faster network card than the server and is flooding it with traffic. Upgrading the server is actually what would fix the problem.

An advocate would take a more active role by observing the problem—ideally, before the customer notices it if there is a good monitoring system in place—and using various diagnostic tools. Based on this information, the advocate proposes a solution. The advocate takes the time to explain the problem and the various potential solutions with enough detail so that advocate and customer can select the best solution together. The proposal is presented to management by the customer, who is now capable of explaining the issues. The SA is in the room to support the customer if he or she stumbles. After management approves the purchase and the problem is fixed, the customer is very happy.

The result of such a team effort is a customer base that is more invested in the evolution of the network.

### **Simple Automation**

The advocate manages time (see [Chapter 50](#), "[Time Management](#)") to allow work on projects that will prevent problems. The advocate also creates additional time in a day by automating the most time-consuming tasks. Automation opens doors to better ways of doing things.

The clerk might choose to put off all account-creation requests until a certain day of the week, then do them all in a batch. Customers see extremely poor turnaround time. The advocate automates the task so that creating accounts is simple and requests can be executed on demand.

Automating a task does not have to be extremely complicated. Don't get bogged down in creating the perfect system. A simple script that assists with the common case may be more valuable than a large system that automates every

possible aspect of a task. For example, automating account creation is easy, except for special cases, such as administrative accounts. Automate the 80 percent that is possible, and save those special cases for the next version of the program. Document the cases that require manual handling. Documentation is particularly important in these cases because the process is being done far less often, so it is easier to forget the nuances. Automation also hides how the normal cases are handled; thus, coworkers performing the special cases have no baseline against which to compare the solutions.

Rather than writing a script to automate things, one can write a script that outputs the commands that would do the task. The SA can review the commands for correctness, edit them for special cases, and then paste them to the command line. Writing such scripts is usually easier than automating the entire process and can be a stepping stone to further automation of the process.

Manually loading a machine's OS also can be a long process, especially if a series of from-memory customizations must be made. OS vendors provide tools for automating installations and configuration, as discussed in [Chapter 7](#), "[Workstation Software Life Cycle](#)." The advocate takes advantage of these tools; the clerk continues manually loading OSs.

Automation is often undertaken to improve repeatability or prevent errors and does not otherwise speed up the task. An easy sequence of commands that involves filenames that are easy to mistype is a good process to automate even if little time will be saved. The time savings come from not having to fix mistakes.

## 49.2 Visibility

So far, this chapter has dealt with improving how you are perceived. The next level is to increase your visibility. We were talking about quality; now we are talking about quantity.

The SA **visibility paradox** is that SAs are noticed only if something breaks. Achieving consecutive months of 100 percent uptime takes a huge amount of behind-the-scenes work and dedication. Management may get the impression that the SAs aren't needed, because it does not see the work required. Then a server crashes every hour until a controller is replaced. Suddenly, the SA is valuable. The SA is the hero. The SA is important. This is not a good position to be in; people get the impression that the SAs do nothing 95 percent of the time, because they don't see the important back-office work the SAs do.

One alternative is to maintain an unstable system so that the SAs are always needed and noticed. That is a bad idea. A better option is to find subtle ways to

make sure that the customers understand the value of what the SAs do.

You are responsible for the amount of visibility you receive. Take responsibility for achieving it. Nobody else will promote you except yourself.

None of these techniques should be attempted if you aren't providing good service to your customers. The keystone of good visibility is to already be doing good work. There is no point in advertising a bad product.

### **49.2.1 System Status Web Page**

A good way to be visible to your customers is to provide a web page that lists the status of your network. SAs should be able to easily update the status message so that when there are outages, they can easily list them. When there are no outages, the system should state so. The status should be time/date stamped so that people understand how current the information is. The page should have information about how to report problems, links to your monitoring systems, a schedule of planned outages, and news about recent major changes.

Most people start a web browser at least once a day. If the default web page is your status page, there is an opportunity to be in their field of vision quite often. Because customers can change their browser's default page, it is important to include content that is useful to the customers on a daily basis so that they are not tempted to change the default. Links to local weather, news, and corporate services can discourage people from switching.

A status web page sends a message that you care. If there is an outage being reported, the page can tell the customers that you are working on the problem, which in turn will reduce the number of redundant phone calls complaining about the interruption.

A status message should be simple and assure people that the problem is being looked into—for example, "Server `sinclair` is down; we're working on it." In the absence of this information, people often assume you are out to lunch, ignoring the problem, or ignorant of the problem. Taking ten seconds to give the status creates the positive visibility you desire.

As customers become accustomed to checking this web page, there will be fewer interruptions while you try to fix the problem. There is nothing worse than being delayed from working on a problem because you are busy answering phone calls from customers who want to help by reporting the problem. People will not develop the habit of checking this page until you consistently update it.

This can be as simple as a single web page or as complicated as a web portal that delivers other news and features. Commercial and free portal software has a

range of features and complexity.

Low-tech solutions also work well. One site simply put a whiteboard at the entrance to the machine room, and SAs wrote status messages there. If a method gets the message across, it is a good solution. This approach works well only in smaller environments where the machine room is convenient to the customers. At some sites, a webcam is aimed at the whiteboard to make it even more convenient to see.

### **49.2.2 Management Meetings**

Although being visible to all customers is useful, a more targeted approach is also valid. A regularly scheduled one-on-one meeting with the head of each customer group can be very valuable. Often, each SA is aligned with one or two customer groups. A 30-minute meeting every 2 weeks with the manager of each group can keep that manager aware of the projects being done for the manager's people. The manager can stay abreast of which kind of work is being requested by the staff and help prioritize those projects. You will occasionally find the manager eliminating certain requests. The secondary purpose of such meetings should be to educate managers about infrastructure changes being made that, although invisible to them, are being done to improve the network. This process goes a long way toward shedding light on the invisible.

### **49.2.3 Physical Visibility**

When considering your visibility, consider your physical visibility. Where you sit may cause you to be out of sight and out of mind. If your office is hidden behind a physical barrier, you are projecting an image of being inaccessible and unfriendly. If your office is in everyone's view, you are under the microscope. Every break you take will be seen as slacking off. People will not understand that an SA's job involves bursts of activity and inactivity.

It is good to find a balance between being physically visible and working in the shadows. A strategic way of managing this is to have a good mix of visibility within your team. People responsible for directly interfacing with customers and doing customer care should be more visible, whereas back-office programmers and architects should be less visible.

---

#### **Office Location and Visibility**

When Tom was at Bell Labs, there was an opportunity to move to new offices, owing to some renovation work being completed. His boss



arranged for the customer-facing SAs to be relocated to offices on high-traffic hallways. Senior, more project-oriented SAs received offices at the end of dead-end hallways that received less walking traffic. The physical location resulted in better visibility for the people who should be seen and hid the ones who should not usually be interrupted.

---

#### **49.2.4 Town Hall Meetings**

Another way of increasing your positive visibility is to host regularly scheduled meetings that are open to all customers. Such meetings can be an excellent forum for bidirectional feedback. Of course, they can also be a disaster if you are unprepared. Planning is essential.

Some organizations have yearly town hall meetings, which usually include a high-level manager giving a “state of the network” address that reviews the last year’s accomplishments and the next year’s challenges. This is usually a slide presentation. One way to communicate that your SA team has division of labor and structure is to have different area heads present briefly on their domains, rather than have one person do the entire presentation. Question-and-answer sessions should follow each presentation. Meetings like this can drag on forever if not planned properly. It is important to have all speakers meet to plan how long each will talk and what each will say. Have a script that is reviewed by all speakers. Make sure that there is little redundancy and that everyone adheres to the schedule. Have someone introduce people and signal people when their time is up.

Some organizations host monthly or quarterly meetings. These user group meetings often are half entertainment (to draw people in) and half bidirectional communication. The first half may be a demo or a speaker. You might have a vendor come in to talk about something exciting, such as a future product roadmap, or to introduce a new offering. Avoid sales presentations. SAs might introduce an interesting new tool to the customers and spark interest by presenting plans for major network upgrades, or information on hot topics, such as how to avoid spam, and so on. This is an excellent forum that can act as a dress rehearsal for paper presentations that SAs may be doing at future conferences, such as LISA. Doing this also communicates to the customers that the SA staff is receiving external professional acknowledgment of their good work.

The second half of the meeting consists of a bidirectional feedback session, such as a question-and-answer forum or a facilitated discussion on a particular

issue. Such meetings are also an opportunity to announce scheduled changes and explain why they are needed. Following is the format one organization uses for its quarterly user group meeting:

- 1. Welcome** (2 minutes): Welcome the group and thank them for attending. It is a good idea to have the agenda written on a whiteboard so people know what to expect.
- 2. Introductions** (5 minutes): The attendees introduce themselves. (For large groups, call for a show of hands for each team or other segment of the community.)
- 3. Feedback** (20 minutes): Solicit feedback from the customers. Set per-person time limits, so that everyone has an opportunity to express his or her views.
- 4. Review** (10 minutes): Review what you recorded during the feedback portion by reading the items out loud. Consider reviewing the feedback with your management afterward as well, to prioritize (and reject) tasks.
- 5. Show and tell** (30 minutes): This is the main attraction. People want to be entertained, which means different things to different people. The best way to entertain technical people is to have them learn something new. Nontechnical people might want to learn about some mysterious part of the system. Ask a vendor to present information on a product, have an internal person present some information about something he or she does or has found useful, or focus on a new feature of your network. This may be a good time to explain a big change that is coming soon or to describe your network topology or some aspect of the system that you find customers often don't understand.
- 6. Meeting review** (5 minutes): Have each person in turn briefly assess the meeting (less than one sentence each). For large groups, it can be better to have people say a single word or to simply raise their hands if they want to comment.
- 7. Closing** (2 minutes): First, ask for a show of hands to indicate whether people thought the meeting was useful. Remind them that you are available if they want to drop by and discuss the issues further. Then—and this is important—thank people for taking time out of their busy schedules.

---

## How to Get Good Feedback

Getting feedback from the customers is part art and part science. You want people to focus on their needs rather than on how those needs should

be met. Consider taking a class on meeting facilitation if you feel weak in this area. Ask open-ended questions—for example, “If one thing could be improved, it would be . . .” “The single best part of our computing environment is . . .” “The single worst part of our computing environment is . . .” “My job would be easier if . . .”

Keep a log of what participants suggest. A huge pad of paper on an easel is best, so everyone can see what is being recorded. Don’t write full sentences, only key phrases, such as “faster install new C++ releases” or “add CPUs server5.” Once a sheet is filled, tape it to the wall and go to the next one.

Do not reject or explain away any requests. Simply record what people say. To get the best responses, people must feel they are safe. People do not feel safe—and will stop talking—if every suggestion is answered with your reason why that isn’t possible or is too expensive. However, be clear that recording an idea does not guarantee that it will be implemented.

Be careful of permitting one talkative person to dominate the meeting. If this happens, you might want to use such phrases as “Let’s hear from the people in the room who have not spoken yet.” In extreme cases, you can go around the room, having each person provide feedback in sequence, without permitting others to interrupt.

Don’t get stuck in an infinite loop. If you hit your time limit, cut off discussion politely and move on. People have busy schedules and need to get back to work.

---

## 49.2.5 Newsletters

Many large system administration organizations produce a monthly or quarterly newsletter. Sometimes, these are excellent and useful to the customers, but they are mostly ignored or may backfire by making it appear that your team is more concerned with PR than with solving problems.

If you must have a newsletter, it should be simple and useful: a simple layout, and easy to read. The content should be useful to the intended audience, possibly including frequently asked questions or an “ask the SAs” column that gives a detailed answer to a question that has been asked.

If you have to hire a full-time person to do nothing but produce the newsletter, you don’t have a simple enough newsletter.

## 49.2.6 Mail to All Customers

Before major changes, such as those that might be implemented in a maintenance window ([Chapter 34](#), “[Maintenance Windows](#)”), send a brief email to all customers, telling them about the outage and the improvements that they will experience afterward. Sending useful mass email is an art. Make the message very brief and to the point. The most important information should appear in the first sentence. Most people will read only the subject line and possibly the first sentence. Extra information for those who are interested should be contained in a few additional, brief paragraphs.

Include the text “Action Required” in the subject line if action is required. Then clearly indicate that action in the body, and explain what will happen if action is not taken.

For example, a good mass email about system maintenance might read as follows:

[Click here to view code image](#)

```
Subject: No printing in Bld 1 and 2 SATURDAY MORNING
```

```
Printing in Buildings 1 and 2 will not work on Saturday, June 24,
between
8 AM and 11 AM because of essential system maintenance.
```

```
If this is a problem, please let John Smith know at extension 54321
as
soon as possible.
```

```
The printing servers in those buildings are approaching the end of
their
useful lives, and we anticipate that they will become less reliable
in
a few months. We are replacing them with new hardware now, so that we
don't suffer from reliability problems in the future. If you have
further
questions, please contact us by email at print-team@company.com.
```

In contrast, an email sent to all customers that is wordier, like a traditional letter, is less useful. The following, for example, does not get the important point across quickly:

[Click here to view code image](#)

```
Dear all,
```

```
To serve you better, the system administration team monitors all
components of the system. We endeavor to anticipate problems and to
fix them before they arise. To do so, we need to occasionally
```

schedule  
outages of some component of the system to perform maintenance. We do  
our best to schedule the maintenance at a time that will not  
adversely  
affect any urgent work in other parts of the company.

We have identified a potential problem with the print servers in  
Buildings 1 and 2. We anticipate that those print servers will become  
less reliable in a few months' time and will then cause problems  
printing  
in those buildings. Because of this, we have scheduled some time on  
Saturday, June 24, between 8 AM and 11 AM to replace those servers  
with  
new, reliable machines. Because of that, you will be unable to print  
in  
Buildings 1 and 2 during those hours.

If the timing of this maintenance window will interfere with some  
urgent  
work, please let us know as soon as possible and we will reschedule  
it for  
another time. John Smith, at extension 54321, is the person to  
contact  
about scheduling issues.

As always, we are happy to answer any questions that you might have  
about  
this maintenance work. Questions can be addressed directly to the  
people  
working on this project by sending email to [print-team@company.com](mailto:print-team@company.com).  
All  
other questions should be addressed to [helpdesk@company.com](mailto:helpdesk@company.com) as usual.

Thank you for your cooperation with our ongoing maintenance program.

The SA Team

Such email should always include two ways of contacting the SAs if there are  
questions or if someone has an issue with what is being announced. It is critical  
that this contact information be accurate. We've seen many complaints from  
people who received such a notice but for various reasons weren't able to contact  
the author. It is important that the various contact methods are two different  
means (email and phone, web and phone, and so on) and not simply, for  
example, two email addresses. That would essentially be like telling people, "If  
your email isn't working, please send email to our helpdesk to have your  
problem fixed."

Mass email should be used only occasionally and for important changes. Too  
much mass email or mass email that is too wordy wastes everyone's time and

will be seen as a nuisance.

---

### **Who Needs to Read This?**

As part of the Google culture, all mass email begins with a statement saying who doesn't need to read the message. Some real examples:

- If you don't use [URL to internal service], you can stop reading now.
  - If you don't write C++ code on Linux, you can stop reading now.
  - If you don't use GFS or have no idea what GFS is, you can stop reading now.
  - If you haven't had a birthday in the last 12 months, you can stop reading this now.
- 

## **49.2.7 Lunch**

Breaking bread with customers is an excellent way to stay visible. Eating lunch with different customers every day or even once a week is a great way to stay in touch in a way that is friendly and nonintrusive.

---

### **Free Lunch**

Tommy Reingold at Bell Labs watches the serial numbers on the trouble tickets that are generated. The creator of every 1,000th ticket is taken out to lunch. He pays for the lunch out of pocket because of the value he gets out of the experience. It's a simple, inexpensive thing to do that helps build the reputation of the SA team as being fun and interesting. This "contest" isn't advertised, which makes it even more of a surprise to customers when they win. It can be fun to watch people send an extra ticket or two to try to increase their chances of winning.

---

## **49.3 Summary**

Do not leave perception and visibility to fate. Take an active role in managing both. Unmanaged, they will be a disaster. As we become conscious of these concepts, we quickly find many ways to improve them.

Perception is about quality—that is, how people perceive you. Creating a good first impression is a technical issue that requires much planning. Establishing the processes by which new customers have their computers and all accounts the day they arrive requires coordination between many different

departments. It is important that new customers receive some kind of orientation to welcome them to the network, as well as some kind of getting-started documentation.

We find that referring to users as “customers” changes how we treat them. It focuses our attention on the fact that we serve them. It is important to treat your customers with respect. There is no such thing as a “stupid” question.

We discussed reordering the requests that you receive, so that their completion time is aligned with the expected completion time.

We also discussed the system-advocate philosophy of system administration. A system advocate is a proactive SA who takes the initiative to solve problems before they happen. Transforming a clerical role into a proactive system advocacy role involves a change in attitude and working style that can dramatically improve the service you provide to your customers. It isn't easy: It requires hard work and an investment in time now for a payoff later.

Visibility is about quantity—how much people see of you. We discussed many ways to increase your visibility. Creating a system status web page puts you in front of customers' eyes daily. Meetings with managers help them understand what you do and help you maintain focus on those managers' highest priorities.

The office locations of every member in your team affect your team's visibility. Customer-facing people should be in more visible locations. Town hall and user group meetings should be held to increase your visibility even further. Newsletters are often produced by SA groups but rarely read by customers. They are a lot of work to produce and too easy to ignore. Having lunch and participating in social functions with customers is a simpler way to maintain interaction.

Taking control of your perception and visibility is required to manage your and your team's personal positive visibility. Managing these things effectively enhances your ability to work well with your customers, increases your opportunities to serve your customers better, and has great potential to enhance your career.

## **Exercises**

1. What is the first impression you make on your new customers? Is it positive or negative? What can you do to improve this?
2. Ask three customers what they remember about the first time they interacted with your SA team. What did you learn from this?
3. At your site, who gives new hires their first-day orientation?

4. Do the members of your team use the term “user” or “customer”? Which behavior do you role-model for others?
5. When you need to vent about a customer, whom do you talk to and where?
6. Select ten typical customer requests, each requesting a different service. For each of these requests, try to guess the customers’ expectations as to how long it should take to complete a request of that type. Poll three customers for their actual expectations for those types of requests. How close were your guesses to their answers? What did you learn?
7. How does your organizational structure benefit or hurt attempts to match customer expectations of completion time? How could this be improved? What do you do on the personal level in this regard, and what can you do to improve?
8. On a scale of one (clerk) to seven (advocate), where are you? Why do you say so? How did you decide on your rating? Which steps can you take to move toward becoming an advocate?
9. Do you experience the system administrator’s visibility paradox mentioned in [Section 49.2](#)? Give some examples. What can you do to turn this situation around?
10. Which of these projects would have the biggest positive impact on your organization’s visibility: a system status web page, regular meetings with key managers, reorganizing your team’s office locations, town hall meetings, user meetings, a newsletter, or lunch with your customers?
11. Does your group produce a newsletter for customers? Poll five customers on whether they read it and what they find useful if they do. What did you learn?
12. Who on your team is best qualified to host a town hall meeting?
13. Ask your manager what needs the most improvement—your perception or your visibility. What about your team?



## Chapter 50. Time Management

This chapter is about time management: how to manage the tasks on your plate, deal better with interruptions, and achieve perfect follow-through.

Time is limited, so it should be used wisely. Rather than increasing productivity by working more hours, you can do more in the same amount of time by using a number of helpful techniques and doing a little planning.

Customers don't see how hard you work. They see what you accomplish. Align yourself with this bit of reality. Anybody can work hard, and we're sure that all the readers of this book do. Successful SAs focus on results achieved, not on effort expended.

### 50.1 Interruptions

Time management is extremely difficult for SAs because an SA's job is typically driven by interruption. People or other external events interrupt you with requests. So, rather than working on your high-priority goals, you spend time responding to these requests, which are based on other people's priorities. Imagine trying to drive a bus from New Jersey to San Francisco, but stopping the bus every time a passenger asked you to. You might never get to your destination! Although it is valuable and interesting to have stopped at those locations, your job was to get the bus and the passengers to San Francisco. A critical step toward good time management for SAs is to break this kind of cycle.

Try to prevent interruptions in general. Make sure people know when you do or don't want to be interrupted. If you have an office door, close it when you need to focus on a project. If people are supposed to open a ticket rather than come to your desk, make sure this policy is advertised on your department's home page, in your email signature, and anywhere else people may see it.

When you are interrupted, you can deflect the interruption by recording the request in your personal to-do list and tell the person when you will have it done.

Sometimes, you may be unable to record the request because someone approaches you in the hallway and you don't have your phone with you. Ask the person to submit a ticket. You might say something such as "I'm not going to remember your request, because I can't write it down right now. Could you please open a ticket?" It is easy to come off sounding rude in this situation, so be extra nice. Asking people to open their own ticket trains them to use the channels that were created specifically to help them.

### **50.1.1 Stay Focused**

Sometimes our focus is broken for reasons that we cause ourselves. Messy desks are full of distractions that make it difficult for the brain to focus. Extending that to computer users means also keeping a clean email box and a clean computer GUI desktop. The more icons on the screen, the more visual distractions present to make your mind wander. As a better alternative, virtual screens can greatly help maintain focus by showing only the information that you want to address.

It may sound radical, but you also might disable any systems that let you know when you have new email and instead set aside a couple of small blocks of time each day during which you read email. Email can become one of the interruptions that prevents you from getting your job done.

### **50.1.2 Splitting Your Day**

Another way to manage interruptions is to split your day between project time and customer-focused time. During project time, you work on long-term projects that require focus and concentration. If you are interrupted, you tend to ask the person to open a ticket, find someone else, or wait. You still handle emergencies, but you are biased toward deflecting interruptions.

During customer-focused time, you reverse these priorities. You focus on direct customer support and work on projects only between interruptions.

If you are on a team of two SAs, you can alternate filling these roles. In the morning, one person works on projects and the other handles interruptions. In the afternoon, the two switch. This way each SA has about half a day of project time each day.

A solo SA can use this technique by allocating the first and last two hours of the day for customer-focused work, and using the middle of the day for project time. The first and last part of the day is when people most need support, so this schedule works well in most organizations.

## **50.2 Follow-Through**

Follow-through means completing what you committed to do. Successful SAs stay organized by recording their to-do lists and appointments in either a written or electronic form. Whether you use a written (paper) organizer or a smartphone app, being organized goes a long way toward ensuring good follow-through. Nothing is more frustrating to customers than dropped requests and appointments. You will be happier if you develop the respect that comes from having a reputation for excellent follow-through. This is one of the reasons that,

throughout this book, we put so much emphasis on using trouble-ticket tracking software. Such software assures us that promises are not forgotten.

Your brain has only so much storage space, so don't overload it with items that are better recorded in your organizer. Albert Einstein is rumored to have been so concerned about making sure that 100 percent of his brain was used for physics that he didn't "waste" it on such silly things as picking out what to wear (he had seven outfits, all the same—one for each day of the week) or remembering his home address and phone number (he kept them written on a card in his wallet). When people asked him for his phone number, he advised them to look it up in the phone book. He didn't memorize it.

Your memory is imperfect. Your organizer, however, will not accidentally drop an action item or reverse two appointments. You should have a single organizer, so that all this information is stored in one place. This is better than having a million pieces of paper taped to your monitor. Combine your work and social calendars so that conflicts don't arise. You wouldn't want to miss an important birthday, anniversary, or meeting with a customer. Maintaining separate calendars at home and at work is asking for trouble. They will become out of sync.

A combined organizer should include your social events, anniversaries, doctor's appointments, non-work-related to-do items, and reminders of regular events, such as the dates for your next annual physical exam and your car's next inspection. Use your organizer to remind yourself of regular, repeating events. Include reminders to take breaks, to do something nice for yourself, and to compliment your significant other. Use your organizer to record the names of movies that you want to see so the next time you have free time, you aren't frustrated trying to remember that great movie that co-workers recommended. None of these things should be lost in a mess of notes taped to your desk or in your messy and imperfect brain.

### **50.3 Basic To-Do List Management**

In an attempt to achieve perfect follow-through, Tom maintains a separate to-do list for each day. If someone asks him to do something next Thursday, he writes it on that day's to-do list. It doesn't get lost.

During the day Tom works on the items on that day's list. He ends each day by copying the incomplete items from today's list to tomorrow's list. Already on tomorrow's page are items that he previously committed to doing that day. If he's overloaded, he can copy the incomplete items to lists even further into the

future.

In the morning, Tom reads today's list and tags each item that absolutely has to be done today. This 5-minute investment has a huge payoff in keeping him focused. He first works on those tagged items. After those are completed and crossed off, he works on the remainder of the list. Previously, Tom felt it was impossible for an SA to prioritize; now, he is able to set priorities each morning.

When a new item is added, Tom writes it on the first day that he'll be able to work on it. Usually, that's today's list; sometimes, however, it may be a few days or weeks away. Previously, Tom felt that it was impossible for an SA to plan very far ahead; now, he is able to schedule work rather than always feeling pressured to do it right away or before he forgets.

Near the end of the day, Tom can see what hasn't been completed. If it looks as though he'll miss a deadline he committed to meeting for a customer, he can call the person and ask whether the item can wait until first thing the next morning or whether it is critical and he needs to stay late to work on it. Previously, he always felt pressured to work late every night or feel guilty over every missed deadline. Now, he can call people and renegotiate deadlines. The need to copy incomplete items forward also gives Tom an incentive to not let them slip, because it takes time to copy them, and some might have to be copied several times before they are finally completed.

By the end of the day, every item has been somehow managed. An item is either completed and crossed off or moved to a future date and crossed off. Tom benefits from a feeling of accomplishment and closure. Previously he'd leave work feeling that there was no end in sight to all the work he had, since there were so many open issues. Now he can sleep better knowing that he has gotten the important things done and the others are no concern of his until the next day.

When writing monthly or yearly reports, rather than struggling to remember what he's accomplished, Tom can use his completed to-do lists as a reference.

There are software packages, smartphone apps, and web-based systems that automate this process. Before smartphones, Tom used a paper-based system because it is easier to write notes in the margin, and has zero reboot time. Tom also draws a lot of diagrams, and that's easier to do with paper. Now, however, apps have finally reached feature parity, plus they have the benefit of being able to integrate with enterprise calendar systems and sync so that the information is available on any device.

Once you have been maintaining your organizer for a while, you will find that you can focus and concentrate better. You can focus better on what you're doing

when you aren't also trying to remember what you have to do next, what you have to do next week, and whether that customer meeting is next Wednesday or the Wednesday after that. It's as if you can purposely forget things after you have recorded them in your organizer. Each morning, you can review today's items to refresh your memory and set your focus for the day. Let your organizer work for you; don't duplicate its work.

Having any kind of system is better than having no system at all. It's not important which system you use, but it is important that you use some kind of system. This is such a critical work component that employers should pay for their employees' time-management tools, whether they are apps or stationary refills.

---

**Make the First Hour Count**

Some people come to the office a little earlier than their co-workers so that they have time to work with very little interruptions. You can get more done in that time than you can during normal work hours.

Don't squander this time reading email and visiting the various web sites that you check every day. Use this most productive hour for something more productive, such as starting a project you haven't been able to find time for.

---

## 50.4 Setting Goals

SAs often feel as though they are spinning their wheels—working hard month after month, but not getting anywhere. As has been said in other places in this book, you are spending so much time mopping the floor that you don't have time to fix the leak. You can break this cycle only with conscious effort—and goal setting can help you accomplish that feat. Take some time to set goals for the next month. Write down everything that comes to mind, in any order. Then prioritize those items, and eliminate the low-priority items. Plan which smaller steps are required to meet the remaining goals. Prioritize the tasks that will bring you closer to those goals, and stop doing the things that won't.

Planning a year in advance may seem like science fiction. If you are at a startup, even planning a few months ahead might seem impossible. Even so, don't pressure yourself to have only hard “business” goals. Examples include: Fix or replace the four least-reliable servers, be on time for meetings more often than not, meet monthly growth plans on time, force yourself to take time for yourself, get a promotion, accrue a down-payment on a house, get a master's

degree, automate OS installation.

One useful technique is to spend one hour on the first day of each month reviewing your accomplishments from the previous month to see how they got you closer to your bigger goals. Then plan what you want to accomplish in the next month, possibly revising your one-year goals. You can begin each week by reviewing your status. This has the effect of keeping your goals at the front of your mind. How many times have you gotten close to a deadline and thought, “Gosh, I can’t believe I forgot to work on that!” Trust the process. It works.

Although the first hour of the day may be extremely productive because of the lack of interruptions, you are likely to find another time of the day that is extremely productive because of your individual biological clock. Some people find that their most productive hour is in the midafternoon or 6 PM or even 2 AM. There is no rhyme or reason to it; it’s just the way people are built. Whatever it may be, plan your schedule around this golden hour. Schedule the work that requires the most thought, attention to detail, or energy during that hour. Your most productive hour might not be when you expect it. It might also change as you grow older, sort of the same way that your sleep patterns change as you grow older. The only way to find your productivity peak is to slow down, listen to your body, and pay attention.

## **50.5 Handling Email Once**

If you open an email message, process it. If you don’t have time to process it, don’t open it. We tend to double the amount of time consumed by mail that we receive because we read an email and decide that this is something we’ll need to deal with later. When later arrives, we reread the email and deal with it. We’ve had to read the message and mentally process it twice. We can go through email faster by opening only the emails we have time to process.

In the old days when paper memos ruled, executives received memos and wrote memos in reply. A lot of time was spent filing the original memos and a copy of the replies. Time-management books in the 1980s recommended breaking this rule by encouraging people to simply write a brief reply in the margin of the original memo, then send it back to the person. The amount of time spent managing a file cabinet of old memos did not justify the few times an old memo was needed. If you ever did actually need a copy of an old memo (which you never did), you could have asked the person who sent it for another copy.

We can apply this technique to email. First scan the subject line. We can

usually decide to delete most emails without reading more than the subject. This is especially true if we adopt a culture of putting the important information in the subject line. If the subject is “4:00 PM meeting canceled,” we’ve gotten all we need to know from that message. If the subject is “about the 4:00 PM meeting,” then we have to open the message to find out that the meeting is canceled.

The remaining messages should be read once. If you open it, reply. If you won’t have time to process it, don’t open it. Minimize the amount of time spent filing messages. Some people file each email they receive in a folder named after the project, topic, or issue. Soon the list of email folders is so long that filing each message takes longer than reading the message. Instead, if you need to file a message, file it to a single archive folder and trust that the search capabilities of your email system will enable you to find it if you need it in the future. One of the reasons Google’s Gmail service is so popular is that it archives all messages and provides excellent search capabilities.

SAs are often on many mailing lists. If you aren’t using some kind of email filtering system, you’re working too hard. Each high-volume mailing list should be filtered to its own folder, which is read weekly or when there is spare time. The messages that get to your inbox should be messages that are directed specifically at you. We make an exception for mailing lists that are extremely low volume, such as announcement-oriented lists.

---

### **Do Daily Tasks Early**

If something must be done today, do it first. That ensures that it gets done. If there is a task that you have to repeat every day, schedule it for early in the day.

Tom used to have to change backup tapes every day. It took about 15 minutes if everything went right and an hour if it didn’t. His plan, which didn’t work well, was to start changing the tapes at 5:30 PM so he could leave by 6:00 PM.

If he started at 5:30 PM but the task took only 15 minutes, he would waste the other 15 minutes because he didn’t want to start a new project right before leaving. That amounted to more than an hour a week of wasted time.

If he had been deeply involved in a project and lost track of time, he would find himself changing tapes when he was already late for whatever he was doing after work. He’d be stressed because he was late, angry while changing tapes because now he was going to be even later, and

arrive at his after-work function stressed and unhappy.

He tried changing the tapes first thing in the morning, but that conflicted with his attempt to use that first hour for his most important tasks. Finally, he settled on changing tapes immediately after lunch. This worked well and became somewhat of a ritual that got him back into work mode. If something has to be done every day, don't schedule it for the end of the day.

---

## 50.6 Precompiling Decisions

It is more efficient to make a decision once rather than over and over again. This is why compiled languages tend to be faster than interpreted languages. Think about what a compiler's optimizer does: It spends a little extra time now to save time in the future. For example, if a variable is added to a constant, the optimizer will eliminate this calculation if it determines that the constant is zero. An interpreter, in contrast, can't do such an optimization because it would take longer to check whether the constant is zero before every addition than it would to simply do the addition.

You can precompile your decisions the same way. Decide to do something once and retain that decision. When you are breaking your old habit, your mind might start to try to make the decision from scratch. Instead, distract your brain by replacing those thoughts with a mantra that represents your precompiled decision. Here are some mantras that have worked for us:

- **It's always a good time to save your work.** In the time it takes to decide whether it's a good point to save your work, you could have pressed the keys to have it done already.
- **Always make backups.** Always back up a file before making a change. Often, someone thinks that the change is so small that it can be undone manually. Then the person ends up making a big change. It's better to decide to always make a backup rather than spend time deciding whether to do so.
- **Record requests.** Rather than try to remember a request, write it down—add it to your to-do list, create a helpdesk ticket, write it on the back of your hand. Anything is better than trying to remember something. And yet, we find ourselves saying, “This is so important! How could I possibly forget it?” Well, if it's that important, it's worth opening a ticket.
- **Change tapes Monday–Wednesday–Friday.** In [Section 44.9](#), we



discussed a backup system that required a lot of daily thinking to decide whether tapes should be changed. Instead, it was decided that time was more valuable than blank tapes, and tapes were simply changed on certain days whether it was necessary or not.

- **Sooner is better than later.** Doing something as soon as possible helps prevent procrastination. For smaller tasks, “sooner is better than later.” For example, Tom would often procrastinate on small items because if they were small, he could do them “anytime.” However, “anytime” would rarely arrive. For example, when driving home late at night, he would notice that his car was low on gas. He’d decide that he could fill the tank on the way to work the next day, especially if he could remember to leave a little early. As luck would have it, he would be late the next day, and needing to get gas would make him even later. After adopting this mantra, he would get gas at night when he realized that his car was running low.

There are many similar situations in the life of an SA: Place the order now, make the phone call now, start the process now, and so on. When you start to debate with yourself about whether it is the right time to do these tasks, remind yourself that “sooner is better than later.”

Take time to consider which decisions you find yourself making over and over again, and select one or two to precompile. You will lose a little flexibility, but you will gain a lot of other benefits. The mantras you develop will be specific to you.

## 50.7 Finding Free Time

People generally feel they do not have enough time to do all the things they want to do. Actually, free time is hiding all over the place, but you have to look hard to find it. You can also create free time by getting rid of time wasters. Here are some easy places to look for spare time:

- **Find a “light” time of the year.** Software shops with a new release every four months usually have a three-to four-week slow period after each release. You might have spare time then, or that may be the busy time for you as you get things ready for the customers’ busy periods. You may find that during their busy periods, nobody bothers you.
- **Eliminate rather than automate.** As SAs, we tend to accumulate things and then spend a lot of time managing what we’ve accumulated. We think we’re improving our situation when we find better ways to manage our accumulated time wasters and often forget that we would save even more

time by having a smaller list of things to manage in the first place.

- **Stop reading social media.** Period. Go a month without it and you'll find you don't miss it.
- **Remove yourself from the two busiest mailing lists you are on.** Repeat this once a month.
- **Filter email.** Take advantage of mail-filtering software, such as `procmail`.
- **Take advantage of the early morning.** Come in an hour earlier. There are fewer interruptions during this time.
- **Shorten your commute to work.** Avoid rush-hour traffic. Work odd hours. Determine your body's "alertness hours," and change your schedule accordingly.
- **Invest in training workshops and books.** Focus on ones that will enable you to automate tasks (write code) in less time. Learn `make`, and a scripting language such as PowerShell, Python, Ruby, or Perl.
- **Set priorities with key customers.** Hold weekly or monthly meetings with each key customer—a manager or a department head—to set priorities and eliminate superfluous items.
- **Hire an assistant.** A part-time assistant to take care of some of the clerical (tactical) tasks can free you for more important (strategic) duties and can be a great way to train someone to be an SA. Possible candidates: a local high school student, a technically inclined administrative assistant, the person who covers for you when you are away, a summer intern, or an interested programmer. Students are particularly good candidates because they are inexpensive and are looking for ways to gain experience.

## 50.8 Dealing with Ineffective People

Sometimes, you find yourself dealing with people who lack good time-management skills, and their ineffectiveness affects your ability to get your job done. We recommend three steps.

First, try to coach them. Help them see what is wrong with their processes, and encourage them to get help. Do this tactfully. People hate to be told that they are wrong; lead them to realize it themselves. Ask whether there is any way that you could help them get this done more quickly.

Second, work with their management. Raise the issue and recruit them to help. Maybe they have advice on how to work better with the person.

The third suggestion is a bit radical and should be done only if the first two are not possible—for example, if you are dealing with these people for only a short time. In this case, we suggest that if they don't have good time-management skills, manage their time for them.

For example, suppose your project is delayed because you are waiting for something to be completed by someone else. This person never gets around to your request because of interruptions. In this case, make sure you are the highest-priority interrupt: Stand in this person's office until your request is complete.

Ironically, this is exactly what you should *not* let anyone do to you. However, it's obvious that the person you are dealing with in this situation hasn't read this chapter. Don't let someone drowning in bad time management drag you down.

If you notice that people are doing this technique to you, maybe that means you need to get some more extensive coaching and time-management help.

## **50.9 Dealing with Slow Bureaucrats**

There are many techniques for dealing with slow bureaucrats. This section discusses the best two. The first technique is to befriend these bureaucrats. These people typically deal with an endless stream of faceless people who are angry and impatient. Be one of the pleasant people they talk with today. Give them a change of pace by being the opposite of the last ten people with whom they've dealt. They will work harder for you. Talk with them about things that might interest them, which may be entirely boring to you. Talk as if you've been friends forever, but be sincere. Ask them how their day is going, and offer sympathy when they tell you how overloaded they are. If talking via phone, ask how the weather is in their area. To really get on their good side, say something like, "Can you believe that latest decision by our CEO/president?" Although this social banter may sound like a waste of time that slows you down, it's an investment that pays off in better service.

The other technique works when you have to deal with the same person over and over. When a bureaucrat tells us that a request takes multiple weeks to complete, we often back off and conserve the number of requests that we give this person so as to not overload him or her. That simply delays our own projects! Instead, do the opposite by giving the person hundreds of requests at once. Rather than taking longer to complete, large requests usually get escalated to managers who have the authority to streamline processes, cut corners, or make special exceptions. For example, if each request requires individual approval, the

manager will find a way to classify some or all of the requests that are similar and push those through all at once. Bureaucrats' jobs involve maintaining processes that they do not set and therefore cannot optimize. If the problem is the process, doing something unusual makes your request break out of that process. Then it is something that the person, or the person's manager, does control and thus can create a more efficient process.

---

### Other Time-Management Resources

Time-management training can be a small investment with a big payoff. It is very inexpensive compared with the potential productivity gain. Many internal corporate education centers offer one-and two-day time-management classes. Take advantage of these internal courses, especially if your department pays for these with "internal dollars" instead of real cash. Most classes teach dozens of techniques, and each student finds a different subset of them useful. Enter such training with an open mind.

Many books are available that provide a deeper discussion of time-management techniques. Of course, we recommend *Time Management for System Administrators* by Limoncelli ([2006](#)). We also recommend *How to Get Control of Your Time and Your Life* by Lakein ([1996](#)) and *The Time Trap: The Classic Book on Time Management, 3rd ed.*, by MacKenzie ([1997](#)). For advice on getting organized and setting goals, we recommend *Organizing from the Inside Out: The Foolproof System for Organizing Your Home, Your Office, and Your Life* ([Morgenstern 1998](#)).

---

## 50.10 Summary

This chapter provided a high-level overview of a few of our favorite time-management techniques. We recommend books and courses on time management to help you establish your own time-management system.

A successful system administrator organizes his or her work in a way that ensures the important tasks get done. You need to have three systems: a goal-setting system, a way to work and handle requests, and a system to manage interruptions.

Time management is a discipline that helps you accomplish your highest-priority goals. It is difficult for SAs to manage their time, because there is so much temptation to let interruptions drive their actions. SAs must set goals if they are to achieve them. Planning your day is a good way to stay on track. Reading email efficiently can cut your mail-processing time in half. Staying

focused requires discipline as well. If something is the highest priority, stay focused on it until it is done. Fill the wait time with your other priorities. Finding free time is a matter of eliminating the time wasters, not managing them better or doing them more efficiently.

You can be more efficient by making decisions once rather than deciding the same thing over and over, by making better use of your time, and by not letting other people's inefficiencies drag you down.

The habits listed in this chapter are difficult to develop. Expect to be a little frustrated when you begin, but assure yourself that things will get easier as time goes on. One day, you'll notice that you've mastered time management without realizing it.

## **Exercises**

1. What do you do to ensure good follow-through?
2. What percentage of your day is interrupt-driven? What can you do to reduce the interrupt-driven nature of your job?
3. What are your goals for the next month, the next year, and the next five years?
4. How do you spend the first hour of your day? What can you do to make that hour more productive?
5. How do you manage your email? Which strategy do use to filter your email?
6. When someone who asked you to do something is standing in your office until you complete it, is the requester doing what is recommended in [Section 50.9](#)? What does this say about your time management and the service you are providing to your customers?
7. What time-management training is available to you?
8. Which tasks do you have to do every day or every week? When do you do them?
9. Name three low-priority items you can eliminate from your to-do list.
10. Name three time wasters you can eliminate.

# Chapter 51. Communication and Negotiation

This chapter is about basic interpersonal communication and negotiation skills. Learning to communicate well is crucial to being successful in your job and in your personal life. All problems can be viewed as communication problems. In an office, two people who don't like each other simply haven't taken responsibility for learning how to communicate and overcome their disagreements. You shouldn't have to like someone to work with that person. You simply need to be able to communicate.

Even technical issues are communication issues: A dead disk on a server is a *problem* only if no one communicated that an outage would be unacceptable or if the communication was not listened to and acted on. For example, someone could have communicated that RAID should have been used to make the system survive single-disk failures.

SAs need good negotiation skills because they are often dealing with vendors, customers, and their own superiors.

## 51.1 Communication

Problems in life generally fall into one of four categories: my problems, your problems, our problems, and other people's problems. Each can be helped by a different set of communication skills.

- **My problems:** When I have a problem, I need to make sure that I'm being heard. Later in this chapter we discuss a technique called "I statements."
- **Your problems:** When you bring a problem to me, I need to make sure that I'm hearing you properly so that I can take appropriate action or help you troubleshoot. Later in this chapter we discuss a technique known as "active listening."
- **Our problems:** When you and I have a common problem, we need to be able to communicate with each other so that we can agree on a problem definition and action plan.
- **Other people's problems:** When other people have problems, I have to use discipline and not get involved. People spend a lot of time worrying about other people's problems. Instead, they should be focused on their own concerns. The communication skill required here is the discipline of minding your own business.

## 51.2 I Statements

**I statements** help you make your point and also communicate your feelings. The value of an I statement is that it lets you get the issue off your chest in such a way that others can do something constructive about what you have said. You have given them a fixable problem statement. What happens as a result is usually positive. When we make our needs known, the universe tends to take care of us.

The general form is “I feel [*emotion*] when you [*action*].” Such a statement makes people aware of the effect of their actions. This is much more effective than simply telling someone that you don’t like a particular behavior.

An I statement expresses soft emotions (sadness or fear), rather than hard emotions (anger). It is said that underlying every hard emotion is a soft emotion. So, before you express your I statement, figure out what the soft emotion is and express *that*. People who hear anger will become defensive and will not hear you. By comparison, hearing fear or sadness makes people care for you and help you solve your problem. Here are some sample I statements:

- I feel hurt when you criticize me instead of my work.
- I feel unvalued when you credit my boss with the work that I did.
- I feel very happy when you complete a project on time.
- I feel frustrated when you demand reliability but won’t fund the changes I recommend.
- I feel disappointed when I receive complaints from customers saying that you don’t meet your commitments.
- I feel untrusted because you installed web-censoring software on our gateway.

## 51.3 Active Listening

Listening is more important than talking. That’s why humans have twice as many ears as mouths. Active listening is a technique that ensures complete communication. We discuss three tools: mirroring, summary statements, and reflection.

### 51.3.1 Mirroring

You wouldn’t trust a file transfer protocol that sent packets but never checked that they had arrived. All too often, though, people talk without pausing to verify that they are understood, and listen without verifying that they understand. In effect, most people they treat conversations as two unidirectional packet streams.

We can do better using active listening, or **mirroring**.

Active listening means that when the listener hears something, the listener seeks to understand what was said before replying to it. Instead of replying with the next statement, the listener takes the time to *mirror* back what was just heard with an accurate but shorter statement. It's like verifying a packet checksum before you use the data.

If someone says, "People are complaining that they can't get any work done and they are going to miss their deadlines because the file server is so slow," the listener might be tempted to offer a solution right away. Instead, it is better to say, "So you are saying that several people are complaining about the file server speed?" The speaker then sends an acknowledgment or correction: "Actually, Mark got a single complaint and passed it along." Now the listener understands the situation much better: Mark forwarded a complaint about slow file server performance.

Although all such complaints should be verified before they are fixed (see [Section 28.3.3](#)), Mark's technical competence can indicate whether the complaint has been filtered for validity already. Also, we now know that this information has been relayed twice, which indicates that information loss is occurring. The reaction to this complete information is much different, and more focused, than a typical reaction to the original statement.

Your interpretation of what someone said is based on your upbringing and experiences, which are completely unique to each person. The more diverse your organization is, the more important mirroring becomes. The more you deal with people from other parts of the country or other parts of the world, the more likely it is that the person you are speaking to has a different basis for semantic interpretation.

---

### **Don't Trust Your Ears**

Tom was in a start-up that was struggling with only 10 people but had a promise of hiring 40 more people at the next round of funding, for a total of 50 people. The funding arrived but was less than expected, and the CEO told the team, "We'll have to make some changes because we will have a smaller team in the next year." The CEO meant that the 50-person goal was reduced, but everyone in the room thought he meant the current team of 10 would be reduced. Both interpretations are semantically valid, but active listening helped clear up the confusion. Someone mirrored his statement by saying, "So I hear you saying that we're reducing our group



rather than hiring new people.” The CEO immediately realized that he was misunderstood and was able to clarify what he meant.

---

To sound as though you aren’t challenging the person, it is useful to begin a mirror statement with “I hear you saying that ....” Once you understand the speaker, you can react to what you’ve heard. If everyone on your team uses that same phrase, it can become a quiet signal that active listening is being attempted and that you are trying to understand the person in earnest. This can be really useful, or it can become a running joke. Either way, consider it “team building.”

---

### **What Does “What?” Mean?**

Standardizing on phrases such as “I hear you saying” may sound silly but can be useful. A small team worked in a noisy environment and often couldn’t hear one another very well. This problem was made worse by the fact that listeners asking “What?” sometimes sounded as if they were challenging the speakers, who then became defensive and reexplained what they had just said instead of simply repeating it. The listener, who simply wanted the last statement repeated, would become frustrated. The team members established the protocol of saying “Say again?” to mean “Please repeat that; I didn’t hear you,” so that it was no longer confused with, “Are you nuts? Prove it!” An outsider listening to them saying “Say again?” all the time might have found it humorous to watch, but this protocol worked well for the team.

---

## **51.3.2 Summary Statements**

Summary statements occur when you pause the conversation to list the points made so far. A **summary statement** is a form of mirroring but covers more material. It is often useful after a person has completed several long points and you want to make sure that you heard everything and heard it correctly. Sometimes, summary statements are used when you feel that people would benefit from hearing what they just said. Sometimes, hearing someone else summarizing what you’ve just said can be enlightening, especially if the person reframes the statements a little differently. Summary statements are also vital toward the end of a meeting to make sure that everyone leaves with an understanding of what’s happened and what will happen next.

A summary statement should list the points in short, pithy statements. If people correct your summary, take the time to process clarifications and then

repeat the summary statement. Sometimes, boiling down a person's thoughts helps the person solve his or her own problem.

Here's a sample summary statement: "Let me summarize what I've heard so far. You are upset at the way John treats you. You feel that he berates you at meetings, disagrees with the decisions you make, and takes credit for the work you do. This is making you very upset." This kind of statement tells the person that he is being heard and lets him verify that you didn't miss any of the key points.

Grouping the issues reframes them and draws out underlying issues that might need to be resolved before the surface ones can be dealt with effectively. Here's an example: "Let me summarize what I've heard so far. On the one hand, two of your points seem to be problems resulting from lack of funding: We're short on staff and are dealing with underpowered servers that we can't afford to upgrade. On the other hand, four of your points are related to a lack of training: Josh is hitting the wall in his Debian knowledge, Mary doesn't understand the debugging tools we have, Larry hasn't learned Python, and Sue hasn't transitioned to MagentaSoft yet."

At the end of a meeting, a summary statement might sound like this: "To summarize: The problem is slow server performance. Customers are complaining. We've eliminated the possibility of an overloaded LAN or hot spots on the volumes. Sarah will check with the programmers to see whether their algorithms work well over a high-latency WAN. Margaret will find two clients and verify that they aren't overloaded. We will revisit this issue next week." This lets everyone verify that they all agreed to the same thing. It can be useful to email such summaries to all the participants as a permanent record of the action plan and agreements.

### 51.3.3 Reflection

**Reflection** is a technique to assure people that their emotions are being recognized. This is particularly important when the person you are dealing with is angry or upset. The secret to dealing with angry people is to immediately do something to acknowledge their emotions. You must deal with their emotions before you can deal with resolving their complaints. This calms them down. Then you can begin to deal with the problem in a more rational manner.

Using the reflection technique, you blurt out the type of emotion you see coming from the person. It sounds simple and bizarre but is very effective. (And we promise that you will feel silly the first time you try it.)

Suppose that someone came to your office and yelled, “I’m sick of the way the network is so damn unreliable!” The best response is to reply, “Wow! You are really upset about this!” This is much better than becoming defensive.

First, the person realizes that he has been heard. This is half the battle. Most of his frustration comes from feeling as though he is not being heard. It takes a lot of courage for him to come to you with a complaint. Maybe there have been other times that he was upset about something but didn’t come to you. When he is in front of you and you acknowledge months of frustration instead of denying it or becoming defensive, this will calm him considerably.

Next, the person will realize how he must look. People often get so enraged that they don’t realize how they are coming off. Reflection subtly tells him how he’s behaving, which can embarrass him slightly but will also help him regain his composure.

Finally, the person will start to collect himself. He may reply, “Darn right I’m upset!” This tells you that reflection has worked, because he is inadvertently using the mirroring technique discussed earlier. You can show openness to resolving the problem by saying, “So sit down, and we’ll talk about it.” At this point, he should calm down, and you can productively discuss the problem.

If the person’s emotions are still running high, draw on your active-listening skills—mirroring and summary statements—to discuss the emotional side of the issue. We can’t stress enough that technical issues can’t be solved until you’ve effectively responded to their emotional aspects. Once the person has calmed down, be ready with your mirroring and summary statements to address the technical issue.

## **51.4 Negotiation**

Negotiation is the art of getting what you want. It requires all the communication skills described previously, and then some.

### **51.4.1 Recognizing the Situation**

The first step in negotiation is to recognize that you are in a negotiating situation. It may sound strange, but countless times someone signs a contract and later says, “I should have asked for more money” or “Maybe they were willing to negotiate.” In other words, the person didn’t pause to consider whether he or she should be in negotiating mode. It’s always polite to say the magical incantation, “Is that negotiable?” when talking with customers, vendors, tech support personnel, auto dealers, and even parents.

When your team is in a negotiating situation, communicate this fact to the entire team. Call a meeting and explain what's happening. Knowledge is power, so make sure that the team knows which information should not be leaked.

---

### **Prepare the Team**

An investment firm learned that the high-tech start-up it was talking to needed the funding by a certain date. The firm was able to exploit this information to its benefit and the start-up's detriment. It was able to get a larger share of the company than the start-up owners wanted to give by indicating that anything less would cause delays in getting the funding approved.

How did the investment firm learn this information? Members of the start-up freely gave out the information because nobody had told them not to provide it. This could have been prevented if the start-up staff had been told which information to protect.

---

You also have to be conscious of the power dynamic. Who is the requester? Who is the requestee? Who holds the power in the negotiation? If you are in the power seat, you can control the negotiation. If the other party is in the power seat, you must be much better prepared to defend your requests. The requester is not always the person lacking power, nor is the person receiving the request automatically in power. The power dynamic is not static. It can change unexpectedly.

---

### **Power Can Shift**

Tom was negotiating with a vendor who was in the power seat because Tom's company had a large legacy investment in the vendor's product and couldn't change vendors easily. However, one of the vendor's salespeople let it slip that if the vendor didn't book the sale by the end of the week, it would be counted toward the next year's sales quota. Now Tom was in the power position. He knew that the salesperson would rather receive his commission on this year's balance sheet. When the salesperson wouldn't reduce the price, Tom introduced issues that would delay the contract being signed but promised that roadblocks would be cleared and the contract would be signed before the end of the week if he got the product at the price he wanted. The sales representative caved.

---

### **51.4.2 Format of a Negotiation Meeting**

The general format for an effective negotiation meeting is to define the terms, get agreement on the common ground, and then work on the more difficult parts. It sets a positive tone to settle the easy issues early in this process. If something thought to be easy starts to take a long time, table it until later. Often, you will get to the end of the list of issues only to discover that there is very little disagreement or that the one item that you disagree on can be dropped. In fact, you are likely to discover that both parties are on the same side, in which case the negotiation should be more along the lines of seeking agreement and making commitments.

### **51.4.3 Working Toward a Win-Win Outcome**

It is important to work toward a win-win situation. That is, negotiate toward an agreement that makes both parties successful. It is useless to negotiate vendors down to a price so low that they cannot afford to give you good service or to be negotiated down to the point that you are not getting what you want. A win-lose situation is where you win and the other person loses. This is the second-best situation.

### **51.4.4 Planning Your Negotiations**

Planning is important. Sit down with the people who will be on your side and decide what you hope to gain. Which goals are must-haves? Which ones are you willing to sacrifice? What is something in between?

Decide which elements need to be kept secret and how such issues will be contained. Discuss the power dynamic and how that affects the strategy. Script how the meeting will be conducted, what you are going to say, and what your reaction will be if certain issues are brought up.

It is important to know what you are asking for in specific terms. Nebulous requests can delay negotiations needlessly.

### **Know the Competition**

You should also know who a vendor's competition is, and gently remind the vendor that you have other options. For example, in the early 1990s, Sun Microsystems and Hewlett-Packard were at war for the workstation market. One SA found himself working at a company that used workstations from both companies. He found it useful to put a Sun poster on his office wall and drink coffee from a Sun coffee mug when HP salespeople visited him, and to do the

reverse when Sun salespeople visited. It was a subtle reminder that he could go elsewhere.

### **Do Your Homework**

Doing your homework can lead to a big payoff, especially if the other side hasn't done any. When Tom meets with vendors to renew a yearly maintenance contract, he always brings a stack of printouts, where each page is a trouble ticket that the vendor did not handle well. On each trouble ticket are handwritten notes from the staff person involved describing what happened. Although the mishandled tickets are always a minority, it isn't Tom's fault that the vendor never arrives with data to show this.

The vendor is instead humbled by the physical depiction of its mistakes and more willing to negotiate. At key points in the negotiations, Tom picks a page, reads the note out loud, frowns, and sits quietly until the vendor responds with another concession. None of this would be so effective if vendors would do their homework and arrive with statistics that show they do an excellent job on the majority of the service requests.

---

### **Other Techniques Require Rehearsal**

Before some difficult price negotiations, Tom made his team rehearse the situation in which Tom would be angry enough to walk out of the room for a couple of minutes. His team was to act nervous, be silent for a minute, and then tell the salespeople, "We've *never* seen him so unhappy with a vendor." They then fidgeted nervously in their chairs until Tom returned. Practice makes perfect.

---

### **Remember the Win-Win Goal**

Hardball tactics should be used rarely, if ever. They hurt the relationship between you and the other party. This will burn you in the future. As we said earlier, it is better to strive for a win-win situation. Each negotiation is an opportunity to develop a positive relationship. The future payoff is immeasurable. There is only one time when such hardball tactics should be used, and that is when you will never have to negotiate with that person again.

The last two examples depict exactly that situation: After the negotiations, that salesperson was "out of the loop" until the contracts were up for renewal. These multiyear maintenance contracts virtually guaranteed that years from now, when the renewals were being negotiated, a different salesperson would be involved.

Quarterly updates to the contracts were done through a customer-service hotline whose staff had no visibility to the harshness of the prior negotiations. The salesperson Tom normally dealt with for hardware purchases was not involved in the maintenance contract negotiations, so no bridges were burned there. Finally, Tom knew that he was leaving that position and would not be involved in the next renewal negotiation. If bridges were burned, his successor could use another famous technique: Blame the predecessor to get on the good side of a vendor. “Oh, *he* was a jerk, but *I’m* the nice guy. So let’s begin this relationship with a fresh start.” This trick works only once.

---

### **Use a Variety of Techniques**

Tom had a manager who successfully repeated one negotiating tactic: She was so impossible to deal with that people always gave her what she wanted. He soon realized, however, that this was her *only* tactic. Soon enough, people avoided dealing with her altogether; her career stalled as she became isolated. Be wary of burning too many bridges around you.

---

## **51.5 Additional Negotiation Tips**

The following tips relate to making requests and offers. They are particularly useful during salary negotiations but apply to all negotiations.

- Ask for what you honestly, honestly want.
- After you make a request or an offer, close your mouth.
- Don’t reveal your strategy to your opponent.
- Refuse the first offer.
- Use silence as a negotiating tool.

### **51.5.1 Ask for What You Want**

Don’t negotiate against yourself. Some people start with a reduced request because they are embarrassed by asking for what they want, feel guilty that they want so much, or think that their opponent will think it is unreasonable and will refuse to continue. Don’t be silly! Reducing the request is your opponent’s job; don’t do the other person’s job. You’ll get more respect if you are honest about asking for what you want. You’ll be surprised at how many times the request is accepted. Your job is to ask. The other person’s job is to agree or disagree. Do *your* job.

### **51.5.2 Don't Negotiate Against Yourself**

After you make a request or an offer, close your mouth. You also shouldn't negotiate against yourself when making a request or an offer. People make the mistake of stating a proposal, getting nervous at the silence they hear, and immediately making a concession to sweeten the deal. Your job is to make the offer or request; their job is to accept or reject it. Sometimes, people are silent because they need time to think or because they are hoping to make you nervous so that you will upgrade your offer without even being asked. If silence makes you nervous, fill your mind by repeating the phrase, "The next person to talk is the loser." Give things time and wait for their response.

### **51.5.3 Don't Reveal Your Strategy**

Although you shouldn't be paranoid, you also shouldn't reveal your strategy to your opponent. Don't reveal how low or high you are willing to go, just the offer you are making. Know who is on your side. If a real estate agent, recruiter, head hunter, or other such agent is negotiating on your behalf, the person really represents whoever is paying him. It is always acceptable to directly ask an agent, "Who pays your commission in this situation?" You don't want to be surprised to find out that it is you! If the agent won't tell you who pays, that person is not being ethical. Being told, "Oh don't worry, you don't have to pay a thing," means that the agent is being paid by your opponent. In which case, the agent is an extension of your opponent, so reveal only what you would reveal to your opponent. If the agent receives a commission from the employer, landlord, or whomever, then he or she nominally "represents your position" to the other side but is acting in your opponent's best interest. Therefore, if the agent asks how high (or low) you are willing to go, address that person as you would your opponent: Reveal only your current offer. If the agent demands to know your low and high range so he or she "can negotiate on your behalf," give an artificial range.

### **51.5.4 Refuse the First Offer**

Typically, every *first offer* has built into it some room for movement in case it is rejected. Therefore, you should reject it. This trick works only once. Don't automatically think that if the offer was sweetened once, it can be sweetened again. This isn't a binary search; employers usually don't make a second iteration. If your opponent isn't willing to budge, put your tail between your legs and accept the first offer. This is a risky technique; use it with caution. Tactfully asking if there is some wiggle room in the offer may go over better, while still



achieving the same result.

---

### **Salary Negotiation for Non-negotiators**

Salary negotiations can be nerve-wracking. It is difficult to know if negotiation is allowed. You don't want to upset a potential employer. You might fear you are asking for too much. If negotiating makes you nervous, here is a simple tip: Except for government jobs, almost every salary offer has some head room built in to allow for negotiation. If you don't try to negotiate, you are leaving money on the table. How do you know if this is the case?

When you receive a salary offer, don't give much of an emotional reaction either way. Simply respond by asking, "Is there any wiggle room in this offer?" Wait for the reply. Let the silence work for you.

If there isn't any wiggle room in the offer, you will be told no and you can politely accept the offer. No harm, no foul. The employer's representative probably won't even remember that you asked. However, you'd be surprised at how many times the perspective employer will immediately respond with an increase.

---

### **51.5.5 Use Silence as a Negotiating Tool**

As mentioned previously, being quiet is a critical negotiating skill. Silence from your opponent may simply mean that he or she is thinking, has nothing to say, or is trying to make you nervous. Most people get nervous when they hear silence during a negotiation and respond by offering concessions that haven't even been requested. Another important time to be silent is when you reach an agreement. Sometimes two sides finally get to an agreement, only to have it ruined by someone bringing up new issues. You've got what you were asking for, so shut up!

---

### **Be Careful What You Say**

A woman was moving to a different division for an opportunity that gave her a raise and a promotion. Her new boss told her what the new salary would be and then asked, "Would you like it to be more?" She replied, "Yes." She was dumbfounded that anyone would ask such a silly question. Is there any other answer she could logically give? He should have simply waited for her to sign the paper and offer more money only if she rejected the offer. Now that he had offered to increase her salary and she had

agreed, he had no recourse but to increase the offer on the spot.

---

## 51.6 Further Reading

There are many books on communication and negotiation, but rarely are they specifically directed at IT professionals. The exception to that rule is the excellent book *The Compassionate Geek: How Engineers, IT Pros, and Other Tech Specialists Can Master Human Relations Skills to Deliver Outstanding Customer Service*, 3rd ed., ([Crawley 2013](#)).

The classic books *The One Minute Sales Person* ([Johnson 1991](#)) and *The One Minute Manager* ([Blanchard & Johnson 2015](#)) are full of advice that applies to SAs even though they do not mention system administration directly.

The negotiating techniques mentioned in this chapter worked for us, but we aren't high-powered negotiation experts. Luckily, some real experts have written books on the topic. We recommend *The Haggler's Handbook: One Hour to Negotiating Power* ([Koren & Goodman 1991](#)) as a good general-purpose book. It has the benefit of providing one tip per page. If you read one page per day while getting dressed in the morning, you will be a much better negotiator in a matter of weeks.

Related to negotiation is assertiveness. We recommend *When I Say No, I Feel Guilty* ([Smith 1975](#), [Smith 2000](#)), especially if you find yourself agreeing to take on additional work that you later regret. The more general *The Assertiveness Workbook: How to Express Your Ideas and Stand Up for Yourself at Work and in Relationships* by Paterson ([2000](#)) includes advice and exercises to build skills that will help you in both your work and personal lives.

## 51.7 Summary

Good communication skills are required to be a successful system administrator. Different situations require different communication tools: There are times when we need to communicate our needs, listen well to what others are saying, work with someone, or stay out of someone else's problem.

We discussed communication skills such as "I statements" to make yourself heard, mirroring to confirm that you understand people, reflecting to deal with emotional people, and summary statements to verify that members of a group are in sync. These tools can help you deal with the four kinds of problems in the world: mine, yours, ours, and other people's. These communication tools are useful in your work, but they also are key to your personal life. In fact, they are the tools that are taught in marriage counseling. We hope that reading this

chapter improves your relationships inside and outside of work.

System administrators often find themselves in negotiations with vendors, management, and customers. It is important to recognize you are in a negotiating situation so you can prepare, plan, and strategize. It is important to work toward a win-win situation, because it builds a strong relationship with the person in ways that will benefit all parties in the future.

Everyone can get nervous when negotiating salary. There is usually some wiggle room in salary offers. If you don't (gently) ask, you'll never know.

## **Exercises**

1. Give an example situation and desired communication technique useful for dealing with the following types of problems: my problem, your problem, our problem, and other people's problem.
2. What is active listening? What are some of the techniques of active listening?
3. Which types of communication or interpersonal skills training are available to you?
4. How confident are you in your negotiation skills? How much negotiating do you have to do? How could you improve your skills?
5. Describe your last negotiation and what you could have done to improve it.
6. Pair up with a peer and role-play salary negotiation. What did you learn from doing this?

## Chapter 52. Being a Happy SA

This chapter is about being a happy system administrator. Happiness means different things to different people. A happy SA deals well with stress and an endless workload, looks forward to going to work each day, and has a positive relationship with customers, coworkers, and managers. Happiness is feeling sufficiently in control of your work life and having a good social and family life. It means feeling like you're accomplishing something and deriving satisfaction from your job. It means getting along well with the people you work with, as well as with the management above you.

Just as happiness means different things to different people, so various techniques in this chapter may appeal more to some readers than to others. Mostly, we've tried to list what has worked for us.

The happy SAs we know share certain habits: good personal skills, good communication skills, self-psychology, and techniques for managing their managers. We use the word *habits* because through practice, they become part of us.

These behaviors come naturally to some people but need to be learned by others. Books, lectures, classes, conferences, and even training camps teach these techniques. It's pretty amazing that happiness comes from a set of skills that can be developed through practice! Making a habit of a technique isn't easy. Don't expect immediate success. If you try again and again, it will become easier and easier. A common rule of thumb is that a habit can be learned if you perform the behavior for a month. Start today.

The goal of this chapter is to give you a taste of these skills and techniques and then refer you to resources for a more complete explanation.

### 52.1 Happiness

Cognitive theorists believe that being happy or sad is not driven by whether good or bad things are happening to people, but rather by how they react to what is happening around them. How can this be? Again, we return to the concept of the critical inner voice. Some people can shut down that voice when they need to; others pay too much attention to it.

For example, suppose that a tree falls on someone's house. One person might think, "Of course it fell on my house; I don't deserve a safe home." Someone else might think, "I'm glad nobody got hurt!" and look forward to the

opportunity to redecorate once the repairs are complete.

The opposite situation can also be true. You typically would think that getting a raise would be a good thing. However, it might introduce a big barrel of worries for some people: “I’m already working as hard as I can; now they’ll expect even more. I’m doomed to fail!” The book *The Feeling Good Handbook* ([Burns 1999](#)) gives further examples, as well as some excellent solutions. We highly recommend this book.

A little insecurity is normal and healthy. It keeps people out of harm’s way and encourages people to “measure twice, cut once.” However, too much worry can cause problems.

Luckily, you can retrain yourself. The first step is to recognize that this critical inner voice exists. People can be so accustomed to it that they believe it without pausing to evaluate what it is saying. Once you have recognized that it is speaking, pause to think about what it is saying. Consider the source. Is it simply doubting everything? Is it viewing the world as black and white? Is it repeating negative things you were told by outsiders?

---

### **Consider the Source**

One SA had a major breakthrough when he realized that his critical inner voice was always saying negative things that his hypercritical mother had said to him when he was young. In fact, the voice sounded like his mother’s voice! He realized that these thoughts were simply echoes of the extreme negativity he had experienced as a child and were not useful bits of advice. He set out to develop the habit of ignoring those thoughts until they disappeared. It worked!

---

Retraining yourself is not easy, but it can be done successfully. Many people choose to do it with the help and guidance of a therapist. Others do it on their own using books such as *The Feeling Good Handbook* by David Burns, which includes a large number of techniques and a useful guide to selecting the ones that are right for you. Take advantage of the confidential employee assistance program (EAP) if your employer provides one as part of its mental health benefits package.

## **52.2 Accepting Criticism**

Nobody is perfect, so it is important to take criticism well. Everyone receives criticism now and then. Some people interpret all comments as criticism; others

let the smallest criticism wreck their self-esteem. That's not good. However, if you take criticism positively, it can help you change your behavior so that you improve yourself. Criticism is a good thing: It prevents people from repeating mistakes. Imagine how terrible it would be if everyone made the same mistake over and over again! Rather than accepting criticism with disdain for the critic, it is healthier to thank the person for his honesty and think about what you can do better in the future.

It is important to distinguish between constructive and nonconstructive criticism. Nonconstructive criticism hurts feelings without helping the situation. Be careful of the nonconstructive criticism you give yourself: Don't "should have" yourself to death. "Should" is a scolding word. When you think to yourself, "Oh, I *should have* done such and such," you are scolding yourself about something you can't control: the past. It is much better to replace "I should have" with "next time, I will."

## **52.3 Your Support Structure**

Everyone needs a support structure. Everyone needs someone to talk with now and then. Your support structure is the network of people you can go to when you need to talk about a problem. Having different people you can go to for advice on office politics, technical issues, and general life problems is important when you feel that you are over your head. It takes time to develop these relationships. Sometimes, the right person is your spouse or significant other, a friend, a coworker or manager, or even an email list of people who share a common interest.

It is important to ask for help. We find that SAs tend not to be very good at seeking help for personal problems, but instead are likely to let a problem build up until they feel like exploding.

Maybe this proclivity is related to some "macho" culture of being self-sufficient. Maybe because they are expected to solve problems before their customers notice them, SAs expect other people to know when they themselves have problems. Maybe the tendency to bottle things up arises because SAs are expected to solve technical problems on their own, and then they carry that into their personal lives. Even when SAs do reach out for technical help, it is often to nonhuman resources: web pages, FAQs, and manuals. Even asking for help on mailing lists has an air of not talking about your problems face to face.

Successful people know that it is not a weakness to ask for help. In fact, people respect someone who takes responsibility for getting help. It creates less

of a burden on others to deal with a problem when it is small rather than after it has escalated into a full-blown emergency. Most important, problems are solved more quickly when many people work on them. Share the wealth! Friends help other friends. It's like a bank account: You make a deposit when you help your friends, and you shouldn't feel bad about making a withdrawal every now and then.

---

### **Should Have Asked for Help**

Everything would have been better if one SA had asked for help. He was going to present a paper at a very large SA conference. When he didn't show up at the designated time, 15 minutes before his presentation, the coordinators went through a lot of pain to reorder the other speakers. He did show up just moments before he was to speak, when the session chair was on stage introducing the replacement speaker.

He was late because he had brought only overhead transparencies rather than his laptop with the presentation on it. Seeing that all the other speakers were projecting directly from laptops, he assumed the conference didn't have a way to present using transparencies. Instead of asking one of the conference coordinators for help, he got permission from his boss to rent a laptop for a large sum of money. The rented laptop ran only Windows, and his presentation had been written under Linux, so he then spent several hours retyping the presentation into PowerPoint.

Had he asked for help, the coordinators would have been able to find a transparency projector or would have been easily able to find a Linux laptop for him to use. Instead, he created a lot of stress for himself and others and spent a large amount of money to procure a temporary laptop. He should have asked for help.

---

We have other similar anecdotes that involve personal issues, such as finance, health, relationship and family problems, and even drug and alcohol abuse. In every case, the person's friends wished they had been called on sooner. That's what friends are for.

## **52.4 Balancing Work and Personal Life**

Finding balance between work and personal time is important to mental health. Although it can be gratifying to be a hard-core techie who works day and night, burnout will eventually become a problem. Taking time for yourself is key.

Taking breaks during the day, getting regular sleep, having a social life outside of work, and not working yourself to death are all critical habits to develop.

Treat your significant other with the respect he or she deserves. Many SAs work so many hours that their significant others become “technology widows or widowers.” That shows little respect for them. Family time is important time; take time for your family. Schedule it in your datebook. Give your family the thanks and admiration they deserve. Put pictures of them on your desk so you are always reminded that you do it for them, just like Homer did on *The Simpsons* episode 2F10, “And Maggie Makes Three” ([Crittenden 1995](#)). The most valuable thing you can give your family is time. Nobody’s last words have ever been, “I wish I had spent more time at the office.” Of course, when we say “family,” we’re using a very wide definition, including biological and chosen families.

Respecting your body is also important. Listen to your body. If you are tired, go to sleep. If you are hungry, eat. If you aren’t feeling well, help your body repair itself. It’s ironic that we often meet people who take care of immense networks but don’t know how to take care of their own bodies.

Your employer gives you vacation time. Take it; the company gives it to you so you won’t burn out and then be completely useless. Long ago, employers discovered that vacations benefit both the employer and the employee.

You don’t do yourself or your company a favor by skipping vacations. Many times, we’ve heard from people who prided themselves for not having taken a vacation in years: “The company can’t live without me.” Or they claim that skipping vacations shows your dedication. Actually, the opposite is true. If you don’t disappear for a week or two occasionally, there is no way to discover whether procedures are documented properly or that your fallback coverage is properly trained. It is better to learn which coverage is missing during a vacation that you return from rather than when you quit or, heaven forbid, get hit by a bus.

## **52.5 Professional Development**

Professional development means receiving the training required to maintain and improve your skills. It also means associating yourself with the professional organizations of your field.

There may be professions that don’t change much and don’t require keeping up with the latest technology and techniques. System administration isn’t one of them.



Learning new skills helps in your career and is fun. SAs tend to rank “learning new things” high on their list of fun things. Never turn down the opportunity to learn.

Reading can keep you up-to-date with new technology. There is a constant flow of new books, blogs, Reddit posts, magazines, and trade and academic journals. Also subscribe to the major trade journal of the industry in which your customers work so that you are familiar with the industry trends that concern your customers.

One-day workshops and training programs serve a different purpose than week-long conferences. One-day seminars tend to be *tactical*—that is, focused on a particular technology or skill. Week-long conferences are *strategic*, offering opportunities to discuss broader topics, to network, to build community, and to further the craft of system administration as a respected profession. Week-long conferences have a powerful effect, providing a much-needed opportunity to relax, and they provide a supportive environment where you can take a step back from your day-to-day work and consider the big picture. Attendees return to their jobs brimming with new ideas and vision; they’re refreshed, motivated, and operating with a new outlook.

Although this book does not endorse particular products, we can’t hold back our enthusiasm for the USENIX LISA (Large Installation System Administration) conferences. Volunteering, writing papers, submitting articles to its newsletter, helping plan conferences, and speaking at its meetings can go a long way toward developing your reputation and career. It certainly helped us.

## 52.6 Staying Technical

We often hear SAs complain, “They’re trying to turn me into a manager, but I want to stay technical!” They ask for advice about how *not* to be promoted. If this is your situation, there are a couple of things to remember. First, if your manager is trying to promote you to management, he or she means it as a compliment. Don’t be offended. Although some SAs have negative opinions of management, managers tend to think that management is a good thing. Maybe your manager sees great potential for you. Take the suggestion seriously and consider it for a while. Maybe it is right for you. Do, however, remember that becoming a manager does mean giving up technical responsibilities; don’t feel that you’ll be able to retain your technical duties and adopt new responsibilities.

Some SAs slowly slide into management, adopting slightly increasing management responsibilities over time. Suddenly, they realize that they are

managers and don't want to be.

To prevent this, you must be aware of which kind of tasks you are performing. If you want to stay technical, it is important to discuss this with your boss. Establish that there is a difference between being the “technical lead” of a group and being the “manager” of a group. Come to agreement on where the line is drawn—a litmus test, so to speak. Then you can both be aware of the issue. For example, a *technical lead* is part of the technical process of architecting and deploying a new system. *Management* tasks usually involve budgets, salary and performance reviews, and other human resources issues. We'll discuss career management in more depth in [Section 52.9](#).

## 52.7 Loving Your Job

Happy SAs love their jobs. This isn't an accident. They didn't fall into jobs that they love; they worked in many jobs in many types of companies and started to realize what they liked and didn't like. They then could become more focused when job hunting. It can take years to figure out what motivates you to love your job and find a job that provides those qualities, but it is something to think about as your career evolves.

Your manager affects your ability to love your job more than the kind of work you do. *A bad job with a great boss is better than a great job with a bad boss.* Suppose that your job was the best, most fantastic job in the world. For example, suppose that you were being paid to eat chocolate all day. If your boss was a jerk, you would still hate your job. Conversely, if you had a terrible job, a great boss would find a way to make it enjoyable. Our personal experience is that most people leave their jobs not because they don't enjoy their work, but because they didn't like their bosses.

In the 1999 film *Office Space*, Peter Gibbons tells a story about how his high school guidance counselor would ask students what they would do if they had a million dollars and didn't have to work. Whatever their answer, this was what he would suggest they'd do for their career. For example, if you wanted to fix up old cars, then you're supposed to be an auto mechanic. Maybe you are an SA because you would spend your time playing with computers. What are the aspects of computing that you enjoy so much? Consider integrating those things into your career.

---

### Following Our Own Advice

Christine has been a fan of Formula 1 racing since she can remember and

has always wanted to work in that industry. She decided that it was time to pursue that dream; after the first edition of this book came out, she studied aerodynamics with the intention of going into race-car design. After receiving her degree, she got hired onto a Formula 1 race team. She loved her job and was glad that she took the risk of making a career change. Ten years later she returned to IT, but she will never regret those ten years of Formula 1.

Tom has always wanted to be more involved in politics. In 2003, he quit his job and worked on a presidential political campaign. He found it a very interesting and fulfilling experience and will look for other opportunities to get involved in political campaigns that he believes in.

---

## 52.8 Motivation

Being motivated about your job is no accident. Satisfying, long-term motivators are different for different people. Money motivates people, but only in the short term. It doesn't sustain motivation very well. Some people are motivated by the good feeling they receive after helping someone. It sounds simple, but helping people is habit forming. The good feeling you get from knowing that you've helped someone is so powerful that once you've had a taste of it, you crave it even more. You want to return to that good feeling, so helping people becomes even more important, and you strive to help even more people. This is highlighted in the 1988 film *Scrooged* when Bill Murray's character realizes how good it feels to help people. He says, "And if you like it and you want it, you'll get greedy for it! You'll want it every day of your life, and it can happen to you."

The compliments people receive are habit forming in the same way. A compliment propels you forward. Imagine every compliment you get from your boss propelling you, motivating you to continue to achieve great things.

The problem is that those compliments take a long path between your ear and the part of the brain that *accepts the compliment*. Somewhere in that path is a minefield known as your *critical inner voice*. Sometimes, that voice reaches up, grabs that compliment in midair, and pours toxic waste on it. Then that compliment is tainted. By the time it reaches its destination, the toxic waste has corrupted the compliment into something that hurts you. Thus, instead of a stream of incoming compliments that propel you, you have a stream of negatives that sap your energy.

For some people, this critical inner voice is a small, manageable little beast. For some, it is a loud, bellowing giant. Therapy can help manage that giant by

helping you deal with the source of the problem, be it an overly critical parent, an overbearing significant other, or shame.

Shame comes from feeling bad about something, and holding those feelings inside rather than letting them out. People often feel that personal problems should stay at home and not be discussed at work, but bottling up these problems can be unhealthy and wreck your productivity. Suppose that one of your parents is ill and that you haven't shared this information or how it makes you feel with your coworkers. The positive feedback you receive should make you feel good and motivate you, but instead the toxic shame of, for example, feeling that you aren't visiting your ill parent enough negates the compliment: "Oh, they wouldn't have given me that compliment if they knew what a terrible child I am." Now what had been intended as a compliment is poisoned.

It is important to learn to accept compliments. When people deflect compliments, they do a disservice to themselves. People tend to reply to a compliment with some sort of demurral: "Oh, it wasn't a big deal" or "I did a small part; Margaret really did all the work." If someone is being polite enough to compliment you, be polite enough to accept the darn compliment! If you aren't sure what to say, a simple "Thank you!" will suffice.

Shame can take other forms. Fears of racism, sexism, or homophobia can hold people back from reaching their full potential. You might invalidate compliments you receive if you feel that your manager is biased against your sex, race, religion, sexual orientation, or gender identity. You can turn these issues around by discussing these fears with your coworkers and working to gain a better understanding and appreciation for your differences. If your corporate culture discourages openness about personal issues, you may find it useful to at least open up privately to someone, such as your boss or a close coworker.

---

### **An Unsafe Workplace Is an Unproductive Workplace**

A bisexual SA lost a week's worth of productivity because he overheard a coworker in the next cubicle saying that "queers should all be killed." Would it be safe for him to walk to his car after work if this coworker found out he was bisexual? Would the coworker sabotage his work? Every time he tried to work, the memory of his coworker's words distracted him. The next day, he brought up this issue to his boss, who refused to talk with the coworker or move the SA's cubicle. Eventually, the SA left the company.

The manager could have saved the cost of recruiting and training a new

staff person if he had taken the time to make it clear to the coworker that such comments are inappropriate in the workplace and that the company valued people based on the quality of their work, not their race, sexual orientation, gender, or other non-work issues. The manager could have also explained that the diversity of the group was what made it strong. The employee could also have asked for help by escalating the issue to HR, who probably has more experience mediating such situations.

---

Finally, we have one more recommendation for maintaining positive self-esteem and loving your job. Maintain an “accomplishment wall,” where you post all the positive feedback you receive: a note from a customer that says thanks, awards you have received, and so on. Make sure that these items are in a place that you see every day, so that you have a constant reminder of the positive things you’ve done. If you are the team leader, you might consider having such a wall for all the team’s accomplishments located in a place that the entire team will see. When morale is low, you can look at the wall to remind yourself of times when people have said good things about you.

---

### **Electronic Accomplishment Wall**

Much positive feedback is received via email. Save every thank-you message you receive in an email folder named “feathers” because they are the feathers in your cap (thanks to Tommy Reingold for this term). When you write your yearly accomplishments list, you can review this folder to make sure that you didn’t forget anything. On days when you are depressed or things aren’t going well, pop open this folder and remind yourself that people have said good things about you.

---

## **52.9 Managing Your Manager**

You may think that it is your boss’s job to manage you, but the reverse is also true. You must manage your boss—steer him or her toward what will make you happy.

The first part of managing your boss is to make your needs known. Managers can’t read your mind, so don’t get upset when they don’t guess what you want. At the same time, recognize that you are not the only thing on your manager’s mind. Respect that by not going overboard and pestering your manager. Strike a balance.

To get your boss to do what you want, you need to make sure he or she is able

to do his or her own job. Your boss has a job, too. Your boss's performance is measured by whether certain goals are achieved. These goals are too big for any one person to complete alone—that's why you exist. Your assignment is to do a bunch of tasks that equal a small fraction of your boss's goal. Your fraction, plus the fractions of all your coworkers, should complete those goals. Some people think of their jobs as the tasks they are assigned. That's not true. Your job is to make your boss a success, and by doing your job you contribute to that happening. Amazingly, your boss is in the same situation. Your boss has been assigned a small fraction of what his or her boss needs to accomplish. The boss's boss and everyone up the chain all the way to the CEO are in this situation. The total of all these little fractions is one big success.

Why should you care about your boss's success? First, successful bosses are given leeway to do what they want, and to take on more interesting projects. That means you'll get more flexibility and more interesting projects. A successful boss gets promoted. If you helped create that success, the boss will take you along.

A manager has a limited amount of time and energy, which will be expended on the people who are most likely to help him or her succeed. To manage your boss, you are going to need your boss's time and energy. Obviously, a manager is going to respect the wishes of a star performer more than those of a slacker.

---

### **Raise-Time Humor**

A group of SAs were talking about the recent round of raises. A person who didn't get a great raise complained about someone rumored to have gotten a very good raise.

He griped, "That guy always gets big raises because he just does whatever our manager tells him to."

Someone responded, "How's that do-the-opposite-of-what-our-boss-wants strategy working for you?"

---

One need you should communicate, perhaps once or twice a year, is your career goal. This doesn't have to be a 20-page document supporting the reasons for your request, but it should not be simply mentioned in passing, either.

---

### **Steer Promotions to You**

Tom claims he never received a promotion for which he didn't directly ask. In college, he was a student operator at the university computer

center. One day, he walked into the director's office and stated, "I want you to know that I want to be one of the student managers here, and I'll do what it takes to get there." He was told that if he worked hard and was on his best behavior for the entire summer, he would receive the promotion before the new school year. He worked hard and was on his best behavior and received the promotion. History repeated itself in his future jobs.

---

Putting an idea in a manager's ear means that the next time the manager needs to fill an open position, you will be a potential candidate. A good manager will immediately start coaching you to groom you for that position, testing the waters, and watching whether you show promise to successfully fulfill the role you have requested. The manager can structure your tasks and training in the right direction.

If you are unsure of your career goals, you might communicate a technical skill you want to develop. If you want to stay where you are, be sure to let your boss know that, too!

Another steering technique is to let your boss help you with time management. When you have a completely overloaded schedule with no end in sight, let your boss set your priorities. Don't bring a complaint about being overloaded. Managers receive complaints all day long and don't want another one. Instead, bring your to-do list annotated with how long each item should take to complete. Explain that the total time for these projects is more than your 8-hour day (or 40-hour week), and ask for help prioritizing the list.

A typical manager will have several positive reactions. First, it's quite a compliment that you are seeking the manager's wisdom. Second, it makes the manager happy, because after a day of receiving selfish request after selfish request, you have come to your manager with a message that says, "I want to meet your top goals, boss; tell me what they are." This can be very refreshing! Finally, this gives your manager a clear view into which kind of work you do. Your manager may notice tasks that should be completely eliminated or may reduce your load by delegating tasks to other people in your team. Maybe that other team member made the manager aware that he wanted more of a particular kind of assignment, and this is your boss's opportunity to give it to him.

---

### **Case Study: Pleasing the Visionary**

An SA joined a university to fix a crumbling, unstable heterogeneous



network that needed the most basic of upgrades: quality wiring, modern switches and routers, uniform OS configuration, and so on. The dean thought himself to be quite a visionary and wasn't interested in projects with such little flair. He wanted futuristic projects that would bring higher status and acclaim, such as desktop video and virtual reality systems.

None of those projects could possibly happen until the basic upgrades were done. The SA couldn't get any of the fundamental problems fixed until he started explaining them to the dean in terms of how they were the steps along the way to achieve the dean's futuristic goals. Smartly, he didn't explain these steps as preconditions to the dean's goal, but rather positioned them as the steps that achieved the dean's goal.

---

Finally, there's the concept of **upward delegation**, or delegating action items to your boss. Certain tasks are appropriate for upward delegation, and others are not. Your manager should be concerned with steering the boat, not rowing it. Don't delegate busywork upward. However, upward delegation is appropriate for anything that you feel you don't have the authority to do. Creating an action item for your boss is most appropriate when it will leverage the manager's authority to make other action items go away. For example, you might be having a difficult time creating ad hoc solutions for individuals who need backups done on unsupported hardware. Your boss, however, might have the authority to create a policy that only officially supported servers are backed up, with all other requests considered new-feature development projects to be prioritized and either budgeted or rejected like all new-feature requests. By using his or her authority to make or reiterate policy, the manager is able to remove an entire class of requests from your plate or possibly get funding to do the request properly.

Sometimes, it is obvious when upward delegation is appropriate; at other times, it is not. When your manager asks you to do something, replying, "No, I think you should do it" always looks insubordinate. For example, if your manager asks you to give a presentation to people on a certain topic, a lousy reply is "I'd rather you do it." A better reply is that you think this information will be better received if it comes from the horse's mouth. That is leveraging the boss's authority. (You might then be asked to write the presentation, but the boss will present it.)

Upward delegations create more work for your boss, who is usually in a position to delegate work downward. For that reason, you should be careful with the quantity and timing of upward-delegation attempts. Don't do them



continually; don't make them at inappropriate times or during a heated discussion.

When should you make a request of your boss? Timing can be critical. You might keep a mental list of items you want to ask for and pull out the most important one when you receive a compliment. It's difficult for a manager to turn down a *reasonable* request made immediately after having complimented you on a job well done or thanking you for saving the company money.

---

### **Use Your Boss's Power Wisely**

A group of SAs responsible for deploying PCs was missing deadlines because of an increase in custom configuration requests. These special requests required extremely large amounts of time. It turned out that many of these requests were *not* work related. One request was for the SA group to configure a MIDI (synthesizer controller) card, something that was not required for the person's job. Another request was to create a dual-boot PC on which one of the partitions would be an OS that was not officially supported but had better games available for it. Both of these requests were against policy. The SAs delegated to their boss the task of explaining why these requests were being rejected—that is, the SAs were not there to help with the staff's hobbies or entertainment. The manager was able to use his authority to save time for an entire team of SAs.

---

## **52.10 Self-Help Books**

There are many excellent self-help books that can help you with the various topics discussed in this chapter. We've already mentioned some of them, such as *The Feeling Good Handbook* ([Burns 1999](#)).

If you've never read a self-help book, it's difficult to imagine that a stack of paper with writing on it can solve your problems. Let us lessen that skepticism right here. Self-help books help millions of people every year! But let's also be realistic: Only you can change you. Books offer suggestions, advice, and new ways of looking at what you've been seeing all along. Not every book will be the right one for you. Maybe the way the author addresses the subject, the particular problems the book addresses, or the writing style isn't the right match for you. That is why you can find a dozen self-help books on any given topic: Each has a different style to appeal to different people.

We also recommend that you think critically about advice being offered in a book before you try it. Be suspicious of any book that professes to fix all your

problems. If it seems too good to be true, it probably is. All behavior modification requires work, so disbelieve a book that claims otherwise. If it requires you to promote the techniques to others, be concerned, because a technique that works well doesn't require a pyramid scheme to sell it.

We have tried to recommend timeless classics that have been on the market for a long time—books that have developed solid reputations for being effective. If you still doubt the usefulness of self-help books, put down the one you're reading right now.

## **52.11 Summary**

It is important to be happy. Happiness does not come from good things happening or bad things not happening, but because we have created the support infrastructure that lets us celebrate the good times and be resilient during the bad times. Skills that help enable happiness include accepting criticism, having a support infrastructure, and maintaining a good work/life balance.

Professional development is especially important in the continually changing high-tech field of system administration. One-day tutorials tend to be tactical (skills); week-long conferences tend to be strategic (vision). Both are useful and important.

You want to love your job and be happy with it. That means maintaining good mental health, balancing stress, handling criticism, and taking care of yourself. You don't do anyone a favor by skipping vacations.

Managing your boss is a component of ensuring your happiness. This involves paying attention to your boss's priorities so that he or she will pay attention to yours. Make your needs known. Attend to your manager's success. It is appropriate to delegate to your boss action items that leverage his or her authority to solve problems for the people who work for him or her.

The average person spends the majority of his or her waking hours at work. You deserve to be happy when you are there.

## **Exercises**

1. Imagine that you've won the lottery and no longer have to work. What would you do? Why aren't you doing it now? How could you be doing it now?
2. What are your primary outlets for professional development? What kind of support do you receive from your employer for this?

3. Are you an optimist or a pessimist? Give two examples.
4. How accepting is your workplace to you being open about your personal life? If your workplace does not condone this, who can you go to for support when you need it? Is your workplace safe?
5. Describe your support network.
6. When was your last vacation? Did you read your work email account during it? If you did, was it really a vacation? Do you promise not to read email on your next vacation?
7. Spend 15 minutes creating your awards wall.
8. Are your priorities in alignment with those of your boss? How do you know? How do you ensure that they are?
9. What is the next promotion you want? Who knows that this is your goal?
10. Describe the last time you needed to use upward delegation. How did your manager react? What will you do in the future to improve this?
11. What's your favorite self-help book? Why?

## Chapter 53. Hiring System Administrators

This chapter looks at hiring system administrators, focusing on areas that are different from hiring for other positions. These are the things that the SA staff and managers need to know. Everything else should be taken care of by the human resources department.

This chapter does not cover areas of hiring that are generic and apply across the board, such as compensation packages, oncall compensation, overtime pay, comp time, or traditional incentive schemes, such as stock options, hiring bonuses, and performance bonuses. Nor does this chapter look at how location, lifestyle, working from home, facilities for remote work, or training and conferences can factor into the hiring process.

Rather, this chapter covers the basics of how to recruit, interview, and retain SAs. It also looks at some things a company can do to stand out as a good company for which to work.

The hiring process can be simplified into two stages. The first stage is to identify the people whom you want to hire. The second stage is to persuade them that they want to work for you.

Identifying whom you want to hire is in many ways the more complicated stage. To determine whether you want to hire someone, you must first know what you want the new hire to do and how skilled the person needs to be. Then you need to recruit a team to gather resumes from interested, qualified SAs. Then you need to pick the interview team and make sure that they are briefed appropriately on the interview process and determine which interviewer will focus on which topics. The interviewers need to know how to interview the candidates in the particular areas of interest and how to treat the candidate during the interview process.

The second stage, persuading the candidate to work for you, overlaps a little with the first stage. The most important part of recruiting the candidate is doing the interview process well. The interviewers should show the best of the company and convey the position to the candidate. The experience should make the candidate want to work there. The timing also needs to be right; move quickly for the right candidates. Later in the chapter we provide tips on how to avoid having to hire new employees by retaining the staff you have.

In addition to this chapter, we recommend the article “Technical Interviews” ([Chalup 2000](#)) for advice on the interview process itself, plus the concise but

thorough book *Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent* by Spolsky ([2007](#)), which explains technical recruiting and hiring in general.

### 53.1 Job Description

The first step in the hiring process is to determine why you need a new person and what that person will be doing. Figure out which gaps you have in your organization. [Chapter 48](#), “[Organizational Structures](#),” describes how to build a well-rounded team. That discussion should help you decide with whom the new hire should be working and which skill sets the person should have to complement the existing team's skills. Then think about which roles you want the new employee to take on, as described in [Appendix B](#), “[The Many Roles of a System Administrator](#).” This should give you an idea of the personality type and the soft skills for which you are searching. From this list of requirements, figure out which are essential and which are desired. You now have a basis for the **job description**.

Although it's a time-consuming task, write a job description specific to every position. A written job description is a communication tool. Writing it is a catalyst that encourages team members to express their visions for the position, resolve differences, and settle on a clear definition. Once written, it communicates to potential candidates what the job is about. During the interview, it serves as a touchstone for the interviewers. When having to decide between two similarly qualified candidates, the job description serves to focus the decision makers on what is being sought.

A job description should include two parts: a list of responsibilities the person will have and a list of skills the candidate should have. To make it easy to write and easy to read, make each job description a bulleted list, with the more important items at the top of the list. However, do not confuse a job description with a **job advertisement**, which is the job description revised for external consumption plus contact information, and any other legally required statements.

In [Section 31.1](#), we mentioned that documenting the tasks that you dislike doing makes it easier to delegate them. The next time you get the opportunity to hire, you can use those documents to help create the job description. The list of tasks you documented—and want to delegate—can be added to the list of responsibilities. The skills required to perform those tasks can be added to the list of required skills. The job description practically writes itself.

Two competing philosophies are related to filling an open position. Hiring the

skill means looking for someone with the exact skills specified in the job description. By contrast, hiring the person means looking for quality people and hiring them, even if their specific skills just overlap with what's listed in the job description. If you "hire the person," you are hiring the individual for his or her intelligence, creativity, and imagination, even if some of the specific credentials associated with the job you're trying to fill are lacking. These people may surprise you by changing a job so dramatically that the effort required is much less and the results are much better, perhaps by eliminating unnecessary tasks, automating mundane tasks, and finding publicly available resources that accomplish the task.

It is easier to "hire the person" when you have many open positions, each requiring a different set of skills. If the first quality person you find has a few skills from each job description, you can redefine the position around that person. The next person you look for should have the remaining skills.

We tend to find ourselves hiring mostly senior-level SAs in such large quantities that we almost exclusively "hire the person." We recommend "hiring the skill" only for specific tactical instances.

---

### **Case Study: When to "Hire the Skill"**

At Bell Labs, Tom's SA team "hired the person" for certain roles and "hired the skill" for others. For example, the senior SAs were expected to find creative solutions for the problems at hand. Those positions called for "hiring the person."

In contrast, a league of technicians who deployed new PCs needed only to start automated procedures to load the OS and then deliver the fully configured computers to the right office. For that role, "hiring the skill" worked best.

The "hire the person" strategy works only within limits. Make sure that you have something for the person to do. One Silicon Valley computer company had a strategy of hiring people because they were great, on the assumption that they would find projects on their own and do wonderful things. Instead, many of these people became lost, felt unwanted, and grew extremely unhappy. Such a strategy works well for hiring researchers doing transformational work but not when hiring SAs.

---

Some candidates want to be very clear about what the job is before they decide that it is something that they want to do and will enable them to advance

their career goals. These candidates will reject an offer when the job duties are vague, or are unclear to them. Other candidates want to know whether there is flexibility in the job, so that they can gain experience in a variety of areas. In both cases, the interviewers need to know what the job entails and where it can be flexible, so that both sides know whether the candidate will be suited to the job. Be flexible if multiple positions are available. Some candidates may be perfect for part of one job and part of another. To maximize everyone's options, the interviewers should be aware of all the requirements in all the open job descriptions, so that they can talk to the candidate about other possibilities, if appropriate.

## 53.2 Skill Level

After building the basic job description, you should have a reasonable idea of the skill level required for the position. Ideally, you should map the job requirements to the USENIX LISA skill levels described in the booklet *Job Descriptions for System Administrators* ([Darmohray 2012](#)). This has become the standard way of communicating this information in job descriptions, advertisements, and resumes.

The skill level of the person you hire has economic implications. To save money, some organizations believe in hiring the lowest-possible skill for first- and second-tier support. The theory is that the most common position to be filled should cost the least, using minimally trained people.

Other organizations prefer to hire people who are smart and highly motivated and are on the verge of being overqualified for the position. These people will be bored with the repetitive tasks and will eliminate them by instituting automation or recommending process changes that will continually improve the organization. In essence, these people will gladly put themselves out of a job because they know that their motivation will lead to other positions in the organization.

A talented, motivated SA who verges on being overqualified is worth more than two unmotivated, unskilled SAs who are hired to make up the numbers. The ideal candidate for the more junior position is bright and interested and has recently entered the field. If you invest in such people and give them opportunities to learn and do more, their skill levels will rise rapidly, giving you ideal candidates for promotion into more senior positions that can be more difficult to fill. If you are lucky enough to hire people like this, make sure that their salaries and bonuses keep pace with their rising skill levels, even if it means pushing through special raises outside review periods. These people are

valuable and difficult to replace, and they will be hired away by another company if you do not keep pace. Giving them extra raises outside of review periods also builds loyalty for the long term.

When rebuilding a struggling team, it is important to hire reform-oriented, senior-level people who have experience doing things “the right way” so that they are more likely to work to a higher standard. When a team is doing well, you have the luxury of hiring a mixture of senior and junior people to fill various specializations. The anecdote “[When Things Are Dire, Hire Higher](#)” in [Section 48.4](#) has advice and examples related to this issue.

### 53.3 Recruiting

Once you have decided which position you are trying to fill and which skill level you are looking for, you need to find appropriate candidates for the job. The best candidates are usually found through personal recommendations from members of the SA team or customers. When demand is high, the best people already have a job, and must be recruited away from their existing position. They probably aren’t thinking of changing jobs until you mention the possibility. Even then, it can take months of relationship building before they consider interviewing.

System administration conferences are often good places to recruit candidates. It may be possible to do some interviewing at the conference, and it is also possible for the candidates to talk to members of the SA team to find out what they are like and what it is like to work at the company. SAs who attend these conferences are often good candidates because they are keeping up with what is happening in the field and are interested in learning.

There are many web sites for advertising job openings. Many are general, but some focus on technology jobs. You’ll get the best results by sticking with the ones focused on technology-related jobs. (We would be remiss if we didn’t mention that Tom works at one such site:

<http://StackOverflow.com/jobs>.)

The Internet is full of communities specific to particular skills; many of these communities have job-listing areas. By seeking out people in the community, you can usually find better candidates. For example, the FreeBSD community has a mailing list just for community members looking for jobs and those who want to hire them.

Many headhunters specialize in SA recruiting and may have a useful database of SAs and the types of jobs that interest them. Your company’s own web site should advertise available positions. If you do not list any openings on your own



web site, people will assume that the organization isn't hiring. There may be political barriers preventing the posting of job listings on your web site, but it is worthwhile to overcome those obstacles.

One problem that often arises in hiring SAs is that the human resources department does not fully understand SA resumes or SA job descriptions the way SAs do. The best recruiting campaign in the world will fail if the resumes do not get past the HR department. Excellent resumes often get rejected before they are even seen, particularly if the company uses a computerized resume scanning and processing system. Such systems typically code for keywords and are fine if you want to "hire the skill," but do not work as well if you want to "hire the person."

The best way to make sure that good resumes aren't discarded out of hand is to have one HR person assigned to hiring SAs, and to have SAs and SA managers team up with that person to gain an understanding of how to identify good and bad resumes, what is relevant for one position or another, and so on. Eventually, the HR person should be able to go through the resumes and pick out all the good ones. The HR person will tend to focus on keywords to look for, such as certifications, particular technologies, products, and brand names. You will have to explain terminology. For example, you may find yourself explaining that Linux, RHEL, Solaris, and FreeBSD all mean Unix. If you are looking for someone with experience with a particular Linux distro, explain whether you want someone with experience with that particular distro, or if any experience will be fine as long as the person is willing to learn and adopt yours. Provide HR with a chart of terms and their equivalents. It is also useful to give copies of the LISA booklet *Job Descriptions for System Administrators* ([Darmohray 2012](#)) to HR personnel. This booklet provides industry standard nomenclature to use for the different kinds of SAs and skill levels.

Once the HR contact is trained, the SA managers can let the HR person do his or her job in peace and have confidence that the HR person is not missing out on good candidates. Reward the successes of the HR person whom you have trained. Make sure he or she is the first person in HR to receive computer upgrades. Make all the other HR staff want to be the next to be trained.

Some companies do not trust their HR department to evaluate resumes at all. Instead, engineers or managers volunteer to take rotations evaluating resumes into "call," "don't call," and "maybe" piles. This can be especially useful when HR is still getting up to speed or when hiring is only an occasional task and training HR personnel would have little return on investment. Be sure that you have a mechanism to inform HR when you are expecting a resume from someone who deserves special attention—for example, someone well known in

the field, someone you've particularly recruited, or someone whom you otherwise know to be a good candidate.

Sometimes, a recruiter or someone from HR is the first person to talk to the candidate and does a phone screen. It is important to make sure that this first contact goes well. A recruiter who knows what to look for in the resume should also know how to perform a phone screen.

---

### **Bad Phone-Screening Alienates a Candidate**

A large software company phone-screened a candidate for a software development vacancy. The candidate had a research degree in computer science and some software development experience. The HR person who performed the phone-screen asked the candidate to rate her programming skills against the other people she had worked with in the research group. The candidate tried to explain that everyone in the group was good at programming but that the ability to understand and solve the problems was the more important and relevant skill. Her interviewer didn't seem to understand what she was saying. At the end of the interview, the interviewer told her that it was interesting talking to her and that he had never spoken to a computer scientist, as opposed to a programmer, before. The candidate decided right then that there was no way she wanted to work for that company.

Various recruiters and managers called her after that, trying to persuade her that she was interested in the job after she told them that she was not interested. They also displayed some corporate arrogance: Even after she told them she had accepted another job, they couldn't understand why *she* was not interested because *they* were interested in continuing to interview her. This attitude confirmed her decision not to work for that company.

---

If your company has an in-house recruiter, an education process should be instituted. The best recruiting campaign in the world will fail if potential candidates think that your recruiter is an idiot.

## **53.4 Timing**

In hiring, timing is everything. There is a brief window when someone is available and a brief window when you have open positions. The fact that those windows overlap at all and enable you to hire people is practically a miracle. Sometimes, positions open up just after an excellent candidate has taken a

position elsewhere, or the perfect candidate becomes available right after the job has been offered to someone else. Sometimes the timing mismatch is unavoidable, but at other times a company's inability to move quickly costs it the chance of hiring the best candidates.

When trying to attract SAs into a company that is meant to be a high-tech, fun company, the company needs to be able to move quickly in making hiring decisions. If a company delays too long in making its decision or has a long, drawn-out interviewing process, the best candidates already will have accepted offers somewhere else. Some companies even write the offer letter before the candidate arrives, particularly if the candidate is being flown in from far away. The offer can be made on the spot, and the candidate can stay for an extra day or two to look at housing and schools, if necessary.

---

### **Strike While the Iron Is Hot**

A Silicon Valley start-up went bankrupt and had to let all its employees go on a Friday morning. One of those people was a well-known SA. Several companies rushed to employ him. During the following week, he had first- and second-round interviews with a few companies, and at the end of the week had several offers and made his decision. One of the companies that he decided against did not have the right people interview him and had not trained the employees well in the art of interviewing, because they unintentionally told him why he should not work there. Another company that was very interested in hiring him was unable to move as quickly as the others and failed to have him even interview with the necessary people before the end of the week.

---

A balance must be found between interviewing lots of candidates to find the best ones and moving quickly when a potentially suitable candidate is identified. Moving too slowly can result in losing the candidate to another company, but moving too quickly can result in hiring the wrong person.

---

### **Don't Rush a Hiring Decision**

A midsize software company needed to hire a network administrator to develop network tools. The team interviewed a candidate who was reasonably well qualified, but his experience was on different hardware. He already had an offer from another company. The team knew that it had to make a decision quickly but was not sure about his skills or his fit with the team. The team wanted to do another round of interviewing, but there

was not enough time. The team decided to offer him the job; he accepted the offer. Unfortunately, he did not work out well in that position or with the team. The team was stuck with the bad decision that was made under time pressure. Given more time and another round of interviews, the team might have decided against hiring him.

---

Hiring a new employee is a long-term commitment; it pays to take the time to get the right person. If the person is genuinely interested, he or she will ask the other suitors for more time to make a decision and should be given it, for fear of losing the person by forcing his or her hand. All parties should take the time they need and give the others that same respect. However, once you have made a decision to hire a person, act fast or your candidate might take a different offer.

### **53.5 Team Considerations**

It is important to make sure that the person you are hiring will fit into the team. One aspect of this is making sure that there aren't any serious personality clashes. A personality clash can lead to people not enjoying their work and using their energy in a negative way that lowers morale in the team. Low morale is bad for the team. The USENIX LISA booklet *Hiring System Administrators* ([Phillips & LeFebvre 1998](#)) contains an excellent discussion of the psychology of hiring and how every new person who joins the group changes the team dynamic.

Sometimes, the decision is difficult because there may not be any obvious personality clashes, but the candidate has a different style from the rest of the team. This is where careful consideration of the job description before trying to interview candidates should help. The candidate's style of working may be something that the team is lacking and, as a result, all the interviewers will be aware that it is a desirable quality.

---

#### **Know What You Are Looking For**

A midsize company was trying to hire a second person for the security team. The SA organization was a young, dynamic, fast-paced, fast-growing team. The security SA's manager gave her the responsibility for choosing the right partner for the security team. However, the manager did not provide direction or advice on what was lacking in the organization. The SA interviewed one candidate who was well qualified but decided against hiring him because she was unsure of the team fit. He was a steady, reliable worker who always carefully researched and considered all the options, whereas the rest of the SA team was made up

of people who made quick decisions that were almost always good but occasionally had negative repercussions. The SA was not sure how well he would get on in that environment. In retrospect, she realized that he was the ideal candidate, precisely because of his difference in style.

---

Another aspect of team diversity relates to cultural and ethnic diversity. Diversity in a team can be managed to garner excellent results. Everyone on a team brings a unique history and background, and these differences enrich a team. It's one of the ways that a group can make better decisions than one person alone. We all grow up with different experiences, based on our level of wealth, ethnic culture, stability, and myriad other issues. These experiences affect our thought processes; that is, we all think differently because of them. If we all had the same life experiences and thought the same way, we'd be redundant; instead, it is possible to manage diversity as your strong suit.

---

### **Three Different Backgrounds**

Three SAs who worked together had very different upbringings. One had strict, religious parents who gave her a highly structured upbringing. In turn, structure gave her comfort and security; everything had to be documented in detail before she could move forward. Another was kicked out of his house as a teenager when his parents found out that he was gay. He had to learn to survive on his own; he resisted structure and valued self-reliance. The third SA had been brought up in a very unstructured household that moved around the country a lot. He craved structure but expected unanticipated change. As a result, all three had relatively predictable work habits and design styles.

Because they saw these differences as having many tools in their toolbox, it made for powerful results. Any design had to have enough structure to satisfy the first person, each component had to be self-reliant enough to satisfy the second person, and the system had to be able to deal with change well enough to satisfy the third. When documentation needed to be written, they knew who would see to its completion. This situation could have been a disaster if it had led to disagreements, but instead it was managed to be their strength.

---

A manager must create an environment that supports and celebrates diversity rather than trying to fit everyone into the same "box." Model a willingness to

listen and understand. Challenge the group members to understand one another and find best-of-both-worlds solutions.

---

### **Four Worldviews**

Tom once interviewed the highest-performing engineering team in his division to find the secret of its success. One of the engineers pointed out that the four of them came from very different religious backgrounds, each of which represented a different worldview. He felt that this was the group's biggest strength. One's deity was inside us all, one's looked down on everyone with a mindful eye, another was at the center of everything, and another was out of reach to all but a very few. These worldviews were reflected in how they addressed engineering problems.

Although the engineers had different worldviews, they all agreed that none of those worldviews was the right analogy for every engineering problem. Instead, each could perform his own analysis of a problem using very different philosophies. When they came together to share their results, they had a very thorough analysis. The discussions that resulted helped each member gain a deeper understanding of the problem, because each person tended to find different issues and solutions. Once they all saw eye to eye, they could design a solution that took the best aspects from each of their analyses.

---

Another aspect of the team picture is no less important: mentoring and providing opportunities for growth for the junior members of the team. Junior SAs need to be trained, and they also need to be given the opportunity to grow and demonstrate their talent, especially in their first SA job. It can be very worthwhile to take a risk on hiring a junior SA who seems to have potential if an appropriate mentor is available and can be assigned that role before the new hire's first day. Never hire junior SAs into positions where they will not be properly supervised and trained on a day-to-day basis. People hired into positions like this will not enjoy their work and often will make mistakes that other people have to correct. Hiring juniors into an SA role and giving them superuser access and insufficient mentoring is a recipe for disaster, and they are likely to create more work for others rather than relieve the workload.

---

### **Why Junior SAs Need Mentors**

An ISP start-up split its SA organization into two parts to avoid a conflict

in priorities. One team looked after the Internet service, and the other looked after the corporate systems and networks. Senior SAs kept transferring from the corporate team to the service team; at one point, only a couple of junior SAs were left on the corporate team. The senior SAs on the service team answered questions and helped out when they could, but their priority was maintaining the service. One day, the root partition on the corporate mail server filled up. The junior SA who found the problem went looking for help from the senior SAs, but they were all heavily involved in some important work on the service and couldn't spare much time. She decided that the best thing to do was to find some large files that probably weren't used very much, move them to another disk partition, and create links to them from the original location. Unfortunately, she chose the shared libraries (the Unix equivalent to Windows DLL files), not knowing what they were or the effect that moving them would have.

When she moved the libraries, she discovered that she could not make links to them, because that command no longer worked. As she tried to figure out what was wrong, she discovered that none of the commands she tried worked. She decided to try rebooting the system to see whether that fixed the problem. Of course, the system was unable to boot without the shared libraries, which made things worse. She ended up taking the mail server and another system apart in the computer room to try to boot the machine and recover it to a working state. As luck would have it, when she was in the middle of this operation, with computer parts strewn all over the floor, a TV crew that was doing a feature on the company came through the computer room and decided that she made for interesting footage. This added even more stress and embarrassment to an already awful situation.

Eventually, when the corporate mail server had been down for several hours and senior management was expressing displeasure, one of the senior SAs on the service team was able to break away and help her fix the problem. He also took the time to explain what the shared libraries were and why they were so critical. If she had had a mentor who was able to spend time with her and help her through problems such as this, the situation would have been resolved quickly without a huge outage and embarrassing TV footage.

---



## 53.6 The Interview Team

To evaluate candidates properly, the interviewers need to work as a team to find out whether the person has the necessary technical and soft skills and is a good fit with the team. This is another situation in which the job description plays an important part. Which skills should the candidate have? Divide the desired skills up into small, related sets, and assign one set of desired skills to each interviewer. Select the interviewers based on their ability to interview for, and rate the candidates on, a particular set of skills. Some people are better at interviewing than others. If you are doing a lot of hiring, make sure that the better interviewers don't get totally swamped with first-round interviews. Consider saving them for the second round. Make sure that all the interviewers are aware of which skills they are interviewing for and which skills other people are interviewing for, so that there is minimal duplication and everything is covered. Only really critical skills should be assigned to more than one person. If two people are assigned to interview for the same skill, make sure that they ask different questions.

If there are many openings, have the interview team cover the skills required across all the jobs. It will probably take a while to fill all the positions. As more positions are filled, the search can be narrowed and the skills needed restricted to cover the gaps that remain in the SA team's skill set. If there are many positions, describe them all to the candidate and, if appropriate, express willingness to have some crossover between the positions. Find out which positions the candidate finds the most interesting and appropriate for his or her skills.

Both the candidates and the team members will want to know with whom they will be working. It is vital that the candidates spend time during the interview process with the person who will be their manager and the people with whom they will work most closely. Give the candidates the opportunity to meet as many people in the team as possible, without making the interview process overwhelming and exhausting. Consider adding an internal customer to the interview team, which benefits the candidates and your relationship with your customers. Bring a promising candidate to lunch along with two or three SAs. This gives the candidate an opportunity to get to know some of his or her potential co-workers in a more relaxed environment, which should help in the decision of whether to work with your team. One common mistake is to ignore the person at lunch. Make sure that you try to keep the candidate involved in the conversation. Ask open-ended questions to encourage the candidate to converse with the group.

---



### Involve the Candidate

Tom once worked at a site that always took candidates out to lunch with the entire team to see whether the candidate fit in well with the group. Tom was appalled to watch as no one spoke to the person. Tom tried to bring the candidate into the conversation, but the candidate would immediately be interrupted by someone, and the conversation would be diverted away from the candidate. As a result, Tom instituted “how to have lunch” training, whereby everyone was forced to practice such useful phrases as “Hey, that’s interesting. Tell us more about that.”

---

## 53.7 Interview Process

The most important point that all interviewers need to remember is to respect the candidate. The candidate should always leave feeling that the interview was worthwhile and that the interviewers were the sort of people with whom the candidate would like to work. The candidate should have the impression that interviewing him or her was the most important thing on the interviewers’ minds. Remember, you want the candidates to want to work with you. You want them to tell their friends that your company looks like a great place to work. Candidates need to leave the building with their self-respect intact and liking the company. Some ways to show respect include the following:

- **Read the candidate’s resume before you arrive at the interview.** Nothing insults a candidate like being asked about the flight when home is one town over, or being asked to list information that is clearly available on the resume.
- **Be prompt for your interview; don’t leave the candidate waiting.** The candidate has given up a lot of time to come to the interview; don’t make the candidate feel that it was a waste of time.
- **Before meeting the candidate, turn off all radios, pagers, and cellphones.** Your company will survive for an hour without you, even if it doesn’t feel that way. You don’t want the candidate to feel as though the site is in such a state of chaos that he or she will never have a moment’s peace. Nor do you want to waste precious interviewing time worrying about, or dealing with, problems at the site.
- **Show interest in the candidate.** This helps make a good impression, calms the candidate down, and makes the candidate feel wanted. Some good questions to ask include these ice-breakers: Did you have a difficult time finding the office? Was the traffic bad? How are you finding the

interviews so far? Would you like a drink or a quick break?

- **Make sure that everyone is asking different questions.** Doing so shows the candidate that the group communicates and works well as a team. It shows that thought and preparation went into the interview process. Answering the same questions over and over is a waste of time for the candidate. Asking different questions also gives the interviewers a broader view of the candidate when they compare notes later.
- **Don't set out to prove you know more.** The point of the interview is to find out how much the candidate knows, not to demonstrate your own brilliance or to try to catch the candidate out on some insignificant little detail that most SAs will never come across in their careers.
- **Ease the candidate into it.** Start with relatively easy questions to give the candidate a chance to build confidence, and gradually make the questions more difficult. This process will give you a good idea of the candidate's skill level and gives the candidate a chance to demonstrate his or her true skills, rather than succumbing to nerves and forgetting everything. It is in both parties' interests for the candidate to show his true skill level.
- **Don't prolong the agony.** When you find the candidate's limits in one area, back off and try to find his limits in another area. Give the candidate a chance to build up confidence before probing his limits.
- **Try to identify your concerns as you go.** If you have a concern about the candidate, try to address it during the interview. You may not get a second chance. For example, if it isn't clear whether the candidate understands a particular technology, ask more questions about it.

---

### Take Care of the Candidate

An Internet start-up interviewed a senior SA who had been personally recommended and recruited by the two most senior SAs on staff. The interview time was 6 PM to 10 PM. The employees of the start-up were working late nights all the time and always ordered take-out dinners from nearby restaurants. During one of the interviews, someone asked the interviewer what he wanted for dinner but did not ask the candidate, who also had not eaten. The next interviewer brought in his dinner and proceeded to eat it in front of the candidate, without thinking to ask whether she wanted anything. The same thing happened during the second round of interviews.

The candidate was offered the job, but the company members were

surprised when she turned it down. In the ensuing months, she talked to many people about the awful interview process and came across others who had undergone similar experiences and also turned the company down.

---

Successful interviewing requires preparation, practice, and coordination. Record your thoughts on the interview and the candidate as soon as you come out of the interview. Doing so gives you the freshest, most accurate picture of the candidate when the team gets together to decide whom to hire. It also gives you the opportunity to think about concerns that you may not have probed in the interview and to ask a later interviewer to direct some questioning toward those concerns.

One of the biggest mistakes that companies make is to “sell the company” only after they’ve asked enough questions to determine that the candidate is qualified for the job. They feel that anything else is wasting time with people who aren’t going to be hired. As a result, qualified candidates walk away from an interview thinking, “What a boring company!” or possibly “What a bunch of jerks!” Instead, start each interview by explaining what a great company you have and why you personally like the company. This starts the candidate off feeling eager to join; it relaxes the person and makes the interview go more smoothly. It also puts you in better light. If the candidate isn’t qualified for the position, he or she might have friends who are. Also, an unqualified person may be qualified later in his or her career, and it would be a shame if you’ve turned that person off to your company through one bad interview.

A related mistake is not giving the candidate enough time to interview you. A prospective employee will want to know more about the company, the position, the team, the work, and so on. It’s a good idea to allot one-quarter to one-third of the interview time for the candidate’s questions. Good candidates will ask good questions, and you will get an idea of what is important to them by the questions they ask. That will help you to make an attractive offer if the company decides to pursue a specific candidate.

## **53.8 Technical Interviewing**

Technical interviewing finds out whether the candidate has the skills and knowledge required for the job. (Nontechnical interviewing is discussed later.)

Some interviewers should be assigned to interview the candidates for their technical skills. This task can prove to be quite a challenge in a field as broad as

system administration. However, the job description should provide a basis for an approach to the technical interviewing. It should identify whether the person needs specific technical skills and experience or more general problem-solving skills, for example. Technical interviewing for an architect position should identify design skills rather than intimate knowledge of the latest hardware from a particular vendor.

The interviewers must pitch the interview questions at the right level. Architects should be presented with a set of requirements and be asked to talk through potential solutions. With senior SAs, pick an area of technical expertise that both the candidate and the interviewer have in common and dig deep. Check for problem-solving skills and the ability to explain what the SA would do and why. Intermediate-level SAs should have a reasonable breadth of experience, problem-solving skills, and the ability to explain their decisions. Look for junior SAs who pay attention to detail and are methodical. When interviewing SAs with no previous experience, look for someone who has an interest in computers and in how things in general work. Have those candidates written computer games or taken their home PCs apart?

When looking for problem-solving skills, it can be informative to see whether the candidate understands how things other than computers work. Ask the person to explain how a couple of everyday items work, such as an internal combustion engine or a flush toilet. Find out what else the candidate fixes. Does the candidate fix broken electrical appliances or blocked sinks or furniture? Does he or she know how to change the oil in a car or do any small jobs on it? Some people believe that the ability to read sheet music demonstrates logical, methodical thinking and good problem-solving skills. Come up with imaginative ways to figure out how the candidate thinks. Ask the person why manhole covers are round, and watch the person's deduction and logic as they figure it out. Ask the candidate to estimate the number of fax machines in Manhattan and see the logic they use to produce engineering estimates.

It is always a good idea to look at candidates' past experiences and get their thoughts on them. For example, asking which accomplishment they are most proud of and why can give you a good idea of what they are capable of and what they see as difficult challenges to overcome. Ask them about situations they have been in that they wish they had handled differently and what they would have done differently in hindsight and why. You want to know that they can learn from their mistakes. Ask the candidates to describe a big mess that they have had to clean up, how it came into existence, how they fixed it, and which constraints they had to work under to fix it. The candidates will enjoy telling their stories,

and this should provide insight into how they think and work. Ask them to tell you about a big project they have had to work on alone or in a team. If you are interested in a particular skill, ask the candidates about a problem that needed solving in that area. Ask them about a tricky problem you have encountered in that area, and see how they would have approached it. Most of all, get them talking about real-life experiences. It's easier and more interesting for them and gives you a better idea of what they are capable of than dry, "how does this work" questions.

Don't ask trivia questions—that is, questions with a single, highly specific, correct answer. People can forget that kind of thing when under stress. If their job requires them to know which pin on a V.35 interface is used to transmit data, they can look it up when they have the job. General questions help you understand more about candidates than trivia. Trivia questions are also called "gotcha" questions because there is a temptation to say "Gotcha!" when the candidate doesn't know the answer.

Also avoid brain-teaser questions. We cringe when we see companies ask candidates to connect nine dots laid out  $3 \times 3$  using four straight lines, without lifting their pencil from the paper. These questions have nothing to do with whether a person can do a particular job. In fact, we assert that they test only whether the candidate has heard the question before.

*The secret to good interviewing is the follow-up question.* The questions that you ask are minimally important compared to the follow-up question you ask in response to the candidate's answer. Here's an example:

Interviewer: Do you know C++?

Candidate: Yes.

I: Do you know how to set up MS Exchange?

C: Yes.

I: Do you know Apache?

C: Yes.

I: Do you know PHP?

C: Yes.

I: Do you know SonicWall firewalls?

C: Yes.

Wow! What a great candidate. She knows everything from programming in C++ to setting up web servers, email servers, and even firewalls! However, we

haven't asked any follow-up questions. Let's see how follow-up questions make the interviewing process gather more accurate information:

Interviewer: Do you know C++?

Candidate: Yes.

I: What was the longest C++ program you've written?

C: A 200-line program that reformatted CSV files into a template.

I: What made C++ the best language to use? Why didn't you use a language like Perl?

C: It was a homework assignment in a C++ class. I didn't have a choice.

I: What about C++ programs you've written outside of school?

C: I really haven't written much C++.

I: Why might you choose C++ over other languages?

C: I don't know. C++ is difficult; I'd rather use something easier.

I: Do you have experience with Apache?

C: Yes.

I: Have you maintained an Apache system or set one up from scratch?

C: From scratch. I downloaded the package and installed it on a Linux system. I then maintained the site for the company. The design was done by a consulting firm, but I had to fix problems with its HTML and change the configuration to work with its CGI scripts.

I: When I run Apache, I see a lot of subprocesses. What do they do?

C: I've never really looked into that.

I: Can you take a guess?

C: Maybe they each take care of a different request?

I: Why would that be a good thing?

C: To split the load.

I: Well, the machine has only one CPU. Why would multiple processes help?

C: I don't know.

I: Let's move to another topic. Do you have experience with SonicWall firewalls?

C: Yes.

I: In general, how does a firewall work?

C: (The candidate gives a very technical explanation of how firewalls work and mentions the state table.)

I: In which situations might the state table fill?

C: (The candidate discusses long-lived sessions, timeouts, and ways to tune the timeouts for sessions. She also explains the FIN packet process and other details.)

Because we used follow-up questions, we have a much clearer sense of this person's real skills. We see that she has rudimentary C++ skills that haven't been exercised since college. At least she understands some programming concepts. The candidate's knowledge of Apache web servers is somewhat superficial but sufficient to run a basic web server. She doesn't know the internal workings of Apache, which would be important for massive scaling projects or debugging oddball problems, but the experience with changing the configuration to permit CGI scripts—if not done through a menu—shows real understanding. Finally, the firewall questions show that she has a deep knowledge of the subject, not only understanding how they work but also demonstrating experience with a common scaling issue.

Of course, no one really conducts an interview as badly as the first example. We exaggerated the lack of follow-up questions but not too much, compared to some interview processes we've seen. Yet, the second example reveals another deficiency in the interviewer's technique: Obviously, the interviewer was not well prepared. Studying the candidate's resume beforehand would have indicated that she is a network or security engineer, and the questions should have been directed to those areas. Questions about C++ and Apache might have been fine questions for later in the interview, to see the breadth of her knowledge. However, asking the questions in that order must have made the candidate concerned that she was being interviewed for the wrong job.

Determine the limits of a candidate's knowledge by asking deeper and deeper questions until the person finally says, "I don't know." The downside of this technique is that candidates may feel that for every topic you touched on, they were unable to answer the last question asked; meanwhile, you were pleasantly surprised that their "limit" was just fine.

Seeking out the limit of the candidates' knowledge in the areas they feel they are strongest is an excellent way to understand their primary knowledge. Then ask questions in related topic areas to see how broad their knowledge is. To

identify the starting topic, ask them where they feel they are strongest in a list of choices, such as network protocols (how they work), network equipment, operating system internals, and scripting. The topics should be based on the experience listed in their resume. Ask them to pick their single strongest area.

Do not ask candidates to rate their skills on a scale of 1 to 10 for each topic. A study by Kruger and Dunning ([1999](#)) finds that unskilled people are unaware of their lack of skills and will overestimate them, whereas more skilled people are more aware of their limits. This is known as the Dunning–Kruger effect.

---

### **Asking Candidates to Rate Their Own Skills**

Asking candidates to rate their own skills can help indicate their level of self-confidence, but little else. You don't know their basis for comparison. Some people are brought up being taught to always downplay their skills. A hiring manager once nearly lost a candidate who said that she didn't know much about Macs. It turned out that she used one 8 hours a day and was supporting four applications for her department, but she didn't know how to program one. Ask people to describe their experience instead.

---

## **53.9 Nontechnical Interviewing**

For nontechnical interviewing, it can be particularly useful to take a course on interview techniques. There are several approaches to interviewing. One that works well is behavioral interviewing, which looks at past behavior to predict future actions. This technique can also be used for technical interviewing. Questions are phrased something like this: “Think of a time when.... Describe the situation to me, and tell me what you did.” Do it in such a way that it is clear that you won't fault the candidate for this situation having arisen and that you simply want to know how he or she handled it. For example, you might say to an SA candidate, “We have all experienced times when things have become really busy and something has fallen through the cracks or has been delayed. Think of a time when this happened to you. Describe the situation and tell me how you handled it. What did you learn from the experience?” Ask about both good and bad things. For example, ask about the best project that the candidate worked on, and why he or she felt it was the best, and then ask about the worst one. Ask about the person's best and worst managers to find out what he or she likes and dislikes in a manager and what the candidate's working style is.

This technique works better than an approach that asks the candidate how he or she *would* deal with a particular situation, because it is often easier for people



to see how they *should* behave when asked in such an academic fashion. When these kinds of situations arise in the real world, there may be other pressures and other factors that result in different behavior. Behavioral interviewing looks at how the candidate behaved in real-life situations.

The nontechnical interviews are used to evaluate the candidate's soft skills: how the person works in a team environment, how the person relates to customers, how the person organizes his or her work, whether the person needs lots of direction, whether the person likes a narrow job description or the opportunity to work in lots of areas, and so on. Typically, these questions have no right or wrong answers. The interviewing team needs to hear the answers to decide whether the person fits into the team, fits the particular position, or perhaps is more suited to another position.

For example, a company might interview a candidate who is very bright but is someone who completes 80 percent of the task and then loses interest because all the difficult problems have been solved. If this person can be teamed up with someone else who enjoys taking tasks to completion and will enjoy learning from the other person, this candidate would be a good hire. Conversely, if there is no one to pair with this person, this candidate probably is not a good choice.

Try to find out what the candidate's work habits are. If he or she stays in touch with technology by reading mailing lists and newsgroups and by surfing the web, does the person do so to the detriment of his or her other work, or does he or she find the right balance? The interviewer should get a good idea of the candidate's approach by chatting about how the candidate keeps up with technology and which mailing lists he or she reads. Some people waste phenomenal amounts of time keeping up with these things. Such candidates can be fun and interesting to talk to but frustrating to work with if they don't get much real work done.

Gauge the candidate's interest in computers. If computers are the focus of the candidate's life and are his or her only pastime, the candidate will burn out at some point. The candidate may work long hours because he or she enjoys the work so much, but that is not healthy or a recipe for long-term success. The candidate may also spend a large part of his or her time in the office playing with fun technology that has nothing to do with his or her work. Because the candidate works such long hours and spends all free time with computers, it will be difficult for the candidate to figure out how much real work is being done and how much is simply playing around. Conversely, some candidates may be surprisingly computer-averse. They may not want to touch or see a computer outside of working hours. These people generally lead a sustainable lifestyle and

will not burn out—they may have already had that experience—but they are less likely to be happy working lots of long hours. However, they are more likely to be productive all the hours that they do work. They are also less likely to be the first to hear about a new technology or to try it and suggest introducing it. Although the ideal candidate should have a balance between these two extremes, some jobs are suited to people at either end of the scale. Again, the interviewers need to know what they are looking for in the candidate.

### **53.10 Selling the Position**

Beyond finding out which candidates the team wants to work with, the interview is the time to persuade candidates that they want to work for the company. The first step on that path is to respect the candidates and make sure that the interview process does not waste their time or leave them feeling humiliated. The people whom the candidates meet during the interview process should represent the best of the team. They should be people with whom the candidates will want to work.

The position itself is also important. Find out what the candidate is looking for and figure out whether you can offer that in this position. Does the candidate want to work in a specific area of system administration or with many different aspects? Let the candidate know whether the position can be made into what he or she wants it to be. Each interviewer should think about all the positive things about the company and the group and share those with the candidates.

Internet commerce sites, ISPs, and SA consulting companies can make the most of the fact that the SA function is part of the core business of the company, so it is, or should be, well funded and viewed as a profit center rather than a cost center. It is almost always more enjoyable to work for companies that view SAs as valuable resources that must be invested in rather than as overhead that must be reduced.

### **53.11 Employee Retention**

Once you have hired a good employee, you want the person to stick around for the long term. What motivates SAs to remain at a company is different for each person, but some areas cover most people.

Demand for SAs follows the ups and downs of the high-tech world, as do SA salaries. Although salary is important—because, after all, no one wants to feel exploited—offering the highest salaries does not necessarily retain employees. Salaries should be competitive, but it is more important for the SAs to be happy

in their work.

One of the secrets to retaining SAs is to keep their jobs interesting. If there are tedious, repetitive tasks, get someone to automate those tasks. The person who is creating the automation will enjoy it, the SAs who no longer have to do the repetitive task will appreciate it, and efficiency will be improved.

As do most people, SAs like recognition for working hard and doing a good job. Performance bonuses are one way to recognize employees, but simply thanking them in group meetings for the work they are doing can work just as well.

---

### **Appreciation and Enjoyment Are Key**

A consulting company that had many excellent SAs was going through a turbulent period and losing a lot of staff. One SA who left commented that the senior management of the company had no idea what motivated people. Most of those who left were well paid, and all were offered a pay increase to stay. But as this departing employee put it: “People work for money; people work hard for appreciation; but people work the hardest when they love what they do. People are recognized as ‘the best’ when they love what they do, where they do it, and who they do it for.”

---

Good SAs enjoy working with other good SAs. They like to see important infrastructure projects funded and like to contribute to building good infrastructure. They like to see long-term planning rewarded. SAs like to feel that they are a part of the company rather than an unimportant, often ignored appendage. They like to have a good relationship with the people with whom they work closely, who generally are their customers. Although most SAs thrive on being busy, most do not enjoy being overloaded to the point that they are unable to meet customer expectations. SAs are often not the most communicative employees, but they like to be kept informed and expect their managers to notice when they are overloaded or when they pull off great feats. They also want to have opportunities to advance their careers; they will move on if they know that they will stagnate where they are.

SAs also like “fun stuff.” Fast connections to home, high-end laptops, and the opportunity to work with new technologies help to keep the appeal of the work environment high. Finding opportunities for SAs to work from home can also help retain them.

The final element in keeping most SAs happy is their direct manager. Some

people like a friendly relationship with their manager; others prefer clear lines of management and the knowledge that their manager has faith in them and will support them in what they do. People take jobs because of money, but people leave jobs because of bad managers. As discussed in [Section 52.7](#), it isn't what you do, but for whom you do it.

### **53.12 Getting Noticed**

Once a company has mastered the basics of hiring SAs, the final thing to consider is how to make the company stand out as one where SAs will want to work.

Employing some nontraditional incentives can help to get the company noticed and enhance its reputation as a fun place to work. For example, one company gave all interview candidates a Palm Pilot PDA—the current hot new product—at the interview. Another company had informal group outings to a nearby roller-coaster park, and the company bought season tickets for those who were interested. Another company had the latest video games and game controllers in rooms in a few buildings around the campus. Senior SAs and some managers in another company developed a tradition of playing multiplayer computer games one night a week. Others arranged sports or games leagues within the company or against other companies. Although most of these activities are not of universal interest, they are interesting to enough people that they distinguish those companies as being fun places to work, which can even help attract employees who are not interested in the activity that is offered.

Other, work-oriented schemes can help to distinguish the company in a positive way. Hosting the local monthly SA meetup/user group makes the company more attractive to SAs, especially since finding consistent and free meeting space is difficult to come by. Encouraging SAs to write and present papers and tutorials at conferences also demonstrates that the company is a good place for SAs to work and gets the company's name known in the SA field.

Ultimately, the aim is to get the company noticed and considered to be a fun place to work and a place that values its SAs. Be creative!

### **53.13 Summary**

The secret to hiring is good planning and follow-through. Hiring SAs starts with a good job description. The job description helps to define the skill level required and the soft skills needed, and it directs the interviewers' questioning toward relevant topics. It is also used for recruiting appropriate candidates. The

best way to recruit good candidates is through personal recommendations from customers and staff. Other ways that are oriented toward technologically minded people are good choices as well.

The skill level of the person hired has financial implications, and companies should be aware of the hidden costs of hiring under-qualified staff with lower salary requirements. The interview process should give the candidate the opportunity to meet the key people with whom he or she will be working, if hired. The interviewers should represent the company at its best. They should respect the candidate and make sure that the interview experience is pleasant. The technical and soft skills required for the job should be divided among the interviewers, and they should all know how to interview for their assigned skills.

The secret to good interview questions is the follow-up. Questions should give candidates a chance to shine by easing them into the questioning and letting them build confidence. This chapter looked at ways to assess problem-solving skills and design skills. Soft skills are just as important as technical skills. Interviewing courses can be very helpful for suggesting ways to accurately assess these skills in the candidates.

Once identified, the right candidate must be persuaded to take the job. This process starts with the first contact the candidate has with the company. The candidate must feel respected and be treated well throughout the interview process. Trying to recruit someone who has been offended in the interview process is hopeless. The compensation package is a part of wooing the candidate, but so are the positive aspects of working for the group, such as a cool staff or being well funded. Having a reputation as a fun place to work or a place that greatly values SAs can help enormously in hiring the right employees.

After people have been hired, they should be retained. Interviewing is a costly process.

## **Exercises**

1. What are the major components of the job interview process?
2. Does your company hire people above, below, or at the skill level required for a position? Illustrate your answer with some examples.
3. Write a detailed job description for an open position in your group or for a position for which you would like to be able to hire someone.
4. Which technical questions would you ask of a candidate for the position for which you wrote a job description?
5. Which nontechnical questions would you ask of a candidate for that

position?

- 6.** Which roles do you think are missing from your team?
- 7.** How diverse is your team? What could you do to better balance the gender, ethnic, and cultural diversity of your team?
- 8.** Who would be able to act as a mentor for a junior SA, if your group were to hire one?
- 9.** How have you made the interview process pleasant for a candidate in the past?
- 10.** How have you made the interview process unpleasant for a candidate in the past? Why did that happen? What can be done to prevent that from happening again?
- 11.** What motivates you to stay at your current company?
- 12.** What do you think your company does or could do to stand out as a good place to work?

## Chapter 54. Firing System Administrators

This chapter is about how to remove fellow SAs from your site because they've been terminated. Our wish is that you never need to use the information in this chapter. But the reality is that everyone will need to use this information someday. You may not be a manager saddled with the responsibility of deciding to terminate someone, but you may be the SA who must deal with the consequences.

This chapter is not about why someone should be fired. We can't help you there. Instead, it focuses on the technical tasks that must be performed when it happens. In fact, you might say that this chapter begins after the management decision has been made to fire someone and you must remove the person's access from the network.

Large companies usually have procedures for removing access when someone leaves. Small and midsize companies may have ad hoc procedures. However, often there is no procedure defined for the special case of a system administrator leaving the company. This chapter serves as a starting point for defining such a procedure, but we've structured it in a way to help you if you are reading this because someone just informed you that an SA on your team is about be terminated.

The process described is not theoretical. It is based on personal experience and the best current practices of many established companies. This chapter is based on the paper "Adverse Termination Procedures, or How to Fire a System Administrator" ([Ringel & Limoncelli 1999](#)).

Removing SAs' access against their will is a unique situation. SAs have privileged access to the systems and are likely to have access to all systems. Some SAs may know the system better than you do because they built it. The techniques in this chapter can also be used when dealing with an SA who is leaving a company on good terms. Whether people are leaving on good or bad terms, the potential for the situation to turn ugly makes them a liability to the company's computer infrastructure.

Our discussion of this topic will be restricted to the technical side of the process. The nontechnical side—the human side—is equally important. You are changing this person's life in a very profound way. The person has bills to pay, a family to support, and a life to live. Companies have different ways to discharge a terminated employee. No matter how you do it, the termination process should

show trust and respect and preserve the person's dignity. This attitude is not so much for the benefit of the person being laid off as for the benefit of those remaining.

## **54.1 Cooperate with Corporate HR**

The single most important rule is that you must follow the policies of your company with regard to firing people. These policies are written by people who understand the complex legal rules governing employee termination. Those people are the experts, not you. They have guidelines about what to say, what not to say, how to do it, and how not to do it. Usually, SA termination is such a rare and delicate event that someone from HR holds your hand throughout the process.

The HR staff should have a process that they follow. It might involve the employee's manager telling the person in a meeting room, calling him or her at home, or confronting the employee at his cubicle and informing him that his personal effects will be shipped to him as security guards escort him out of the building.

That said, HR may not be aware of the special issues regarding removing a system administrator's access. They aren't SAs and don't know how SAs think. This may lead them to be overzealous or lacking of zealousness in ways that could cause problems. Alternatively, you might be the one who needs a reality check. Therefore, you may need to pause and educate HR just as much as you may need to pause and be educated.

For example, we've seen situations where a manager requests that a departing SA retain some access for a few weeks after his official last day in case he is needed to fix a problem. HR forbids this for liability and legal reasons. The remaining SAs (the ones not leaving) are torn between being fearful that a technical problem might arise that they are unable to fix without the ex-employee's help and being concerned about the security risk of having a non-employee with privileged network access. At a tiny mom-and-pop operation it is acceptable to have a lackadaisical access policy that permits ex-employees to retain access, but we've also seen major companies with such arrangements. However, we've never seen such a policy at a company that is highly regulated or publicly traded.

We err on the side of caution and recommend access removal at or before the employee becomes no longer contractually part of the company. If support is needed, screen-sharing technology and cellphone camera pictures are amazingly



effective in communicating with an ex-employee who has volunteered assistance. Do remember, however, that this person is volunteering his time.

Either way, your work with HR should be collaborative. Most of the time, though, HR calls the shots.

## **54.2 The Exit Checklist**

Maintain a checklist of tasks to be performed when an employee is terminated. The first part should be constructed with help from HR. The second part should be the technical details of how to disable access. This list should be easy to update; anytime a new service is added, it should be easy to add the associated termination procedure to this list. This list should be analogous to the checklist used when creating accounts, but is likely to contain many more items for services that are not enabled at time of hire.

The benefit of using a checklist is that it eliminates the thinking required when someone is terminated. If you have to think about what is to be done, you may forget something. A checklist assembles all the knowledge in one place. Terminating someone is often an emotional moment, so you might not be thinking very clearly at the time. A checklist also provides a level of accountability. If you print the checklist and mark each item as complete, it creates a permanent record.

Most companies have a corporate-wide checklist that is defined by the HR and IT departments. It includes things such as surrendering company property, such as keys, card-keys, badges, phones, key-fobs, handheld authenticators, PDAs, tablets, laptops, desktops, micro-controllers (Arduino, Raspberry Pi), and any company-owned equipment that the employee may have at home.

SA teams should also maintain their own checklists. For example, the team that maintains the company's web presence may have its own databases, Amazon accounts, source code repositories, and so on. Those resources may not be tied into the global system maintained by the corporate IT team, so a separate checklist is needed for them.

If you centralize authentication and authorization for all applications into one place, then you just need to disable access in the central location (see [Sections 36.2.1](#) and [40.2.5](#)), which is easier and faster. Then, the checklist just needs to contain the central authentication and authorization databases, and any exceptions. Exceptions should also be separately tracked so that they will be remediated.

## 54.3 Removing Access

Removing access for an SA is different than for regular employees. Removing access for a regular employee is done quite frequently, so teams probably have checklists and procedures that are used regularly enough that they are kept up-to-date. SAs often have additional access that is not covered by such checklists. For example, there may be a special administrative access to the VPN server that is used to fix a problem if all network-based authentication systems are down. This may not be covered by any checklist.

An SA team should maintain a checklist of tasks to complete when an SA leaves that is maintained as new services are installed. Yet, when someone does leave, that list may not be up-to-date. Therefore the SA team should review it and consider what might be missing. There are six categories of access to be concerned with:

- **Physical access:** Building-access systems, proximity cards, physical keys
- **Remote access:** VPNs, remote access systems
- **Application access:** Applications, databases, and other services
- **Shared passwords:** Global root passwords, local administrator password, router passwords, any other passwords that SAs have access to
- **External service access:** Accounts on IaaS, PaaS, and SaaS systems such as Google for Work, Heroku, and Amazon AWS
- **Certificates:** SSL and other certificates

### 54.3.1 Physical Access

Removing physical access means, quite simply, making sure ex-employees can't get in the building. If they can enter the building, they can plug a laptop into the local network or enter the datacenter and access machines directly.

Usually, removing this access is the function of the HR department or corporate security. For example, HR should already have a procedure for retrieving the employee's ID badge, which is checked by guards at the building entrance. If a proximity card or other card-key access is used, the system must be programmed to deny access to that card regardless of whether it has been returned. Combination locks, safe combinations, and so on must be changed. Keys to any rooms must be retrieved or locks changed. Consider doors to datacenters, telecom rooms, closets, and automobiles. Consider the company headquarters, as well as primary, secondary, and remote locations. Is there a barn, maintenance shed, shack, outhouse, dog house, or annex that has network

connectivity? Check them, too.

Because physical instruments, such as doors and keys, have been around longer than computers, most companies usually have good procedures related to them. Very small companies might not have any kind of identification badge or similar system in place. In that case, have the receptionist and other employees been told what to do if they see this person in or near the building?

You must also schedule time for the SA to return any equipment that he may have at home. Any computers that are returned should be considered suspect. The disks should be erased to prevent virus transmission.

### **54.3.2 Remote Access**

Remote access refers to the many ways someone might get into the networks from outside the building. It includes the normal VPN access that all users use, but also special emergency or out-of-band access. For example, remote sites often have dial-in modem access to network equipment as an access method of last resort. SAs often set up in-bound SSH or telnet servers that only they have access to for use when the VPN server is down. There may be legacy network systems such as X.25, ISDN, and ADSL.

Network access control (NAC), described in [Chapter 23](#), “[Network Architecture](#),” keeps unauthorized machines off wired and wireless networks. While this may not be considered remote, it is likely controlled by the same team as VPN and other access.

### **54.3.3 Application Access**

Application access is all the various databases, commercial software, and home-grown applications that are deployed around the company. Such applications should tie into the company’s single sign-on (SSO) service so that disabling the SSO account disables access to all other applications. However, there are always applications that deviate from this norm. Accounts on each of those applications will need to be handled individually. The need to fix each of the deviant applications should also be tracked with a problem ticket, so that it is not forgotten, or simply deemed acceptable.

Things to check include single sign-on systems, centralized authentication systems, NIS domains, ActiveDirectory domains, NT domains, superuser access IDs, password files, TACACS/TACACS+/RADIUS servers, configuration management systems, database servers, corporate applications, and team applications.

---

### **The Protective Triangle**

Physical access, remote access, and application access form a **protective triangle** that self-corrects for any mistakes in one area. Any one leg of the triangle can be damaged while the other two will remain safe. For example, if an account is not disabled on an application, but physical and remote access are secured, the application is safe. If a remote access method is left open, but physical and application access are properly secured, the risk of someone getting into the network is relatively moot.

---

### **54.3.4 Shared Passwords**

SAs have access to accounts that are shared by many people. If the password is changed, everyone with access to that account must be informed. SA teams usually keep such information in a cryptographically secure password safe, such as KeePass or Conjur, but the password may be embedded in other places as well, such as configuration files.

In some companies, the Windows local administrator password is the same on all machines, as is the local root password. Changing these globally may be a simple matter due to the use of ActiveDirectory or a configuration-management system, or it may mean touching every machine in the network. There is often an emergency-access administrator password on network equipment (routers and switches) that must be changed on each device. If all devices are centrally managed by a network-management tool, then making that change should be relatively easy. Otherwise, it will be quite time-consuming.

Most likely, shared passwords will be the most labor-intensive items to change. Invest in automation that enables you to rapidly and accurately perform these changes on all affected systems.

### **54.3.5 External Services**

External service access is any third-party service used by the company, such as web-based services (SaaS systems such as Google for Work, Salesforce, and recruiting, hiring, and benefits systems), platforms (PaaS systems such as Heroku and Digital Ocean), and hosted infrastructure (IaaS systems such as Google Compute Engine, Amazon AWS, and Microsoft Azure).

Sometimes the service has one shared account that all SAs use. In that case, the password should be changed similarly to shared passwords. In other cases, the service has individual accounts for each SA, in which case the specific account can be disabled. In this situation, there may be a master account used to

create the individual accounts. If that master account has a password to which the ex-SA had access, that password also must be changed. Be sure to check CDN and ISP support accounts, hardware vendor portals, and DNS domain registrars.

External services are particularly high-priority items since they can be accessed from anywhere on the Internet—in fact, that is the point of such services. They are not protected by the physical–remote–application triangle.

### **54.3.6 Certificates and Other Secrets**

Certificates, and any digital secrets, are bits that are used in cryptography to authenticate people or machines, authorize access, or encrypt data. They include X.509 certificates (used for SSL and more), SSH keys for regular and role accounts, and GnuPG/PGP keys.

If the SA had access to the private key in any SSL certificate, he or she may have kept a copy. That could be intentional, or it could be just a side effect of not deleting temporary files. Either way, these certificates should be regenerated. Having the private key of a certificate would enable the person to decrypt private communication, execute a man-in-the-middle attack, or set up a web site that claims to be another. Regenerating such certificates at most companies tends to be either highly automated or painfully manual and error-prone. In the latter case, we’ve often seen companies simply decide that the risk associated with not changing the key is acceptable, especially if the SA’s departure is amicable.

Often these secrets are protected by a passphrase. Changing this passphrase is insufficient. For example, if an X.509 passphrase is changed, the private key behind it is not changed. An SA that kept a copy of the private key will not be prevented from using the private copy. Similarly, changing the password on a password safe does not prevent the ex-employee from using any passwords that he may remember or may have stored elsewhere.

## **54.4 Logistics**

When an SA needs to be removed from the network, it is important to track which access is being disconnected. Otherwise, someone might assume that someone else disabled an account; two people might change the same password, causing confusion; or other problems might happen.

Start by making a checklist of all the tasks that need to be completed. Use a Google Spreadsheet, wiki page, or other shared document so that everyone involved can see the list and mark the tasks that are completed. There should be

a column for the task, the name of the person who has volunteered to remediate that issue, a status column where people mark either “done” or the time and date of completion, and a notes column.

Augment the list by brainstorming each of the six categories. Someone may remember a new router that was installed recently and was accidentally left off the standard checklist. A new service may have come online recently. The person being removed may be involved in a project that hasn’t launched yet but needs special attention. There may also be other logistical issues to be added to the checklist—for example, taking an archival backup of the SA’s laptop hard drive, erasing certain media, shipping personal items to the SA’s home, and so on.

A large team might divide the work based on the six categories. For example, two people who specialize in network issues might take all the network and remote access tasks. Line managers, or their administrative support, might take the tasks related to physical access. Managers can take care of the HR processes.

## **54.5 Examples**

The following anecdotes are true. The first looks at a case where an SA decided to leave, and the separation was amicable. The second is about an SA manager who had privileged access and was suspended while under investigation. The third and final case focuses on the termination of an SA who had been working in the same place for a long time. The names have been changed to protect people’s identities.

### **54.5.1 Amicably Leaving a Company**

An SA at a small software development company announced he was leaving and offered one month’s notice to help the company transition. The SA was actually involved in interviewing his replacement. All responsibilities had been transitioned by the termination date. On his last day, the SA walked into his boss’s office and became `root` in a window on his workstation. While the boss watched, he disabled his own regular login. He then issued the commands to change the password of various routers and let his boss enter the new password. He then repeated this for a few other system accounts. Finally, he issued the command to activate the “change `root` password on all systems” procedure and let his boss enter the new password. The boss was shocked that the soon-to-be ex-employee was doing such a complete job of transitioning duties and deleting himself from the system. Finally, the SA put his card-key and physical keys on the boss’s desk and left the building.

About two weeks later, the ex-employee remembered that he had forgotten to change the password on a particular non-privileged system account on a machine that had a modem directly attached to it. The ex-employee reports that he confirmed with former co-workers that the ex-employer didn't disable the account until he notified them of the error.

With one exception, the termination was complete. The exception crossed both the remote and service access categories because the account (service access) was on a host that was directly connected to the outside world (remote access). Two sides of the protective triangle were weak. Also, it was unsafe for the new passwords to be set in front of the exiting employee, who could have been watching the keyboard. The employee also could have recorded the new passwords as they were being set using a keyboard-capturing tool.

The company took a risk by not disabling access as soon as notice was given. However, it was a reasonable risk to take, considering that it was not an adverse termination.

### **54.5.2 Firing the Boss**

A large manufacturing company had to suspend, pending investigation, the manager of a team of SAs. This person had administrative access to all the systems in his domain and access to the network via every remote access method that the system administration group provided.

Without the accused manager knowing, a meeting was called by the accused manager's director with the lead SA and corporate security. The corporate security officer explained the situation and the action plan. The lead SA was to change all the privileged-account passwords that evening; in the morning, the director would meet with the system administration group, without their manager, to inform them of the situation. They would be told to suspend all access to the systems. If something could not be suspended, access should be deleted. If the investigation proved the accusation was true, the suspended accounts would be deleted. Otherwise, they would be reactivated. Either way, the list of which accounts had been suspended would become a vital checklist.

The lead SA spent the evening changing the `root` password on every system. In the morning, all the SAs were brought into a closed meeting and the situation explained to them. After the initial shock, the team set to work. Corporate security would take care of the physical access tasks. The system administration team was assigned the other categories. They brainstormed together to build the checklist but split into subteams to do the actual work. All changes were logged,

and logs were turned over to the lead SA.

There was one problem where, to make a long story short, if the manager had a photographic memory, he could have leveraged a form of remote access to gain access to the network. It was going to take a couple days to rectify this problem, and nothing could make that happen any sooner. Even so, the risk that the manager would leverage this issue was low, and the system administration team was confident that the other sides of the protective triangle were sufficiently strong. They also monitored certain security logs so they could detect intrusion attempts. In other words, they were confident that doing a complete job on two of the categories would compensate for being incomplete elsewhere.

During the investigation, the accused manager resigned. The logs of which types of access had been removed were useful as a checklist of which access had been suspended and now needed to be deleted.

The process worked very effectively. There was increased efficiency from splitting into subteams. The potential problem with remote access was compensated by effectively removing all other access.

### **54.5.3 Removal at an Academic Institution**

Academic entities tend to be very astute about termination procedures, as they often have batches of terminated accounts at the end of every academic year. The university setting is a good example of how practice makes perfect. However, this anecdote involves someone with privileged access to many machines.

At a large state university, a long-time operator with `root` access to all Unix systems was to be terminated. She had been employed for so long that there was a concern that eliminating all access might be impossible. Such a situation had never been encountered previously.

Because all access could not be disabled instantly, there were concerns that the operator would be able to get back into the system. Retribution was not necessarily expected, but the university couldn't take that risk. Because this was a public university, physical access to the building could not be eliminated, but card-key changes prevented the operator from gaining direct physical access to sensitive machines. Remote access could not be disabled, because the large university did not use a firewall.

After some discussion, it was decided to use the following procedure: A small team was assembled to list all the types of access she had and how to disable each, including card-key and host access. When the designated time arrived, the employee was told that her boss needed to speak with her in his office. The two



offices were in opposite parts of the building complex and the trip would take at least 10 minutes. During that time, all the senior SAs would work to disable her access. By the time she reached her boss's office, all her accounts would be disabled.

Because the operator had such pervasive access, the only way to ensure complete coverage was to take such extreme caution.

## 54.6 Supporting Infrastructure

As you can see, certain infrastructure elements make access removal easier. The fundamental infrastructure element is to have some kind of central authentication/authorization database that is used by as many services as possible. For example, ActiveDirectory can be used to authenticate most services. Network services usually authenticate against TACACS+ or RADIUS, which can simply proxy ActiveDirectory or another authentication system. Single sign-on systems make application access more convenient for users and easier to control.

Minimize the number of authentication databases. Adopt a rule that new services must use the centralized authentication system. Keep an inventory of devices that have special administrative access passwords—this can be part of the SA exit list. Develop ways to rapidly change administrative passwords.

From a team perspective, the most fundamental infrastructure element is a cryptographically secure password safe or manager such as KeePass or Conjur. Once you have this component, you will find many ways to use it. In fact, you won't believe you ever operated without one.

From the logistical side, the most important tool is the exit checklist. Keep an inventory of access that needs to be withdrawn when users leave. It is not good enough to have an onboarding checklist and assume that it can be used in reverse when someone exits. That is a good cross-check, but SAs often receive new access to systems long after the onboarding process takes place. Plus, the exit checklist includes issues such as certificate regeneration and deployment, returning physical devices, and so on.

As the labor involved in the exit checklist grows, it becomes increasingly important to automate items. Automating these tasks also facilitates periodic password rotations. Tasks to consider automating include, but are not limited to, changing global `root` and Windows Administrator passwords, changing local administrator passwords (whether they are set the same on each machine, or to a different password on each machine), changing network device passwords, and

regenerating, renewing, and deploying X.509 (SSL) certificates.

Be ever vigilant in identifying ways to minimize the number of systems that need to be updated when an SA leaves. This includes minimizing the number of remote access methods (or authorization methods) and reducing the number of deviant applications (ones that do not interface with the central authentication system).

Nothing is perfect—so there is also a need for security monitoring to catch anything that fell through the cracks. Intrusion detection systems (IDS) detect unauthorized network access. There are products that detect unauthorized file system changes, such as Tripwire. Network access control (NAC) keeps unauthorized machines off wired and wireless networks.

## 54.7 Summary

Firing SAs isn't fun or easy, but sometimes it has to happen. The basic process is very simple. The most important rule is to follow the policies of your HR department. HR personnel are the experts, and you are supporting their process. There are many categories of access that must be removed. Use the list in [Section 54.3](#) to help jog your memory.

The main categories of access are physical access (can someone get into the building), remote access (can someone get into the network), and service access (can someone access an application). These three are the most common types of access and complement each other; if one is incomplete, the others will block access. Additional categories include access to external services (accounts on Internet-based services the company uses), shared passwords (global passwords used by multiple SAs), and certificates (private keys used in cryptography).

When removing an SA, follow HR procedures and use a structured process to remove access based on the six categories. The six categories provide a structured way to brainstorm missing items, and can be useful as a way to create subteams.

Architecture elements such as a central authentication database used by all services and applications facilitate the process of removing access. A well-maintained inventory of exceptions helps during the removal process.

Maintain a removal checklist. Whenever it is activated, take a moment to update it, as new services may have appeared since it was created and not been added to the list. If you are creating a checklist from scratch, start by consulting the onboarding checklist for items that need to be reversed, and brainstorm using the six access categories for inspiration.

## **Exercises**

1. What are the six categories of SA access?
2. When someone is fired in your company, does HR know whom to contact in the IT organization to have the person's access disabled?
3. In your current environment, what would have to be disabled if you were to be fired? Outside of checking individual hosts for local accounts, how many individual administrative systems did you have to touch? (Do not reveal security-sensitive or proprietary information.)
4. Which administrative-access databases are part of your environment? (Do not reveal security-sensitive or proprietary information.)
5. Which improvements to your system could make it easier to disable an SA's access if one was fired?

# **Part X: Being More Awesome**

## Chapter 55. Operational Excellence

This chapter is about measuring or assessing the quality of service operations. It proposes an assessment tool and gives examples of how to use it to evaluate an individual service, a team, or multiple teams.

The assessment system described in this chapter evaluates the degree of formality and optimization of processes—that is, whether processes are ad hoc, formal, or actively optimized. It gauges teams on a generic level, one that is more comparable across teams or across services within a team.

Such assessments help identify areas of improvement. We can then make changes, reassess, and measure the improvement. If we do this periodically, we create an environment of continuous improvement.

The system we describe in this chapter may not be directly suited to your organization, but it is a good starting point for creating your own. The goal is to improve the services being provided, not dogmatically follow the suggestions in this book.

### 55.1 What Does Operational Excellence Look Like?

What is great system administration? Like art and literature, it is difficult to define other than to say you know it when you see it. This ambiguity makes it difficult to quantitatively measure how well or poorly a system administration team is performing.

High-performing organizations have smooth operations, well-designed policies and practices, and discipline in what they do. They meet or exceed the needs of their customers and delight them with innovations that meet future needs, often before such needs ever surface. The organization is transparent about how it plans, operates, provides services, and handles costs or charge-backs to customers. The vast majority of customers are happy customers. Even dissatisfied customers feel they have a voice, are listened to, and have a channel to escalate their issues. Everyone feels the operations organization moves the company forward. Its funding reflects a reasonable budget for the work the organization does. The organization makes its successes visible and, more importantly, is honest and forthright when it comes to discussing its own faults. The organization is constantly improving. Outages and escalated issues result in action plans that reduce future occurrences of that problem. The world is constantly changing, and the organization incorporates new technologies and

techniques to improve its inner workings as well as the services it provides.

Operations organizations seem to fall into three broad categories or strata: the great ones, the ones that want to be great, and the ones that don't even know what great is.

We estimate that 5 to 10 percent of all operations teams fall into the first category. They know and use the best practices of our industry. Some even invent new ones. The next 25 to 30 percent know that the best practices exist but are struggling to adopt them. The remaining super-majority do not even know these best practices exist.

Science fiction writer William Gibson famously said, "The future is already here—it's just not very evenly distributed." Likewise, the knowledge of how to be a great system administration team is here—it's just not very evenly distributed.

## 55.2 How to Measure Greatness

Measuring the quality of an operations team is extremely difficult. Other aspects of operations are easy to measure. For example, size can be measured by counting the number of team members, the number of services provided, or the dollars spent per year. Scale can be measured by counting the number of machines, the amount of storage, the total bandwidth used, and so on. We can measure efficiency using cost ratios.

Alas, quality is not so easy to measure.

Imagine for a moment that we *could* measure quality. Suppose we had a standard way to rate the quality of an operations team with a simple value on a scale of 0 to 1,000. Also imagine that we could, possibly by some feat of magic, rank every operations organization in the world.

If we could do that, we could line all of these organizations up from "best" to "worst." The potential for learning would be incredible. We could observe what the top 50 percent do differently from the bottom 50 percent. Alternatively, one organization could study better-ranked organizations for inspiration on how to improve.

Alas, there is no such single measurement. Operations is just too complex. Therefore the measurement, or **assessment**, must reflect that complexity.

Assessments sound a lot like the grades we received in school, but the concept is very different. A student assessment evaluates an individual student's learning and performance. Grades assess learning, but they also incorporate attendance, participation, and effort. An assessment is more focused.

An assessment of a service is an evaluation based on specific criteria related to process maturity. It is not an evaluation of whether the service is popular, has high availability, or is fast. Not all services need to be popular, highly available, or fast. In contrast, all services need good processes to achieve whatever goals they do have. Therefore we assess process because good processes are a roadmap to success.

## 55.3 Assessment Methodology

The assessment methodology is a bottom-up assessment. A service is evaluated on eight attributes, called **operational responsibilities (OR)**. Each OR is assessed to be at one of five levels, with 5 being the best. If assessment is done periodically, one can see progress over time. A weighted average can be used to roll up the eight individual assessments to arrive at a single number representing the service.

A team performs this assessment on each service. A team can be assessed using the weighted average of the services it provides. When teams are aware of this assessment, they will naturally seek to improve their problem areas.

If there are multiple teams, they should be provided with their score and percentile. We recommend against stack ranking teams because this is a self-assessment. Direct comparisons encourage people to overstate their progress.

### 55.3.1 Operational Responsibilities

Some operational responsibilities may be more or less important for a particular service. Your team may emphasize or de-emphasize certain ORs by using a weighted average when performing roll-ups. Teams may also choose to add more ORs if need be.

We have identified eight broad categories of operational responsibilities that most services have. They are summarized here:

- **Regular tasks (RT):** Handling normal, nonemergency, operational duties—that is, how work is received, queued, distributed, processed, and verified, plus how periodic tasks are scheduled and performed.
- **Emergency response (ER):** Handling outages and disasters. This includes technical and nontechnical processes performed during and after outages (response and remediation).
- **Monitoring and metrics (MM):** Collecting and using data to make decisions. Monitoring collects data about a system. A metric uses that data to measure a quantifiable component of performance.

- **Capacity planning (CP):** Determining future resource needs. Capacity planning involves the technical work of understanding how many resources are needed per unit of growth, plus nontechnical aspects such as budgeting, forecasting, and supply chain management.
- **Change management (CM):** Managing how services are purposefully changed over time. This includes the service delivery platform and how it is used to create, deliver, and push into production new application and infrastructure software. It also includes firmware upgrades, network changes, and OS configuration management.
- **New product introduction and removal (NPI/NPR):** Determining how new products and services are introduced into the environment and how they are deprecated and removed. This is a coordination function. Introducing and removing a service or product from an environment touches on multiple teams. Removing it involves tracking down the current users and managing the migration away from the service so it can be eliminated. For example, it may involve coordinating all teams that are touched by the introduction of a new brand of hardware into an ecosystem, the launch of an entirely new service, or the decommissioning of all instances of an old hardware platform.
- **Service deploy and decommission (SDD):** Determining how instances of an existing service are created and how they are turned off (decommissioned). After a service is introduced to an environment, it is deployed many times. After serving their purpose, these deployments are decommissioned. Examples include turning up a new datacenter, adding a replica of a service, and adding a replica of a database.
- **Performance and efficiency (PE):** Measuring how well a service performs and how cost-effectively resources are used. A running service needs to have good performance without wasting resources. Examples include managing utilization and power efficiency and their related costs.

The differences that distinguish NPI, SDD, and CM are subtle. NPI is how something is launched for the first time. It is the nontechnical coordinating function for all related processes, many of which are technical. SDD is the technical process of deploying new instances of an existing item, whether it is a machine, a server, or a service. It may include nontechnical processes such as budget approval, but these processes are undertaken in support of the technical goal. CM is how upgrades and changes are managed, either using a software deployment platform or an enterprise-style change management review board.



[Chapter 56, “Operational Assessments,”](#) describes these and other ORs in detail.

### 55.3.2 Assessment Levels

The rating, or assessment, uses a scale of 1 to 5, based on the **Capability Maturity Model (CMM)**. The CMM is a tool designed to measure the maturity of a capability or responsibility.

The term “maturity” indicates how formally or informally an operational aspect is practiced. The formality ranges from ad hoc and improvised, to having a documented process, to measuring the results of the process, to actively improving the system based on those measurements.

The CMM defines five levels:

- **Level 1, Initial:** Sometimes called Chaotic. This is the starting point for a new or undocumented process. Processes are ad hoc and rely on individual heroics.
- **Level 2, Repeatable:** The process is at least documented sufficiently that it can be repeated with the same results.
- **Level 3, Defined:** Roles and responsibilities of the process are defined and confirmed.
- **Level 4, Managed:** The process is quantitatively managed in accordance with agreed-upon metrics.
- **Level 5, Optimizing:** Process management includes deliberate process optimization/improvement.

#### Level 1: Initial

At this level, the process is ad hoc. Results are inconsistent. Different people do tasks in different ways, usually with slightly different results. Processes are not documented. Work is untracked and requests are often lost. The team is unable to accurately estimate how long a task will take. Customers and partners may be happy with the service they receive, but there is no evidence-based determination at this time.

#### Level 2: Repeatable

At this level, the process has gone from being ad hoc to repeatable. The steps are defined in such a way that two different people can follow them and get the same results. The process is documented with no missing steps. The end results are relatively consistent. This is not to say that errors don't happen; after all, nothing

is perfect. Because there is little measurement at this level, we may not know how often work is defective.

### **Level 3: Defined**

At this level, the roles and responsibilities of the process are defined and confirmed. At the previous level, we learned what needed to be done. At this level, we know who is responsible for doing it, and we have definitions of how to measure correctness. The correctness measurements might not be collected, but at least the people involved know what they are. Each step has a series of checks to find errors as they happen, rather than waiting until the end of the process to find out what went wrong. Duplication of effort is minimized. If we need to increase the rate of production, we can duplicate the process or add more capacity.

### **Level 4: Managed**

At this level, aspects of the process are measured. How long each step takes is measured, including how much of that time involved waiting to begin the process itself. Measurements include how often the process is done each month, how often it is performed without error, and how many times an exceptional case arises that requires special treatment and ad hoc processes. Variation is measured. These measurements are collected automatically. A dashboard shows all of these measurements. It is easy to spot bottlenecks. Postmortems are published within a specific amount of time after an exception. Exceptions are cataloged and periodically reviewed. Requests to change a process are justified using measurement data to show that there is a problem. Capacity needs are predicted ahead of time.

### **Level 5: Optimizing**

At this level, the measurements and metrics are used to optimize the entire system. Improvements have been made at all previous levels, but the improvements made here are prompted by measurements and, after the change is made, the success is evaluated based on new measurements. Analyzing the duration of each step exposes bottlenecks and delays. Measurements indicate month-over-month improvements. We can stress-test the system to see what breaks, fix it, and then run the system at the new level.

---

#### **Defined Versus Managed**

People often have a difficult time conceptualizing the subtle difference

between Levels 3 and 4. Consider the example of assessing the load balancer aspect of a service. Load balancers can be used to improve capacity or resiliency, or both. Resiliency requires the system to run with enough spare capacity to withstand a failed backend.

If there is a written policy that the load balancer is used for resiliency, this is a Level 3 behavior. If there exists monitoring that determines the current level of redundancy ( $N + 0$ ,  $N + 1$ , and so on), this is Level 4 behavior. In this case, Level 4 is significantly more difficult to achieve because it requires a lot of effort to accurately determine the maximum capacity of a backend, which is required to know how much spare capacity is available.

Somewhat surprisingly, in many situations there is no clear sense of how a load balancer is being used, or there is disagreement among team members about whether the role of a load balancer is to improve capacity or to improve resiliency. This demonstrates Level 1 behavior.

---

### 55.3.3 Assessment Questions and Look-For's

To perform an assessment for an operational responsibility, describe the current practices in that area. Based on this description, evaluate which level describes the current practices.

[Chapter 56](#), “[Operational Assessments](#),” contains a standard set of questions to ask to help form your description.

That chapter also provides a set of **look-for's** for each level. A look-for is a behavior, indicator, or outcome common to a service or organization at a particular level. In other words, it is what a level looks like. If you read a description and it sounds like you're talking about where you work, then that's a good indication that your organization is at that level for that service.

For example, in the operational responsibility called Regular Tasks, at Level 1 there is no playbook of common operational duties, nor is there a list of what those duties are. At Level 3, those duties have been defined and documented.

Look-for's are not checklists. One does not have to demonstrate every look-for to be assessed at that level. Some look-for's are appropriate only for certain situations or services. Look-for's are simply signals and indicators, not steps to follow or achievements to seek out.

[Chapter 56](#) lists operational responsibilities, questions, and look-for's for each level. Take a moment to flip there and review some of them. We'll wait.

## 55.4 Service Assessments

To put this evaluation process into action, staff must periodically perform an assessment of each major service or group of related services they provide. The assessment is used to expose areas for improvement. The team brainstorms ways to fix these problems and chooses a certain number of projects to fix the highest-priority issues. These become projects for the new quarter.

The team then repeats the process. Services are assessed. The assessment inspires new projects. The projects are worked on. The services are assessed. The process then begins again. Teams that work this way benefit from having a structure in place that determines projects. They always know what they should be working on.

### 55.4.1 Identifying What to Assess

First, identify the major services that your organization provides. A web-based service might identify the major components of the service (major feature groups served by software components) and infrastructure services such as networks, power, cooling, and Internet access. An enterprise IT organization might identify the major applications provided to the company (e.g., email, file storage, centralized compute farm, desktop/laptop fleet management), plus infrastructure services such as DNS, DHCP, ActiveDirectory/LDAP, NTP, and so on. Smaller sites may group services together (DNS, DHCP, and ActiveDirectory/LDAP might be “name services”), and larger sites may consider each individual component to be its own service. Either way, construct your list of major services.

For each service, assess the service’s eight operational responsibilities. Each section in [Chapter 56, “Operational Assessments,”](#) lists questions that will help in this assessment. These questions are generic and should apply to most services. You may wish to add questions that are appropriate for your organization or for a particular service. It is important to use the same questions for each assessment so that the numbers are comparable. Make a reference document that lists which questions are used.

### 55.4.2 Assessing Each Service

During each assessment period, record the assessment number (1 through 5) along with notes that justify the assessment. Generally these notes are in the form of answers to the questions.

[Figure 55.1](#) shows an example spreadsheet that could be used to track the

assessment of a service. The first column lists the eight operational responsibilities. The other columns each represent an assessment period. Use a different subsheet for each service. Use the spreadsheet’s “insert comment” feature to record notes that justify the assessment value.

| Service A                              | 12/2014 | 1/2015 | 2/2015 | 3/2015 | 4/2015 |
|----------------------------------------|---------|--------|--------|--------|--------|
| Regular Response                       | 1       | 2      | 2      | 2      | 3      |
| Emergency Response                     | 3       | 3      | 4      | 4      | 4      |
| Monitoring and Metrics                 | 1       | 1      | 2      | 2      | 2      |
| Capacity Planning                      | 1       | 1      | 1      | 1      | 2      |
| Life-Cycle Management                  | 2       | 2      | 2      | 3      | 2      |
| New Service Introduction and Removal   | 1       | 2      | 2      | 3      | 3      |
| Service Deployment and Decommissioning | 2       | 2      | 3      | 3      | 4      |
| Resource Efficiency                    | 1       | 1      | 1      | 2      | 2      |

Figure 55.1: Assessment of a service

It is a good idea to list the responsibilities in an order that indicates their importance to that service. For example, a service that does not grow frequently is less concerned with capacity planning, so that responsibility might be listed last.

Color the squares red, orange, yellow, green, and blue for the values 1 through 5, respectively. This gives a visual indication and creates a “heat map” showing progress over time as the colors change.

For the person responsible for a service, this tool is a good way to help evaluate the service as well as track progress.

### 55.4.3 Comparing Results Across Services

[Figure 55.2](#) shows an example spreadsheet being used to roll up the assessments to compare the services. The services should be listed by order of importance. Each number represents all eight assessments for that service. The number may be a weighted average or simply the mathematical “mode” (the most common number). Whatever method you use, be consistent.

| Team 1    | 2014Q1 | 2015Q1 | 2015Q2 | 2015Q3 | 2015Q4 |
|-----------|--------|--------|--------|--------|--------|
| Service A | 1      | 2      | 2      | 2      | 1      |
| Service B | 3      | 4      | 4      | 4      | 5      |
| Service C | 1      | 2      | 2      | 3      | 3      |

Figure 55.2: Roll-up to assess a team

Both of these spreadsheets should be filled out by the team as a group exercise. The manager's role is to hold everyone to high standards for accuracy. The manager should not do this assessment on his or her own and surprise the team with the result. A self-assessment has an inherent motivational factor; being graded by a manager is demotivating at best.

#### 55.4.4 Acting on the Results

We can now determine which services need the most improvement and, within each service, identify the problem areas. Create new projects focused on fixing the problem areas. We recommend focusing not on the easy problems to fix, but the ones that will have the greatest impact. That is, focus on the most needed improvements for the most important services.

Level 3 is often good enough for most services. Unless a service is directly revenue generating or has highly demanding requirements, generally one should achieve "a solid 3" across most responsibilities before expending effort on projects that would result in achieving Level 4 or 5.

Work on the selected projects during this quarter. At the start of the next quarter, repeat the assessment, update the spreadsheet, and repeat the process.

#### 55.4.5 Assessment and Project Planning Frequencies

The frequencies of assessments and project planning cycles do not have to be the same. Plan for monthly assessments and quarterly project cycles. Monthly assessment is a good frequency to track progress. More-frequent assessments

become a waste of time if very little measurable progress will have happened in that interval. Less-frequent assessments may make it difficult to spot new problems quickly.

The project cycle may be quarterly to sync up with performance review cycles or simply because three months is a reasonable amount of time for a project to be completed and show results. If the project selection occurs too infrequently, it may slow the cadence of change. If the project cycle is too frequent, it may discourage large, meaningful projects.

The longer the gap between assessments, the larger a burden they become. If the cycle is monthly, a meeting that lasts an hour or two can generally complete the assessment. A yearly cycle makes assessment “a big deal” and there is a temptation to stop all work for a week to prepare for day-long assessment meetings that analyze every aspect of the team. Such annual assessments become scary, unhelpful, and boring, and they usually turn into a bureaucratic waste of time. This is another case of small batches being better.

## **55.5 Organizational Assessments**

For an executive or manager who is responsible for many services, this assessment system is a good way to guide the team’s direction and track progress. These assessments also clearly communicate expectations to team members.

The people responsible for the service should do the assessments as a group. When people are allowed to do their own assessments, it motivates them to take on related improvement projects. You will be surprised at how often someone, seeing a low assessment, jumps at the chance to start a project that fixes related problems, even before you suggest it.

Your role as manager is to hold the team to high standards for accuracy and consistency as they complete the assessment. The rubric used should be consistent across all services and across all teams. If you use the assessment questions and look-for’s provided in [Chapter 56](#), “[Operational Assessments](#),” revise them to better suit your organization.

A culture of trust is required to get honest feedback. Assessments are for improvement, not punishment. The service is being assessed, not the people. Therefore we recommend against stack ranking teams, having a leaderboard, or other direct comparisons. Directly comparing teams drives bad behavior, and it creates shame and political stress. Stack ranking creates the perception that the assessment will be used to determine who should be punished. It encourages



teams to seek out quick fixes or to overstate their progress during assessments. By providing teams with their score and percentile, they see their need to improve without shaming them. When showing assessment data to executive management, anonymize the results so that the focus is on trends, rather than on specific individuals or teams. Rather than stack ranking teams, group results by business priority. For example, show aggregate results for the high-priority services, the medium-priority services, and the low-priority services. This not only anonymizes the results, but also provides a more actionable analysis.

[Figure 55.3](#) shows an example spreadsheet that could be used to roll up the assessments of many teams so they can be compared. Notice that teams 1, 4, and 5 have consistently improved over time, although team 4's progress has been slow. Team 3 had a bad year and is just starting to make progress. This team may need extra attention to assure the progress continues. Team 2's progress is erratic, flipping between Levels 3 and 4 with a small trend upward.

|        | 2014Q1 | 2015Q1 | 2015Q2 | 2015Q3 | 2015Q4 |
|--------|--------|--------|--------|--------|--------|
| Team 1 | 1      | 2      | 2      | 3      | 3      |
| Team 2 | 3      | 4      | 3      | 4      | 5      |
| Team 3 | 1      | 1      | 1      | 1      | 2      |
| Team 4 | 1      | 1      | 2      | 2      | 2      |
| Team 5 | 2      | 3      | 3      | 3      | 4      |

Figure 55.3: Roll-up to compare teams

High-performing employees will see such comparisons as a source of pride or inspiration to do better. Teams with high assessments should be encouraged to share their knowledge and best practices.

## 55.6 Levels of Improvement

Each CMM level builds on the next. Level 2 contains the seeds that enable Level 3; Level 3 contains the seeds that enable Level 4. For this reason organizations pass through the levels in order and levels cannot be skipped.

It is not important to get to Level 5 for all operational responsibilities of all



services. It would be a waste of resources to achieve Level 5 for a service that is little used and has low priority. In fact, it would be professionally negligent to expend the resources required to achieve a Level 5 assessment on a low-priority service when more important services need improvement.

Level 3 is designed to be good enough for most services. Going above that level should be reserved for high-priority services such as revenue-generating services or services that are particularly demanding.

When setting organizational goals, focus on fixing a problem, rather than achieving a particular assessment level. That is, never set a goal of improving the assessment of an operational responsibility for the sake of improving the assessment. That is putting the cart before the horse. Instead, identify a problem, engineer a solution, and measure the success or failure by whether the assessment improves. This is a subtle but important difference.

Setting the goal of an improved assessment drives the wrong behavior. It encourages people to create spot fixes that do not solve the larger problem or to fix unimportant-but-easy issues just to get a better assessment. It would be like paying teachers based on the grades their students receive; such a system would simply lead to each student receiving an A+ as exams become easier and easier.

For this reason it is equally wrong to set a goal of raising the assessment level of all services to a certain level. An engineer's time is scarce. A push to have a high assessment for every organizational responsibility results in expending those scarce resources on low-priority services in the name of raising an average. Such a pursuit creates bureaucratic handcuffs that strangle an organization. It is something you should never do; indeed, it is a trap that you should only hope your competition falls into.

This point cannot be overstated. If your management ever sets a corporate, division, or organizational goal of achieving a certain assessment level on all organizational responsibilities of all services, you are to show them this section of the book and tell them that such a plan is a recipe for disaster.

Likewise, one should never make raises and bonuses contingent on the results of an assessment. This discourages people from joining the projects that need the most help. Instead, encourage your best people to join projects where they can do the most good. Engineers are highly motivated by the opportunity to do good work. The best reward for an improved assessment is not money, but the opportunity to work on the most interesting project, or to receive notoriety by sharing knowledge with other teams. Rewards based on rising assessment levels are also not recommended, because often it is a major achievement to simply

retain a particular level in the face of challenging times.

## 55.7 Getting Started

Introducing this assessment system to a team can be a challenge. Most teams are not used to working in such a data-driven assessment environment. Instead, this system should be introduced and used as a self-improvement tool—a way to help teams aspire to do their very best.

To begin, the team should enumerate the major services it provides. For example, a team that is responsible for a large web site might determine that each web property is a service, each internal API is a service, and the common platform used to provide the services is a service. There may be multiple platforms, in which case each is considered a service. Finally, the infrastructure itself is often counted as a service as far as assessment is concerned.

Another example is a team that is responsible for a corporate email service. It might determine that the SMTP mail relays are a service, and each interface for mail clients is a service. Spam and malware detection and filtering are services, mailing list management is a service, user account management is a service, and user authentication and authorization is a service. The team would also determine that the interface to the HR systems is a service, email storage and archiving are services, and each interface to another system such as calendaring and an online meeting tool is a service. Finally, the platform(s) and infrastructure are services. Each of these services needs to be assessed by the email team as part of assessing the email service as a whole.

Assessments should be done on a periodic and repeatable schedule. The first Monday of each month is a common choice for frequency. The team meets and conducts a self-assessment of each service. Management's role is to maintain high standards and to ensure consistency across the services and teams. Management may set global standards for how certain ORs are evaluated. For example, there may be a corporate change management policy; compliance with that policy should be evaluated the same way for all services by all teams.

The eight core ORs should be used to assess all services. [Chapter 56](#), “[Operational Assessments](#),” includes details about these eight operational responsibilities, along with questions to ask during assessment to aid the team's understanding of the OR. The questions are followed by look-for's describing behaviors typically seen at various levels. Do not attempt to achieve every look-for. They are indicators, not requirements or checklists. Not every look-for is appropriate for every service.

As the monthly assessments progress, the changes over time should be apparent. The results of the assessments will help teams determine project priorities. Over time, the roll-ups described in this chapter can be used to compare services or teams.

---

### **Simplifying to Three Levels**

One way to make the system easier to use, and therefore make assessments easier to get started, is to simplify the system. Some organizations base assessments on three levels: (1) not demonstrating the desired behavior, (2) demonstrating the desired behavior, and (3) excelling at the desired behavior. Team members may debate endlessly over the semantic differences between CMM Levels 3 and 4, but as a manager you are probably most concerned with whether they are doing a particular best practice. Sometimes simpler is better.

---

## **55.8 Summary**

In this chapter we discussed how to assess the quality of operations and how to use this assessment to drive improvements.

Measuring the quality of system administration is complex. For each service we assess eight different qualities, called operational responsibilities: regular tasks (how normal, nonemergency tasks are performed), emergency response (how outages and other emergencies are handled), monitoring and metrics (collecting data used to make decisions), capacity planning (determining future resource needs), change management (how services are purposefully changed from birth to end), new service introduction and removal (how new products, hardware, or services are introduced into the environment and how they are removed), service deployment and decommissioning (how instances of an existing service are created and decommissioned), and performance and efficiency (how cost-effectively resources are used).

Each operational responsibility is assessed as being at one of five levels, reflecting the Capability Maturity Model (CMM) levels used in software engineering. The CMM is a set of maturity levels for assessing processes: Initial (ad hoc), Repeatable (documented, automated), Defined (roles and responsibilities are agreed upon), Managed (data-driven decisions), and Optimizing (improvements are made and the results measured). The first three levels of the CMM are the most important; the other levels are often attempted only for high-value services.

The eight responsibilities of a service are individually assessed, and then these assessment scores are rolled up to create an aggregate assessment of the service. Teams are typically responsible for many services. The individual service assessments are rolled up to assess the team, usually via a weighted average since not all services are equally important. These assessments can be used to rank teams.

By doing assessments periodically, progress can be tracked on the service, team, or organization level. High-ranking teams should be encouraged to share their best practices so others may adopt them.

A culture of trust is required to get honest feedback. Assessments are for improvement, not punishment. The service is being assessed, not the people. Data should be anonymized when shown to executive management.

The goal is to make improvements and measure their effectiveness by seeing whether the assessment outcome changes. The goal should not be to inflate assessment levels or to achieve a particular average assessment across a set of services.

This is not the only assessment system, nor is it perfect. Your organization may choose to modify this system to better suit its needs or create an entirely new system from scratch.

Using assessments to drive decisions in IT brings us closer to a system of scientific management for system administration and moves us away from “gut feelings” and intuition. The importance of the first three levels of the CMM is that they take us away from ad hoc processes and individual heroics and create repeatable processes that are more efficient and of higher quality.

## **Exercises**

1. Why is it difficult to assess the quality of operations work, or to compare two different operations teams?
2. Describe the five Capability Maturity Model (CMM) levels.
3. Rank the eight operational responsibilities from most important to least important. Justify your ranking.
4. How might assessments become too bureaucratic, and how might you prevent this?
5. Why is the mathematical mode recommended when rolling up a service's assessments, and a weighted average used when rolling up a team's assessments?

- 6.** In which other ways could you roll up assessments? Discuss their pros and cons.
- 7.** Select a service you are responsible for. Assess it based on the CMM levels for each of the operational responsibilities.
- 8.** Perform an assessment of a service you are not responsible for but for which you are familiar with its IT staff. Interview those personnel to assess the service based on the CMM levels for each of the operational responsibilities.
- 9.** The eight operational responsibilities may not be appropriate for all services in all companies. Which modifications, additions, or subtractions would you propose for a service you are involved in? Justify your answer.
- 10.** There are disadvantages to repeating the assessment cycle too frequently or not frequently enough. What would be the advantages and disadvantages of using a weekly cycle? A yearly cycle?
- 11.** This chapter advises against setting goals that specify achieving a particular CMM level. Relate the reason given to a personal experience (inside or outside of IT).

## Chapter 56. Operational Assessments

This chapter contains assessment questions and look-for's for various operational responsibilities (ORs) as described in [Chapter 55](#), “[Operational Excellence](#).” The instruction manual for how to use this process begins at [Section 55.7](#).

### Operational Responsibilities

| Page              | Section | Core                                           |
|-------------------|---------|------------------------------------------------|
| 1036              | 56.1    | Regular Tasks (RT)                             |
| 1039              | 56.2    | Emergency Response (ER)                        |
| 1041              | 56.3    | Monitoring and Metrics (MM)                    |
| 1043              | 56.4    | Capacity Planning (CP)                         |
| 1045              | 56.5    | Change Management (CM)                         |
| 1047              | 56.6    | New Product Introduction and Removal (NPI/NPR) |
| 1049              | 56.7    | Service Deployment and Decommissioning (SDD)   |
| 1051              | 56.8    | Performance and Efficiency (PE)                |
| <b>Additional</b> |         |                                                |
| 1054              | 56.9    | Service Delivery: The Build Phase              |
| 1056              | 56.10   | Service Delivery: The Deployment Phase         |
| 1058              | 56.11   | Toil Reduction                                 |
| 1060              | 56.12   | Disaster Preparedness                          |

### Assessment Levels

|         |                 |                                          |
|---------|-----------------|------------------------------------------|
| Level 1 | Initial/Chaotic | Ad hoc and relying on individual heroics |
| Level 2 | Repeatable      | Repeatable results                       |
| Level 3 | Defined         | Responsibilities defined/confirmed       |
| Level 4 | Managed         | Quantitatively managed metrics           |
| Level 5 | Optimizing      | Deliberate optimization/improvement      |

### 56.1 Regular Tasks (RT)

Regular Tasks include how normal, nonemergency, operational duties are handled—that is, how work is received, queued, distributed, processed, and verified, plus how periodic tasks are scheduled and performed. All services have

some kind of normal scheduled or unscheduled work that needs to be done. Often web operations teams do not perform direct customer support but rather handle interteam requests, requests from stakeholders, and escalations from direct customer support teams.

## **Sample Assessment Questions**

- What are the common and periodic operational tasks and duties?
- Is there a playbook for common operational duties?
- What is the SLA for regular requests?
- How is the need for new playbook entries identified? Who may write new entries? Edit existing ones?
- How are requests from users received and tracked?
- Is there a playbook for common user requests?
- How often are user requests not covered by the playbook?
- How do users engage us for support (online and physical locations)?
- How do users know how to engage us for support?
- How do users know what is supported and what isn't?
- How do we respond to requests for support of the unsupported?
- What are the limits of regular support (hours of operation, remote or on-site)? How do users know these limits?
- Are different size categories handled differently? How is size determined?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

## **Level 1: Initial**

- There is no playbook, or it is outdated and unused.
- Results are inconsistent.
- Different people do tasks differently.
- Two users requesting the same thing usually get different results.
- Processes aren't documented.
- The team can't enumerate all the processes a team does (even at a high level).
- Requests get lost or stalled indefinitely.

- The organization cannot predict how long common tasks will take to complete.
- Operational problems, if reported, don't get attention.

## **Level 2: Repeatable**

- There is a finite list of which services are supported by the team.
- Each end-to-end process has each step enumerated, with dependencies.
- Each end-to-end process has each step's process documented.
- Different people do the tasks the same way.
- There is some duplication of effort seen in the flow.
- Some information needed by multiple tasks may be recreated by each step that needs it.

## **Level 3: Defined**

- The team has an SLA defined for most requests, though it may not be adhered to.
- Each step has a QA checklist completed before handing off to the next step.
- Teams learn of process changes by other teams ahead of time.
- Information or processing needed by multiple steps is created once.
- There is no (or minimal) duplication of effort.
- The ability to turn up new capacity is a repeatable process.

## **Level 4: Managed**

- The defined SLA is measured.
- There are feedback mechanisms for all steps.
- There is periodic (possibly weekly) review of defects and reworks.
- Postmortems are published for all to see, with a draft report available within x hours and a final report completed within y days.
- There is periodic review of alerts by the affected team. There is periodic review of alerts by a cross-functional team.
- Process change requests require data to measure the problem being fixed.
- Dashboards report data in business terms (i.e., not just technical terms).
- Every failover procedure has a "date of last use" dashboard.



- Capacity needs are predicted ahead of need.

## **Level 5: Optimizing**

- After process changes are made, before/after data are compared to determine success.
- Process changes are reverted if before/after data shows no improvement.
- Process changes that have been acted on come from a variety of sources.
- At least one process change has come from every step (in recent history).
- Cycle time enjoys month-over-month improvements.
- Decisions are supported by modeling “what if” scenarios using extracted actual data.

## **56.2 Emergency Response (ER)**

Emergency Response covers how outages and disasters are handled. This includes engineering resilient systems that prevent outages plus technical and nontechnical processes performed during and after outages (response and remediation).

### **Sample Assessment Questions**

- How are outages detected (automatic monitoring, user complaints)?
- Is there a playbook for common failover scenarios and outage-related duties?
- Is there an oncall calendar?
- How is the oncall calendar created?
- Can the system withstand failures on the local level (component failure)?
- Can the system withstand failures on the geographic level (alternative datacenters)?
- Are staff geographically distributed (i.e., can other regions cover for each other for extended periods of time)?
- Do you write postmortems? Is there a deadline for when a postmortem must be completed?
- Is there a standard template for postmortems?
- Are postmortems reviewed to ensure action items are completed?
- If there is a corporate standard practice for this OR, what is it and how

does this service comply with the practice?

### **Level 1: Initial**

- Outages are reported by users rather than by a monitoring system.
- No one is ever oncall, a single person is always oncall, or everyone is always oncall.
- There is no oncall schedule.
- There is no oncall calendar.
- There is no playbook of what to do for various alerts.

### **Level 2: Repeatable**

- A monitoring system contacts the oncall person.
- There is an oncall schedule with an escalation plan.
- There is a repeatable process for creating the next month's oncall calendar.
- A playbook item exists for any possible alert.
- A postmortem template exists.
- Postmortems are written occasionally but not consistently.
- Oncall coverage is geographically diverse (multiple time zones).

### **Level 3: Defined**

- Outages are classified by size (i.e., minor, major, catastrophic).
- Limits (and minimums) for how often people should be oncall are defined.
- Postmortems are written for all major outages.
- There is an SLA defined for alert response: initial, hands-on-keyboard, issue resolved, postmortem complete.

### **Level 4: Managed**

- The oncall pain is shared by the people most able to fix problems.
- How often people are oncall is verified against the policy.
- Postmortems are reviewed.
- There is a mechanism to triage recommendations in postmortems and assure they are completed.
- The SLA is actively measured.

## **Level 5: Optimizing**

- Stress testing and failover testing are done frequently (quarterly or monthly).
- “Game Day” exercises (intensive, system-wide tests) are done periodically.
- The monitoring system alerts before outages occur (indications of “sick” systems rather than “down” systems).
- Mechanisms exist so that any failover procedure not utilized in recent history is activated artificially.
- Experiments are performed to improve SLA compliance.

## **56.3 Monitoring and Metrics (MM)**

Monitoring and Metrics covers collecting and using data to make decisions. Monitoring collects data about a system. Metrics uses that data to measure a quantifiable component of performance. This includes technical metrics such as bandwidth, speed, or latency; derived metrics such as ratios, sums, averages, and percentiles; and business goals such as the efficient use of resources or compliance with an SLA.

## **Sample Assessment Questions**

- Is the service level objective (SLO) documented? How do you know your SLO matches customer needs?
- Do you have a dashboard? Is it in technical or business terms?
- How accurate are the collected data and the predictions? How do you know?
- How efficient is the service? Are machines over-or under-utilized? How is utilization measured?
- How is latency measured?
- How is availability measured?
- How do you know if the monitoring system itself is down?
- How do you know if the data used to calculate key performance indicators (KPIs) is fresh? Is there a dashboard that shows measurement freshness and accuracy?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

## **Level 1: Initial**

- No SLOs are documented.
- If there is monitoring, not everything is monitored, and there is no way to check completeness.
- Systems and services are manually added to the monitoring system, if at all—there is no process.
- There are no dashboards.
- There is little or no measurement or metrics.
- You think customers are happy but they aren't.
- It is common (and rewarded) to enact optimizations that benefit a person or small group to the detriment of the larger organization or system.
- Departmental goals emphasize departmental performance to the detriment of organizational performance.

## **Level 2: Repeatable**

- The process for creating machines/server instances assures they will be monitored.

## **Level 3: Defined**

- SLOs are documented.
- Business KPIs are defined.
- The freshness of business KPI data is defined.
- A system exists to verify that all services are monitored.
- The monitoring system itself is monitored (meta-monitoring).

## **Level 4: Managed**

- SLOs are documented and monitored.
- Defined KPIs are measured.
- Dashboards exist showing each step's completion time; the lag time of each step is identified.
- Dashboards exist showing current bottlenecks, backlogs, and idle steps.
- Dashboards show defect and rework counts.
- Capacity planning is performed for the monitoring system and all analysis systems.

- The freshness of the data used to calculate KPIs is measured.

## **Level 5: Optimizing**

- The accuracy of collected data is verified through active testing.
- KPIs are calculated using data that is less than a minute old.
- Dashboards and other analysis displays are based on fresh data.
- Dashboards and other displays load quickly.
- Capacity planning for storage, CPU, and network of the monitoring system is done with the same sophistication as any major service.

## **56.4 Capacity Planning (CP)**

Capacity Planning covers determining future resource needs. All services require some kind of planning for future resources. Services tend to grow. Capacity planning involves the technical work of understanding how many resources are needed per unit of growth, plus nontechnical aspects such as budgeting, forecasting, and supply chain management.

## **Sample Assessment Questions**

- How much capacity do you have now?
- How much capacity do you expect to need three months from now?  
Twelve months from now?
- Which statistical models do you use for determining future needs?
- How do you load-test?
- How much time does capacity planning take? What could be done to make it easier?
- Are metrics collected automatically?
- Are metrics available continuously or do you need to initiate a process to request them?
- Is capacity planning the job of no one, everyone, a specific person, or a team of capacity planners?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

## **Level 1: Initial**

- No inventory is kept.

- The system runs out of capacity from time to time.
- Determining how much capacity to add is done by tradition, guessing, or luck.
- Operations is reactive about capacity planning, often not being able to fulfill the demand for capacity in time.
- Capacity planning is everyone's job, and therefore no one's job.
- No one is specifically assigned to handle CP duties.
- A large amount of headroom exists rather than knowing precisely how much slack is needed.

## **Level 2: Repeatable**

- CP metrics are collected on demand, or only when needed.
- The process for collecting CP metrics is written and repeatable.
- Load testing is done occasionally, perhaps when a service is new.
- Inventory of all systems is accurate, possibly due to manual effort.

## **Level 3: Defined**

- CP metrics are automatically collected.
- Capacity required for a certain amount of growth is well defined.
- There is a dedicated CP person on the team.
- CP requirements are defined at a subsystem level.
- Load testing is triggered by major software and hardware changes.
- Inventory is updated as part of capacity changes.
- The amount of headroom needed to survive typical surges is defined.

## **Level 4: Managed**

- CP metrics are collected continuously (daily/weekly instead of monthly or quarterly).
- Additional capacity is gained automatically, with human approval.
- Performance regressions are detected during testing, involving CP if performance regression will survive into production (i.e., it is not a bug).
- Dashboards include CP information.
- Changes in correlation are automatically detected and raise a ticket for CP to verify and adjust relationships between core drivers and resource units.

- Unexpected increases in demand are automatically detected using MACD metrics (or a similar), which generates a ticket for the CP person or team.
- The amount of headroom in the system is monitored.

## **Level 5: Optimizing**

- Past CP projections are compared with actual results.
- Load testing is done as part of a continuous test environment.
- The team employs a statistician.
- Additional capacity is gained automatically.
- The amount of headroom is systematically optimized to reduce waste.

## **56.5 Change Management (CM)**

Change Management covers how services are deliberately changed over time. This includes the software delivery platform—the steps involved in a software release: develop, build, test, and push into production. For hardware, this includes firmware upgrades and minor hardware revisions.

## **Sample Assessment Questions**

- How often are deployments (releases pushed into production)?
- How much human labor does a deployment require?
- When a release is received, does the operations team need to change anything in it before it is pushed?
- How does operations know if a release is major or minor (a big or small change)? How are these types of releases handled differently?
- How does operations know if a release is successful?
- How often have releases failed?
- How does operations know that new releases are available?
- Are there change-freeze windows?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

## **Level 1: Initial**

- Deployments are done sparingly, as they are very risky.
- The deployment process is ad hoc and laborious.

- Developers notify operations of new releases when a release is ready for deployment.
- Releases are not deployed until weeks or months after they are available.
- Operations and developers bicker over when to deploy releases.

## **Level 2: Repeatable**

- The deployment is no longer ad hoc.
- Deployment is manual but consistent.
- Releases are deployed as delivered.
- Deployments fail often.

## **Level 3: Defined**

- What constitutes a successful deployment is defined.
- Minor and major releases are handled differently.
- The expected time gap between release availability and deployment is defined.

## **Level 4: Managed**

- Deployment success/failure is measured against definitions.
- Deployments fail rarely.
- The expected time gap between release availability and deployment is measured.

## **Level 5: Optimizing**

- Continuous deployment is in use.
- Failed deployments are extremely rare.
- New releases are deployed with little delay.

## **56.6 New Product Introduction and Removal (NPI/NPR)**

New Product Introduction and Removal covers how new products and services are introduced into the environment and how they are removed. This is a coordination function: Introducing a new product or service requires a support infrastructure that may touch multiple teams.

For example, before a new model of computer hardware is introduced into the



datacenter environment, certain teams must have access to sample hardware for testing and qualification, the purchasing department must have a process to purchase the machines, and datacenter technicians need documentation. For introducing software and services, there should be tasks such as requirements gathering, evaluation and procurement, licensing, and creation of playbooks for the helpdesk and operations.

Product removal might involve finding all machines with a particularly old release of an operating system and seeing that all of them get upgraded. Product removal requires identifying current users, agreeing on timelines for migrating them away, updating documentation, and eventually decommissioning the product, any associated licenses, maintenance contracts, monitoring, and playbooks. The majority of the work consists of communication and coordination between teams.

## **Sample Assessment Questions**

- How is new hardware introduced into the environment? Which teams are involved and how do they communicate? How long does the process take?
- How is old hardware or software eliminated from the system?
- What is the process for disposing of old hardware?
- Which steps are taken to ensure disks and other storage are erased when disposed?
- How is new software or a new service brought into being? Which teams are involved and how do they communicate? How long does the process take?
- What is the process for handoff between teams?
- Which tools are used?
- Is documentation current?
- Which steps involve human interaction? How could it be eliminated?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

## **Level 1: Initial**

- New products are introduced through ad hoc measures and individual heroics.
- Teams are surprised by NPI, often learning they must deploy something into production with little notice.

- NPI is delayed due to lack of capacity, miscommunication, or errors.
- Deprecating old products is rarely done, resulting in operations having to support an “infinite” number of hardware or software versions.

## **Level 2: Repeatable**

- The process used for NPI/NPR is repeatable.
- The handoff between teams is written and agreed upon.
- Each team has a playbook for tasks related to its involvement with NPR/NPI.
- Equipment erasure and disposal is documented and verified.

## **Level 3: Defined**

- Expectations for how long NPI/NPR will take are defined.
- The handoff between teams is encoded in a machine-readable format.
- Members of all teams understand their role as it fits into the larger, overall process.
- The maximum number of products supported by each team is defined.
- The list of each team’s currently supported products is available to all teams.

## **Level 4: Managed**

- There are dashboards for observing NPI and NPR progress.
- The handoff between teams is actively revised and improved.
- The number of no-longer-supported products is tracked.
- Decommissioning no-longer-supported products is a high priority.

## **Level 5: Optimizing**

- NPI/NPR tasks have become API calls between teams.
- NPI/NPR processes are self-service by the team responsible.
- The handoff between teams is a linear flow (or for very complex systems, joining multiple linear flows).

## **56.7 Service Deployment and Decommissioning (SDD)**

Service Deployment and Decommissioning covers how instances of an existing

service are created and how they are turned off (decommissioned). After a service is designed, it is usually deployed repeatedly. Deployment may involve turning up satellite replicas in new datacenters or creating a development environment of an existing service. Decommissioning could be part of turning down a datacenter, reducing excess capacity, or turning down a particular service instance such as a demo environment.

## **Sample Assessment Questions**

- What is the process for turning up a service instance?
- What is the process for turning down a service instance?
- How is new capacity added? How is unused capacity turned down?
- Which steps involve human interaction? How could it be eliminated?
- How many teams touch these processes?
- Do all teams know how they fit into the overall picture?
- What is the workflow from team to team?
- Which tools are used?
- Is documentation current?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

### **Level 1: Initial**

- The process is undocumented and haphazard. Results are inconsistent.
- The process is defined by who does something, not what is done.
- Requests get delayed due to miscommunication, lack of resources, or other avoidable reasons.
- Different people do the tasks differently.

### **Level 2: Repeatable**

- The processes required to deploy or decommission a service are understood and documented.
- The process for each step is documented and verified.
- Each step has a QA checklist completed before handing off to the next step.
- Teams learn of process changes by other teams ahead of time.

- Information or processing needed by multiple steps is created once.
- There is no (or minimal) duplication of effort.
- The ability to turn up new capacity is a repeatable process.
- Equipment erasure and disposal is documented and verified.

### **Level 3: Defined**

- The SLA for how long each step should take is defined.
- For physical deployments, standards for removal of waste material (boxes, wrappers, containers) are based on local environmental standards.
- For physical decommissions, standards for disposing of old hardware are based on local environmental standards as well as the organization's own standards for data erasure.
- Tools exist to implement many of the steps and processes.

### **Level 4: Managed**

- The defined SLA for each step is measured.
- There are feedback mechanisms for all steps.
- There is periodic review of defects and reworks.
- Capacity needs are predicted ahead of need.
- Equipment disposal compliance is measured against organization standards as well as local environmental law.
- Waste material (boxes, wrappers, containers) involved in deployment is measured.
- Quantity of equipment disposal is measured.

### **Level 5: Optimizing**

- After process changes are made, before/after data are compared to determine success.
- Process changes are reverted if before/after data shows no improvement.
- Process changes that have been acted on come from a variety of sources.
- Cycle time enjoys month-over-month improvements.
- Decisions are supported by modeling "what if" scenarios using extracts from actual data.
- Equipment disposal is optimized by the reduction of equipment

deployment.

## **56.8 Performance and Efficiency (PE)**

Performance and Efficiency covers how cost-effectively resources are used and how well the service performs. A running service needs to have good performance without wasting resources. We can generally improve performance by using more resources, or we may be able to improve efficiency to the detriment of performance. Achieving both requires a large effort to bring about equilibrium. Cost efficiency is calculated as the cost of resources divided by the quantity of use. Resource efficiency is calculated as the quantity of resources divided by the quantity of use. To calculate these statistics, one must know how many resources exist; thus, some kind of inventory is required.

### **Sample Assessment Questions**

- What is the formula used to measure performance?
- What is the formula used to determine utilization?
- What is the formula used to determine resource efficiency?
- What is the formula used to determine cost efficiency?
- How is performance variation measured?
- Are performance, utilization, and resource efficiency monitored automatically? Is there a dashboard for each?
- Is there an inventory of the machines and servers used in this service?
- How is the inventory kept up-to-date?
- How would you know if something was missing from the inventory?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

### **Level 1: Initial**

- Performance and utilization are not consistently measured.
- What is measured depends on who set up the systems and services.
- Resource efficiency is not measured.
- Performance problems often come as a surprise and are hard to diagnose and resolve because there is insufficient data.
- Inventory is not up-to-date.
- Inventory may or may not be updated, depending on who is involved in

receiving or disposing of items.

## **Level 2: Repeatable**

- All metrics relevant to performance and utilization are collected across all systems and services.
- The process for bringing up new systems and services is documented and everyone follows the process.
- Systems are associated with services when configured for use by a service, and disassociated when released.
- Inventory is up-to-date. The inventory process is well documented and everyone follows the process.

## **Level 3: Defined**

- Both performance and utilization monitoring are automatically configured for all systems and services during installation and removed during decommissioning.
- Performance targets for each service are defined.
- Resource usage targets for each service are defined.
- Formulas for service-oriented performance and utilization metrics are defined.
- Performance of each service is monitored continuously.
- Resource utilization of each service is monitored continuously.
- Idle capacity that is not currently used by any service is monitored.
- The desired amount of headroom is defined.
- The roles and responsibilities for keeping the inventory up-to-date are defined.
- Systems for tracking the devices that are connected to the network and their hardware configurations are in place.

## **Level 4: Managed**

- Dashboards track performance, utilization, and resource efficiency.
- Minimum, maximum, and ninetieth percentile headroom are tracked and compared to the desired headroom and are visible on a dashboard.
- Goals for performance and efficiency are set and tracked.

- There are periodic reviews of performance and efficiency goals and status for each service.
- KPIs are used to set performance, utilization, and resource efficiency goals that drive optimal behavior.
- Automated systems track the devices that are on the network and their configurations and compare them with the inventory system, flagging problems when they are found.

## **Level 5: Optimizing**

- Bottlenecks are identified using the performance dashboard. Changes are made as a result.
- Services that use large amounts of resources are identified and changes are made.
- Changes are reverted if the changes do not have a positive effect.
- Computer hardware models are regularly evaluated to find models where utilization of the different resources is better balanced.
- Other sources of hardware and other hardware models are regularly evaluated to determine if cost efficiency can be improved.

---

### **Case Study: The Unexpectedly Slow Cache**

Stack Overflow purchased a product that would accelerate web page delivery to customers using a globally distributed cache. Most customers deploy this product and assume it has a “can’t lose” benefit.

Before deploying it, Stack Overflow engineer Nick Craver created a framework for measuring end-to-end page load times. The goal was to precisely know how much improvement was gained both globally and for customers in various geographic regions.

Nick was quite surprised to discover that the product degraded performance. It improved certain aspects but only to the detriment of others, resulting in a net performance loss.

Stack Overflow worked with the vendor to identify the problem. As a result, a major design error was found and fixed.

If care hadn’t been taken to measure performance before and after the change, Stack Overflow’s efforts would have unknowingly made its service slower. One wonders how many other customers of this product did no such measurements and simply assumed performance was

improved while in reality it was made worse.

---

## 56.9 Service Delivery: The Build Phase

Service delivery is the technical process of how a service is created. It starts with source code created by developers and ends with a service running in production.

This OR is optional. It is for enterprises that develop their own applications or services, as well as teams that write code for the purpose of gluing together third-party products or configuration management. It may not apply to companies that purchase software rather than develop it in-house.

### Sample Assessment Questions

- How is software built from source code to packages?
- Is the final package built from source or do developers deliver precompiled elements?
- What percentage of code is covered by unit tests?
- Which tests are fully automated?
- Are metrics collected about bug lead time, code lead time, and patch lead time?
- To build the software, do all raw source files come from version control repositories?
- To build the software, how many places (repositories or other sources) are accessed to attain all raw source files?
- Is the resulting software delivered as a package or a set of files?
- Is everything required for deployment delivered in the package?
- Which package repository is used to hand off the results to the deployment phase?
- Is there a single build console for status and control of all steps?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

### Level 1: Initial

- Each person builds in his or her own environment.
- People check in code without checking that it builds.



- Developers deliver precompiled elements to be packaged.
- Little or no unit testing is performed.
- No metrics are collected.
- Version control systems are not used to store source files.
- Building the software is a manual process or has manual steps.
- The master copies of some source files are kept in personal home directories or computers.

## **Level 2: Repeatable**

- The build environment is defined; everyone uses the same system for consistent results.
- Building the software is still done manually.
- Testing is done manually.
- Some unit tests exist.
- Source files are kept in version-controlled repositories.
- Software packages are used as the means of delivering the end result.
- If multiple platforms are supported, each is repeatable, though possibly independently.

## **Level 3: Defined**

- Building the software is automated.
- Triggers for automated builds are defined.
- Expectations around unit test coverage are defined; they are less than 100 percent.
- Metrics for bug lead time, code lead time, and patch lead time are defined.
- Inputs and outputs of each step are defined.

## **Level 4: Managed**

- Success/fail build ratios are measured and tracked on a dashboard.
- Metrics for bug lead time, code lead time, and patch lead time are collected automatically.
- Metrics are presented on a dashboard.
- Unit test coverage is measured and tracked.

## **Level 5: Optimizing**

- Metrics are used to select optimization projects.
- Attempts to optimize the process involve collecting before and after metrics.
- Each developer can perform the end-to-end build process in his or her own sandbox before committing changes to a centralized repository.
- Insufficient unit test code coverage stops production.
- If multiple platforms are supported, building for one is as easy as building for them all.
- The software delivery platform is used for building infrastructure as well as applications.

## **56.10 Service Delivery: The Deployment Phase**

The goal of the deployment phase is to create a running environment. The deployment phase creates the service in one or more testing and production environments.

### **Sample Assessment Questions**

- How is software delivered? (In which format and by which method?)
- How is functionality tested?
- How are packages deployed in production?
- How much downtime is required to deploy the service in production?
- Are metrics collected about frequency of deployment, mean time to restore service, and change success rate?
- How is the decision made to promote a package from testing to production?
- Which kind of testing is done (system, performance, load, user acceptance)?
- How is deployment handled differently for small, medium, and large releases?
- If there is a corporate standard practice for this OR, what is it and how does this service comply?

## **Level 1: Initial**

- Software is delivered differently each time, possibly in ways that are confusing or undocumented (different formats, transmitted via different mechanisms).
- Functionality is not tested, or different team members do a different set of manual tests.
- Deployment involves or requires manual steps.
- Deployments into the testing and production environments are different processes, each with its own tools and procedures.
- Different people on the team perform deployments differently.
- Deployment requires downtime, and sometimes significant downtime.
- How a release is promoted to production is ad hoc or ill defined.
- Testing is manual, ill defined, or not done.

## **Level 2: Repeatable**

- Software is delivered via the same file format and mechanism each time.
- Functionality is tested using a documented list of tests, either manually or via automation. The expected output of tests is not clearly defined, other than looking for obvious crashes or error messages.
- Deployment is performed in a documented, repeatable process.
- If deployment requires downtime, it is predictable.
- Testing procedures are documented and repeatable.

## **Level 3: Defined**

- Which functionality is to be tested is well defined, with input from stakeholders. There is a process for defining how tests will be developed for new functionality. The expected results are defined and verified during testing.
- Metrics for frequency of deployment, mean time to restore service, and change success rate are defined.
- How downtime due to deployments is to be measured is defined; limits and expectations are defined.
- How a release is promoted to production is defined.
- Testing results are clearly communicated to all stakeholders.

## **Level 4: Managed**

- Metrics related to test failures are collected and reported.
- Metrics for frequency of deployment, mean time to restore service, and change success rate are collected automatically.
- Metrics are presented on a dashboard.
- Downtime due to deployments is measured automatically.
- Reduced production capacity during deployment is measured.
- Tests are fully automated.

## **Level 5: Optimizing**

- Metrics are used to select optimization projects.
- Attempts to optimize the process involve collecting before and after metrics.
- Deployment is fully automated.
- Promotion decisions are fully automated, perhaps with a few specific exceptions.
- Deployment requires no downtime.

## **56.11 Toil Reduction**

Toil Reduction is the process by which we improve the use of people within our system. When we reduce toil (i.e., exhausting physical labor), we create a more sustainable working environment for operational staff. While reducing toil is not a service per se, this OR can be used to assess the amount of toil and determine whether practices are in place to limit the amount of toil.

## **Sample Assessment Questions**

- How many hours each week are spent on coding versus non-coding projects?
- What percentage of time is spent on project work versus manual labor that could be automated?
- What percentage of time spent on manual labor should raise a red flag?
- What is the process for detecting that the percentage of manual labor has exceeded the red flag threshold?
- What is the process for raising a red flag? Whose responsibility is it?
- What happens after a red flag is raised? When is it lowered?

- How are projects for reducing toil identified? How are they prioritized?
- How is the effectiveness of those projects measured?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

### **Level 1: Initial**

- Toil is not measured and grows until no project work, or almost no project work, can be accomplished.
- There is no process for raising a red flag.
- Some individuals recognize when toil is becoming a problem and look for solutions, but others are unaware of the problem.
- Individuals choose to work on the projects that are the most interesting to them, without looking at which projects will have the biggest impact.

### **Level 2: Repeatable**

- The amount of time spent on toil versus on projects is measured.
- The percentage of time spent on toil that constitutes a problem is defined and communicated.
- The process for raising a red flag is documented and communicated.
- Individuals track their own toil to project work ratios, and are individually responsible for raising a red flag.
- Red flags may not always be raised when they should be.
- The process for identifying which projects will have the greatest impact on toil reduction is defined.
- The method for prioritizing projects is documented.

### **Level 3: Defined**

- For each team, the person responsible for tracking toil and raising a red flag is identified.
- The people involved in identifying and prioritizing toil-reduction are known.
- Both a red flag level of toil and a target level are defined. The red flag is lowered when toil reaches the target level.
- During the red flag period, the team works on only the highest-impact toil-reduction projects.

- During the red flag period, the team has management support for putting other projects on hold until toil is reduced to a target level.
- After each step in a project, statistics on toil are closely monitored, providing feedback on any positive or negative changes.

### **Level 4: Managed**

- Project time versus toil is tracked on a dashboard, and the amount of time spent on each individual project or manual task is also tracked.
- Red flags are raised automatically, and the dashboard gives an overview of where the problems lie.
- The time-tracking data is monitored for trends that give an early alert for teams that are showing an increase in toil in one or more areas.
- KPIs are defined and tracked to keep toil within the desired range and minimize the red flag periods.

### **Level 5: Optimizing**

- The target and red flag levels are adjusted and the results are monitored as to their effects on the overall flow, performance, and innovation.
- Changes to the main project prioritization process are introduced and evaluated for positive or negative impact, including the impact on toil.
- Changes to the red flag toil-reduction task prioritization process are introduced and evaluated.

## **56.12 Disaster Preparedness**

An operations organization needs to be able to handle outages well, and it must have practices that reduce the chance of repeating past mistakes. Disasters and major outages happen. Everyone in the company from the top down needs to recognize that fact, and adopt a mind-set that accepts outages and learns from them. Systems should be designed to be resilient to failure.

### **Sample Assessment Questions**

- What is the SLA? Which tools and processes are in place to ensure that the SLA is met?
- How complete are the playbooks?
- When was each scenario in the playbooks last exercised?

- What is the mechanism for exercising different failure modes?
- How are new team members trained to be prepared to handle disasters?
- Which roles and responsibilities apply during a disaster?
- How do you prepare for disasters?
- How are disasters used to improve future operations and disaster response?
- If there is a corporate standard practice for this OR, what is it and how does this service comply with the practice?

### **Level 1: Initial**

- Disasters are handled in an ad hoc manner, requiring individual heroics.
- Playbooks do not exist, or do not cover all scenarios.
- Little or no training exists.
- Service resiliency and different failure scenarios are never tested.

### **Level 2: Repeatable**

- Playbooks exist for all failure modes, including large-scale disasters.
- New team members receive on-the-job training.
- Disasters are handled consistently, independent of who is responding.
- If multiple team members respond, their roles, responsibilities, and handoffs are not clearly defined, leading to some duplication of effort.

### **Level 3: Defined**

- The SLA is defined, including dates for postmortem reports.
- Handoff procedures are defined, including checks to be performed and documented.
- How to scale the responding team to make efficient use of more team members is defined.
- The roles and responsibilities of team members in a disaster are defined.
- Specific disaster preparedness training for new team members is defined and implemented.
- The team has regular disaster preparedness exercises.
- The exercises include fire drills performed on the live service.
- After every disaster, a postmortem report is produced and circulated.

## **Level 4: Managed**

- The SLA is tracked using dashboards.
- The timing for every step in the process from the moment the event occurred is tracked on the dashboard.
- A program for disaster preparedness training ensures that all aspects are covered.
- The disaster preparedness program measures the results of disaster preparedness training.
- As teams become better at handling disasters, the training expands to cover more complex scenarios.
- Teams are involved in cross-functional fire drills that involve multiple teams and services.
- Dates for publishing initial and final postmortem reports are tracked and measured against the SLA.

## **Level 5: Optimizing**

- Areas for improvement are identified from the dashboards.
- New techniques and processes are tested and the results measured and used for further decision making.
- Automated systems ensure that every failure mode is exercised within a certain period, by artificially causing a failure if one has not occurred naturally.



# Epilogue

We began this book by asking for a concise definition of system administration. Now we're no closer to an answer. If anything, we've broadened the definition. Rather than building a crisper definition of system administration, we've discussed customer support, repair, operations, architecture definition, deployment, disaster planning, and even management skills. System administration is an extremely broad field, and no simple definition can cover it all.

We hope that you've learned a lot from reading this book. We've certainly learned a lot by writing it. Having to put into words things that had become second nature has forced us to think hard about everything we do, every habit we've developed. The peer-review process stands us naked in front of our mentors and comrades to receive criticism of our fundamental beliefs. We're better for writing this book, and we hope you are better for reading it. We hope that someday you write a book and enjoy the same exhilaration.

The most exciting part of this book has been to record, in such a permanent form, the rants and anecdotes that we have accumulated over our careers. We respond to certain technical and nontechnical issues by getting on our soapboxes to expound our opinions. These monologues are refined every time we repeat them, until we find ourselves repeating them word for word, over and over again. We can honestly say that this book includes every tub-thumping rant we authors blurt out with Pavlovian predictability. With a little bit of luck, these rants will stand the test of time. This book also captures every useful anecdote in our library of experience. Each anecdote teaches an important lesson or two. We can rest assured that these anecdotes will not be lost, and we can safely look forward to all the new anecdotes we will accrue in the future.

System administration is a culture. Every culture has its anecdotes, myths, and stories. It is how we pass our history to new generations and propagate the lessons and values that are important to us. We learn best from hearing our culture's stories and anecdotes. We enrich the culture every time we share a new one.

We'd like to share with you one final anecdote.

---

## **A Concise Definition**

A facility had several researchers from a variety of universities visiting for

the summer. That autumn, after they left, the SAs had to decommission their computers and clean the large room they had been sharing. The SAs found a scrap of paper that had been taped near the phone. It simply said, “Makes things work,” followed by the phone number of the SAs.

It was absolutely the highest compliment they had ever received.

---

# **Part XI: Appendices**

# Appendix A. What to Do When . . .

In this appendix we pull together the various elements from the rest of the book to provide an overview of how they can be used to deal with everyday situations or to answer common questions system administrators and managers often have.

## A.1 Building a Site from Scratch

- Think about the organizational structure you need—[Chapter 48](#).
- Check in with management on the business priorities that will drive implementation priorities.
- Plan your namespaces carefully—[Chapter 39](#).
- Build a rock-solid datacenter or computer room.
- Build a rock-solid network that is designed to grow—[Chapter 23](#).
- Build services that will scale—[Chapter 18](#).
- Build a software depot, or at least plan a small directory hierarchy that can grow into a software depot—[Chapter 45](#).
- Establish your initial core application services:
  - Authentication and authorization (LDAP/Kerberos/ActiveDirectory) —[Chapter 40](#)
  - Desktop life-cycle management—[Chapter 7](#)
  - Email—[Chapter 41](#)
  - Data storage—[Chapter 43](#)
  - Backups—[Chapter 44](#)
  - Network configuration—[Chapter 24](#)
  - Printing—[Chapter 42](#)

## A.2 Growing a Small Site

- Know when to provide a helpdesk—[Chapter 27](#).
- Establish checklists for new hires, new desktops/laptops, and new servers —page 214.
- Consider the benefits of a network operations center (NOC) dedicated to monitoring and coordinating network operations—[Chapter 38](#).
- Think about your organization and whom you need to hire, and provide

service statistics showing open and resolved problems—[Chapter 48](#).

- Monitor services for both capacity and availability so that you can predict when you will need to scale them—[Chapter 38](#).
- Be ready for an influx of new computers, employees, and SAs—[Sections A.24](#), [A.25](#), and [A.26](#).

### **A.3 Going Global**

- Design your wide area network (WAN) architecture—[Chapter 23](#).
- Follow three cardinal rules: scale, scale, and scale.
- Standardize server times on Universal Time (UTC) to maximize log analysis capabilities.
- Make sure that your helpdesk really is 24/7. Look at ways to leverage SAs in other time zones—[Chapter 27](#).
- Architect services to take account of long-distance links, which usually have lower bandwidth and are less reliable—[Chapter 18](#).
- Qualify applications for use over high-latency links—[Section 18.2.2](#).
- Ensure that your security and permissions structures are still adequate under global operations.

### **A.4 Choosing Checklists to Put on Your Wiki**

- What to do for each new employee
- What to do for each exiting employee
- How to change the administrative (root) password globally
- How to set up a new machine (desktop, laptop, server)
- How to decommission an old machine
- For any system that can fail over to a stand-by system, how to do the failover
- Security-related processes that you want to ensure you get right (for example, disconnecting a terminated employee from the network) • Any periodic process for maintaining the data backup system
- How to restore data from the backup system
- For each service, any add/change/delete processes that may exist

### **A.5 Fixing the Perception of Being Unprofessional**

- Make sure the most visible services are delivered consistently. Chasing perfection is like trying to catch a butterfly, but delivering consistent results is the mark of professionalism—[Section 8.1](#).
- Spend more time listening to what people need than explaining why things are the way they are or can't be fixed.
- Don't mock things you dislike. Technicians who consistently say "Microsquish" instead of Microsoft aren't funny; they're immature.

## A.6 Replacing Services

- Be conscious of the process—[Chapter 33](#).
- Never do an in-place, all-or-nothing, no-way-to-turn-back upgrade. Find a way to roll out the upgrade to a few users at a time.
- Factor in both network dependencies and service dependencies in transition planning.
- Manage your Dynamic Host Configuration Protocol (DHCP) lease times to aid the transition.
- Don't hardcode server names into configurations; instead, hardcode aliases that move with the service—[Section 17.5](#).
- Manage your DNS time-to-live (TTL) values to switch to new servers—[Section 21.8.1](#).

## A.7 Moving a Datacenter

- Schedule windows unless everything is fully redundant and you can move first one half of a redundant pair and then the other half—[Chapter 34](#).
- Make sure the facility is compatible with the mover's truck. Will the truck fit in the driveway at each location? Does each location have a loading dock, or will the mover use trucks with lift gates? Will the mover use pallets, and if so does it expect you to have a pallet jack? Do the movers deliver to the loading dock, or do they bring the items all the way to the desired room?
- Make sure that the new datacenter is properly designed for both current use and future expansion.
- Back up every file system of any machine before it is moved.
- Perform a fire drill on your data backup system—[Section 44.2.4](#).
- Develop test cases before you move, and test, test, test everything after the

move is complete—[Chapter 33](#).

- Label every cable before it is disconnected.
- Establish minimal services—redundant hardware—at a new location with new equipment.
- Test the new environment—networking, power, uninterruptible power supply (UPS), heating, ventilation, air conditioning (HVAC), and so on—before the move begins.
- Identify a small group of customers to test business operations with the newly moved minimal services, then test sample scenarios before moving everything else.
- Run cooling for 48–72 hours, and then replace all filters before occupying the space.
- Perform a dress rehearsal—[Section 33.6.4](#).

## **A.8 Moving to or Opening a New Building**

- Use a digital or paper organizer—[Section 50.2](#).
- Pre-wire the offices with network jacks during, not after, construction.
- Communicate to the powers that be that WAN and ISP connections will take months to order and must be done soon.
- Order WAN and Internet service provider (ISP) network connections two to three months in advance.
- Four weeks or more in advance, get access to the new space to build the infrastructure.
- Use radios or walkie-talkies for communicating inside the building.
- Work with a moving company that can help plan the move.
- Designate one person to keep and maintain a master list of everyone who is moving and his or her old and new office numbers, cubicle designations, or other locations.
- Pick a day on which to freeze the master list. Give copies of the frozen list to the moving company, use the list for printing labels, and so on. If someone's location is to be changed after this date, don't try to chase down and update all the list copies that have been distributed. Move the person as the master list dictates, and schedule a second move for that person after the main move.
- Give each person a sheet of 12 labels preprinted with his or her name and

new location for labeling boxes, bags, and personal computer (PC).

- Give each person a plastic bag big enough for all the PC cables. Technical people can de-cable and reconnect their PCs on arrival; technicians can do so for nontechnical people.
- Always order more boxes than you think you'll need.
- Don't use cardboard boxes; instead, use plastic crates that can be reused.

## **A.9 Handling a High Rate of Office Moves**

- Work with facilities to allocate only one move day each week. Develop a routine around this schedule.
- Establish a procedure and a form that will get you all the information you need about each person's equipment, number of network and telephone connections, and special needs. Have SAs check out nonstandard equipment in advance and make notes.
- Connect and test network connections ahead of time.
- Have customers power down their machines before the move and put all cables, mice, keyboards, and other bits that might get lost into a marked box.
- Brainstorm all the ways that some of the work can be done by the people moving. Be careful to assess their skill level; maybe certain people shouldn't do anything themselves.
- Have a moving company move the equipment, and have a designated SA move team do the unpacking, reconnecting, and testing. Take care in selecting the moving company.
- Train the helpdesk to check with customers who report problems to see whether they have just moved and didn't have the problem before the move; then pass those requests to the move team rather than the usual escalation path.

## **A.10 Dealing with Mergers and Acquisitions**

- If mergers and acquisitions will be frequent, make arrangements to get information as early as possible, even if this means that designated people will have information that prevents them from being able to trade stock for certain windows of time.
- If the merger requires instant connectivity to the new business unit, set expectations that this will not be possible without some prior warning (see



the previous item). If connection is forbidden while the papers are being signed, you have some breathing room—but act quickly!

- If you are the chief executive officer (CEO), involve your chief information officer (CIO) before the merger is even announced.
- If you are an SA, try to find out who at the other company has the authority to make the big decisions.
- Establish clear, final decision processes.
- Have one designated go-to lead per company.
- Start a dialogue with the SAs at the other company. Understand their support structure, service levels, network architecture, security model, and policies. Determine what the new support model will be.
- Have at least one initial face-to-face meeting with the SAs at the other company. It's easier to get angry at someone you haven't met.
- Move on to technical details. Are there namespace conflicts? If so, determine how you will resolve them—[Chapter 39](#).
- Adopt the best processes of the two companies; don't blindly select the processes of the bigger company.
- Be sensitive to cultural differences between the two groups. Diverse opinions can be a good thing if people can learn to respect one another—[Sections 52.8](#) and [53.5](#).
- Make sure that both SA teams have a high-level overview diagram of both networks, as well as a detailed map of each site's local area network (LAN)—[Chapter 24](#).
- Determine what the new network architecture should look like—[Chapter 23](#). How will the two networks be connected? Are some remote offices likely to merge? What does the new security model or security perimeter look like?
- Ask senior management about corporate-identity issues, such as account names, email address format, and domain name. Do the corporate identities need to merge or stay separate? Which implications does this have for the email infrastructure and Internet-facing services?
- Learn whether any customers or business partners of either company will be sensitive to the merger and/or want their intellectual property protected from the other company.
- Compare the security policies, looking, in particular, for differences in

privacy policy, security policy, and means to interconnect with business partners.

- Check the router tables of both companies, and verify that the Internet Protocol (IP) address space in use doesn't overlap. (This is particularly a problem if you both use RFC 1918 address space.)
- Consider putting a firewall between the two companies until both have compatible security policies.

## **A.11 Coping with Frequent Machine Crashes**

- Establish a temporary workaround, and communicate to customers that it is temporary.
- Find the real cause—[Chapter 29](#).
- Fix the real cause, not the symptoms—[Chapter 30](#).
- If the root cause is hardware, buy better hardware—[Chapter 13](#).
- If the root cause is environmental, provide a better physical environment for your hardware.
- Replace the system—[Chapter 33](#).
- Give your SAs better training on diagnostic tools—[Chapter 29](#).
- Get production systems back into production quickly. Don't play diagnostic games on production systems. That's what labs and pre-announced maintenance windows—usually weekends or late nights—are for.

## **A.12 Experiencing a Major Outage or Work Stoppage**

- Consider modeling your outage response on the Incident Command System (ICS). This ad hoc emergency response system has been refined over many years by public safety departments to create a flexible response to adverse situations. Defining escalation procedures *before* an issue arises is the best strategy.
- Notify customers that you are aware of the problem on the communication channels they would use to contact you: intranet helpdesk “outages” section, outgoing message for SA phone, and so on.
- Form a “tiger team” of SAs, management, and key stakeholders; have a brief 15-to 30-minute meeting to establish the specific goals of a solution, such as getting developers working again, restoring customer access to the support site, and so on. Make sure that you are working toward a goal, not

simply replicating functionality whose value is nonspecific.

- Establish the costs of a workaround or fallback position versus downtime owing to the problem, and let the businesspeople and stakeholders determine how much time is worth spending on attempting a fix. If information is insufficient to estimate this, do not end the meeting without setting the time for the next attempt.
- Spend no more than an hour gathering information. Then hold a team meeting to present management and key stakeholders with options.
- Have the team do hourly updates of the passive notification message with status.
- If the team identifies several potential fixes or workarounds, specify an order in which fixes are to be applied, and get assistance from stakeholders on verifying that each procedure did or did not work. Document this, even in brief, to prevent duplication of effort if you are still working on the issue hours or days from now.
- Implement fix or workaround attempts in small blocks of two or three, taking no more than an hour in total to implement them. Collect error messages or log data that may be relevant, and report on them in the next meeting.
- Don't allow a team member—even a highly skilled one—to try to pull a rabbit out of a hat. Since you can't predict the length of the outage, you must apply a strict process to keep everyone in the loop.
- Appoint someone not directly involved in resolving the problem to be the team “parent.” This person will ensure that meals are brought in, notes taken, and people gently but firmly disengaged from the problem if they become too tired or upset to work—[Section B.3.7](#).

## **A.13 Creating a Toolkit for Each SA on a Team**

- Include a set of screwdrivers in all the sizes computers use.
- Include a portable label printer—[Section 26.7](#).
- Provide a lab with spare PCs, network equipment, and servers for experimenting with new configurations—[Section 21.8.1](#).
- Include a laptop with network diagnostic tools, such as a network sniffer, dig, a DHCP client in verbose mode, encrypted TELNET/SSH client, TFTP server, and so on, as well as both wired and wireless Ethernet.
- Provide terminal emulator software and a USB-to-serial cable. The laptop

can be an emergency serial console if the console server dies, the datacenter console breaks, or a rogue server outside the datacenter needs console access.

- Provide a to-do list management system (smartphone app or paper organizer)—[Section 50.2](#).
- Include a cable tester.
- Include a pair of splicing scissors.
- Provide access to patch cables of various lengths. Include one or two 100-foot (30-meter) cables. These come in handy in the strangest emergencies.
- Provide a mobile phone with a digital camera. Sending a snapshot to technical support can be useful for deciphering strange console messages, identifying model numbers, and proving damage.
- Provide a portable (USB)/firewire hard drive.
- Provide radios or walkie-talkies for communicating inside the building.
- Provide a datacenter cabinet or annex stocked with tools and spare parts—[Section 26.7](#).
- Provide high-speed connectivity to all team members' homes and the necessary tools for telecommuting.
- Set up a library of the standard reference books for the technologies the team members are involved in.
- Provide membership in professional societies such as USENIX and ACM—[Section 52.5](#).
- Give out printed, framed copies of the System Administrators' Code of Ethics—[Section 47.2](#).
- Keep a variety of headache medicines. It's really difficult to solve big problems when you have a headache.
- Keep shelf-stable emergency-only snacky bits.
- Include a copy of this book!

## **A.14 Ensuring the Return of Tools**

- Make it easier to return tools: Affix each with a label that reads, "Return to [your name here] when done."
- When someone borrows something, open a helpdesk ticket that is closed only when the item is returned.

- Accept that most tools won't be returned. Why stress out about things you can't control?
- Keep a stash of PC screwdriver kits. When asked to lend a single screw driver, smile and reply, "No, but you can have this kit as a gift." Don't accept it back.
- Don't let a software person have a screwdriver. Politely find out what the person is trying to do, and do it yourself. This is often faster than fixing the other person's mistakes.
- If you are a software person, use a screwdriver only with adult supervision.
- Keep a few inexpensive eyeglass repair kits in your spares area.

### **A.15 Asked Why Documentation is Necessary**

- Good documentation describes the *why* and the *how*.
- When you do things right and they "just work," even you will have forgotten the details when they break or need upgrading—[Chapter 31](#).
- You get to go on vacation—[Section 52.7](#).
- You get to move on to more interesting projects rather than being stuck doing the same stuff because you are the only one who knows how it works—[Section 38.6](#).
- You will get a reputation as being a real asset to the company. This leads to raises, bonuses, and promotions, or at least fame and fortune.
- You will save yourself a mad scramble to gather information when investors or auditors demand it on short notice.

### **A.16 Policies Are Not Documented**

- Document policies because other people can't read your mind—[Section B.1.17](#).
- Communicate expectations for your own team, not just for your customers.
- Document policies to avoid being unethical by enforcing a policy that isn't communicated to the people whom it governs—[Section 47.7](#).
- Document policies to offer the organization a chance to change its ways when policy and practice do not match.

### **A.17 Identifying the Fundamental Problems in the Environment**

- Survey the management chain that funds you—[Chapter 48](#).
- Survey two or three customers who use your services—[Section 49.2.2](#).
- Survey all customers.
- Identify which kinds of problems consume the largest portion of your time—[Section 49.1.3](#).
- Ask the helpdesk employees which problems they see the most—[Sections 27.8](#) and [48.4](#).
- Ask the people configuring the devices in the field which problems they see the most and what customers complain about the most.
- Determine whether your architecture is simple enough to draw by hand on a whiteboard; if its not, maybe it's too complicated to manage.

## **A.18 Getting More Money for Projects**

- Establish the need in the minds of your managers.
- Find out what management wants, and communicate how the projects you need money for will serve that goal.
- Become part of the budget process.
- Do more with less: Make sure that your staff has good time-management skills—[Chapter 50](#).
- Manage your boss better—[Section 52.9](#).
- Learn how your management communicates with you, and communicate in a compatible way.
- Don't overwork or manage by crisis. Show management the "real cost" of policies and decisions.

## **A.19 Getting Projects Done**

- Often projects don't get done because the SAs are required to put out new fires while trying to do projects. Solve this problem first—[Chapter 1](#).
- Adopt the minimum viable product (MVP) strategy—[Chapter 20](#).
- Get a management sponsor. Is the project something that the business needs, or is it something that the SAs want to implement on their own? If the former, use the sponsor to gather resources and deflect conflicting demands. If a project isn't tied to true business needs, it is doubtful whether it should succeed.
- Make sure that the SAs have the resources to succeed. (Don't guess; ask

them!)

- Hold your staff accountable for meeting milestones and deadlines.
- Communicate priorities to the SAs; move resources to high-impact projects, not the easy ones.
- Make sure that the people involved have good time-management skills—[Chapter 50](#).
- Designate project time when some staff will work on nothing but projects, and the remaining staff will shield them from interruptions—[Section 49.1.3](#).
- Reduce the number of projects.
- Don't spend time on the projects that don't matter.
- Set priorities. Focus on the most important issues. Leave the others alone.
- Use an external consultant with direct experience in the area to achieve the highest-impact projects—[Section 48.8](#).
- Hire junior or clerical staff to take on mundane tasks, such as PC desktop support, daily backups, and so on, so that SAs have more time to achieve the highest-impact projects.
- Hire short-term contract programmers to write code to spec.

## A.20 Keeping Customers Happy

- Make sure that you make a good impression on new customers—[Section 49.1.1](#).
- Make sure that you *communicate more* with existing customers—[Chapter 49](#), particularly [Section 49.2.4](#).
- Go to lunch with customers and listen to them—[Section 49.2.7](#).
- Create a system status web page—[Section 49.2.1](#).
- Create a local enterprise portal for your site—[Section 49.2.1](#).
- Terminate the worst performers, especially if their mistakes create more work for others—[Chapter 54](#).
- Determine which customer or customer group generates the most complaints or tickets. Work with them or their manager to solve issues that would have the biggest impact.

## A.21 Keeping Management Happy

- Meet with the managers in person to listen to the complaints; *don't* try to do it via email.
- Find out your manager's priorities, and then adopt them as your own—[Section 52.9](#).
- Turn managers' priorities into measurable metrics. Meet with them monthly to review the current measurements.
- Be sure that you know how management communicates with you, and communicate in a compatible way.
- Make sure that the people in specialized roles understand their roles—Appendix B.

## A.22 Keeping SAs Happy

- Make sure that their direct manager knows how to manage them well.
- Make sure that executive management supports the management of SAs.
- Make sure that the SAs are taking care of themselves—[Chapter 52](#).
- Make sure that the SAs are in roles that they want and understand—Appendix B.
- If SAs are overloaded, make sure that they manage their time well—[Chapter 50](#); alternatively, hire more people and divide the work—[Chapter 53](#).
- Fire any SAs who are fomenting discontent—[Chapter 54](#).
- Make sure that all new hires have positive dispositions—[Section 27.2](#).

## A.23 Keeping Systems from Being Too Slow

- Define *slow*.
- Use your monitoring systems to establish where the bottlenecks are—[Chapter 38](#).
- Look at performance-tuning information that is specific to each architecture so that you know what to monitor and how to do it.
- Recommend a solution based on your findings.
- Know what the real problem is before you try to fix it—[Chapter 29](#).
- Make sure that you understand the difference between latency and bandwidth—[Section 18.2.2](#).

## A.24 Coping with a Big Influx of Computers



- Make sure that you understand the *economic difference* between *desktop* and *server* hardware. Educate your boss and chief financial officer (CFO) about the difference, or they will balk at high-priced servers—[Section 15.1](#).
- Make sure that you understand the *physical differences* between desktop and server hardware—[Section 14.1.1](#).
- Establish a small number of standard hardware configurations, and purchase them in bulk—[Chapter 12](#).
- Make sure that you have automated host installation, configuration, and updates—[Chapters 7](#) and [8](#).
- Check power, space, and HVAC (heating, ventilation, and air conditioning) capacity for your datacenter.
- Ensure that even small computer rooms or closets have a cooling unit.
- If new machines are for new employees, see [Section A.25](#).

## **A.25 Coping with an Influx of New Users**

- Make sure that the hiring process includes ensuring that new computers and accounts are set up before the new hires arrive—[Section 49.1.1](#).
- Have a stockpile of standard desktops preconfigured and ready to deploy.
- Have automated host installation, configuration, and updates—[Chapters 7](#) and [8](#).
- Have proper new-user documentation and adequate staff to do orientation—[Section 49.1.1](#).
- Make sure that every computer has at least one simple game and a CD/DVD player. It makes new computer users feel good about their machines.
- Ensure that the building can withstand the increase in power utilization.
- If dozens of people are starting each week, encourage the human resources department to have them all start on a particular day of the week, such as Mondays, so that all tasks related to IT can be done in batches, and therefore an assembly-line approach can be used.

## **A.26 Coping with an Influx of New SAs**

- Assign mentors to junior SAs.
- Have a “New SA Checklist” of systems to gain access to, mailing lists to

be added to, recurring meetings to be invited to, documents to read, and systems to learn. New hires should update the checklist as they find anything missing.

- Have an orientation for each SA level to make sure the new hires understand the key processes and policies; make sure that it is clear to whom they should go for help.
- Have documentation, especially a wiki—[Chapter 31](#).
- Purchase proper reference books, both technical and nontechnical, including ones on time management, communication, and people skills—[Chapter 52](#).
- Bulk-order the items listed in [Section A.13](#).

### **A.27 Handling a High SA Team Attrition Rate**

- When an SA leaves, completely lock that person out of all systems—[Chapter 54](#).
- Be sure that the human resources department performs exit interviews.
- Make the group aware that you are willing to listen to complaints in private.
- Have an “upward feedback session” at which your staff reviews your performance. Consider finding a way to do it anonymously.
- Determine what you, as a manager, might be doing wrong.
- Do not have an all-hands meeting to complain that morale is low. Nothing reduces morale more than having to sit through a long, useless meeting about morale. Instead, take action.
- Do things that increase morale: Have the team design and produce a T-shirt together—a few dozen dollars spent on T-shirts can induce a morale improvement that thousands of dollars in raises can’t.
- Encourage everyone in the group to read [Chapter 52](#), but don’t force it on them... that simply reduces morale.
- If everyone is leaving because of one bad apple, get rid of that person. Really.

### **A.28 Handling a High User-Base Attrition Rate**

- Make sure that management signals the SA team to disable accounts, remote access, and so on, in a timely manner—[Chapter 54](#).

- Make sure that exiting employees return all company-owned equipment and software they have at home.
- Take measures against theft of physical property as people leave.
- Take measures to guard against theft of intellectual property, including possibly restricting remote access.

## **A.29 Being New to a Group**

- Before you comment, ask questions to make sure that you understand a given situation.
- Meet all your co-workers one on one.
- Meet with customers both informally and formally—[Chapter 49](#).
- Be sure to make a good first impression, especially with customers—[Section 49.1.1](#).
- Give credence to your co-workers when they tell you what the problems in the group are. Don't reject them out of hand.
- Don't blindly believe your co-workers when they tell you what the problems in the group are. Verify them first.

## **A.30 Being the New Manager of a Group**

- Don't let your predecessor's incompetence become your first big mistake. That new system or conversion that's about to go live? Stop it until you've verified that it meets your high expectations.
- Meet all your employees one on one. Ask them what they do, which role they would prefer, and where they see themselves in a year. Ask them how they feel you can work with them best. The purpose of this meeting is to listen to them, not to talk.
- Establish weekly group staff meetings.
- Meet your manager and your peers one on one to get their views.
- From day one, show the team members that you have faith in them all.
- Meet with customers informally and formally—[Chapter 49](#).
- Ask everyone to tell you what the problems facing the group are, listen carefully to everyone, and then look at the evidence and make up your own mind.
- Before you comment, ask questions to make sure that you understand the situation.

- If you've been hired to reform an underperforming group, postpone major high-risk projects, such as replacing a global email system, until you've reformed/replaced the team.

### **A.31 Looking for a New Job**

- Determine why you are looking for a new job; understand your motivation.
- Determine which role you want to play in the new group—Appendix B.
- Determine which kind of organization you enjoy working in the most—[Section 48.10](#).
- Meet as many of your potential future co-workers as possible to find out what the group is like—[Chapter 53](#).
- Never accept the first offer right off the bat. The first offer is just a proposal. Negotiate! But remember that there usually isn't a third offer—[Section 51.5.4](#).
- Negotiate in writing the things that are important to you: conferences, training, vacation.
- Don't work for a company that doesn't let you interview with your future boss.
- If someone says, "You don't need to have a lawyer review this contract" and isn't joking, you should have a lawyer review that contract. We're not joking.

### **A.32 Needing to Hire Many New SAs Quickly**

- Review the advice in [Chapter 53](#).
- Use as many recruiting methods as possible: Organize fun events at conferences, use online boards, sponsor local user groups, hire famous people to speak at your company and invite the public, get referrals from SAs and customers—[Chapter 53](#).
- Make sure that you have a good recruiter and human resources contact who knows what a good SA is—[Section 53.3](#).
- Determine how many SAs of which level and with which skills you need. Use the USENIX LISA skill level classifications—[Section 53.2](#).
- *Move quickly* when you find a good candidate.
- After you've hired one person, refine the other job descriptions to fill in

the remaining gaps—[Section 48.4](#).

### **A.33 Increasing Total System Reliability**

- Figure out what your target is and how far you are from it.
- Set up monitoring to pinpoint uptime problems—[Chapter 38](#).
- Deploy end-to-end monitoring for key applications—[Section 38.8](#).
- Reduce dependencies. Nothing in the datacenter should rely on anything outside the datacenter—[Sections 17.4](#) and [34.9.1](#).

### **A.34 Decreasing Costs**

- Let people know the cost of the services they use.
- Provide a dashboard that tells users which software is licensed to them so they can delicense unused software.
- Provide management with per-user or per-incident costs for various services.
- Centralize services for efficiency—[Chapter 35](#).
- Review your maintenance contracts. Are you still paying for machines that are no longer critical servers? Are you paying high maintenance costs for old equipment that would be cheaper to replace?—[Section 14.5](#).
- Determine whether you can reduce the support burden through standards and/or automation—[Chapters 7](#) and [8](#).
- Reduce support overhead through applications training for customers or better documentation.
- Distribute costs more directly to the groups that incur them, such as maintenance charges, remote access charges, and special hardware—[Section 48.2](#).
- Determine whether people are willing to pay for the services you provide. If people aren't willing to pay for the service, it isn't important, so decommission it.
- Take control of the ordering process and inventory for incidental equipment such as replacement mice, mini-hubs, and similar. Do not let customers simply take what they need or direct your staff to order it.

### **A.35 Adding Features**

- Interview customers to understand their needs and to prioritize features.

- Know the requirements—[Chapter 16](#).
- Make sure that you maintain at least existing service and availability levels.
- If you are altering an existing service, have a backout plan.
- Look into building an entirely new system and cutting over rather than altering the running one.
- If it's a really big infrastructure change, use a maintenance window—[Chapter 34](#).
- Decentralize so that local features can be catered to.
- Test! Test! Test!
- Document! Document! Document!

### **A.36 It Hurts When I Do “This”**

- Don't do “that.”
- If you have to do “that,” try to automate it.

### **A.37 Building Customer Confidence**

- Improve follow-through—[Section 50.2](#).
- Focus on projects that matter to the customers and will have the biggest impact.
- Until you have enough time to complete the projects you need to, discard projects that you haven't been able to achieve.
- Communicate more—[Chapter 49](#).
- Go to lunch with customers and listen—[Section 49.2.7](#)
- Create a good first impression on the people entering your organization—[Section 49.1.1](#).

### **A.38 Building the Team's Self-Confidence**

- Start with a few simple, achievable projects; only then should you involve the team in more difficult projects.
- Ask team members which training they feel they need, and provide it.
- Coach the team. Get coaching on how to coach!

### **A.39 Improving the Team's Follow-Through**

- Find out why team members are not following through.
- Make sure that your trouble-ticket system assists SAs in tracking customer requests and that it isn't simply for tracking short-term requests. Be sure that the system isn't so cumbersome that people avoid using it—[Section 27.10](#).
- Encourage team members to have a single place to list all their requests—[Section 50.2](#).
- Discourage team members from trying to keep to-do lists in their heads—[Section 50.2](#).
- Pay for a team's time-management tools such as smartphone apps, stationary, training, and books—[Section 50.2](#).

#### **A.40 Handling an Unethical or Worrisome Request**

- See [Section 47.8](#).
- Log all requests, events, and actions.
- Get the request in writing or email. Try a soft approach, such as “Hey, could you email me exactly what you want, and I'll look at it after lunch?” Someone who knows that the request is unethical will resist leaving a trail.
- Check for a written policy about the situation—[Chapter 47](#).
- If there is no written policy, absolutely get the request in writing.
- Consult with your manager *before* doing anything.
- If you have any questions about the request, escalate it to appropriate management.

#### **A.41 My Dishwasher Leaves Spots on My Glasses**

- Spots are usually the result of not using hot enough water and are unlikely to be fixed by finding a special soap or even using a special cycle on the machine.
- Check for problems with the hot water going to your dishwasher.
- Have the temperature of your hot water adjusted.
- Before starting the dishwasher, run the water in the adjacent sink until it's hot.

#### **A.42 Protecting Your Job**

- Look at your most recent performance review and improve in the areas

that “need improvement”—whether or not you think they do.

- Get more training in areas in which your performance review has indicated you need improvement.
- If your last performance review has indicated you need improvement in certain areas, volunteer to do those tasks more to get more practice (or change your job responsibilities so you don’t have anything to do with them).
- Be the best SA in the group: Have positive visibility—[Chapter 49](#).
- Document everything—policies and technical and configuration information and procedures—[Chapter 31](#).
- Have good follow-through.
- Help everyone as much as possible.
- Be a good mentor.
- Use your time effectively—[Chapter 50](#).
- Automate as much as you can—[Chapter 1](#); [Sections 30.4](#), [44.9](#), and [49.1.4](#).
- Always keep the customers’ needs in mind—[Sections 49.1.3](#). and [52.9](#).
- Don’t speak ill of co-workers. It just makes you look bad. Silence is golden. A closed mouth gathers no feet.

## **A.43 Getting More Training**

- Go to training conferences like USENIX LISA.
- Attend vendor training to gain specific knowledge and to get the inside story on products.
- Find a mentor.
- Attend local SA group meetings
- Present at local SA group meetings. You learn a lot by teaching.
- Find the online forums or communities for items you need training on, read the archives, and participate in the forums.

## **A.44 Setting Your Priorities**

- Depending on which stage you are in, ensure certain infrastructure issues are happening:
  - Basic services, such as email, printing, remote access, and security, need to be there from the outset—[Chapters 39](#) to [44](#).



- Automation of common tasks, such as machine installations, configuration, maintenance, and account creation and deletion, should happen early; so should basic policies—[Chapters 8 to 12](#).
- Documentation should be written as things are implemented, or it will never happen—[Chapter 31](#).
- Build a software depot and deployment system—[Chapter 45](#).
- Monitor everything so that, in due course, you can think about improvements and scaling, which are issues for a more mature site—[Chapter 38](#).
- Think about setting up a helpdesk—[Chapter 27](#).
- Get more in touch with your customers to find out what their priorities are.
- Improve your trouble-ticket system—[Chapter 27](#).
- Review the top 10 percent of the ticket generators—[Section 27.11](#).
- Adopt better revision control of configuration files—[Chapter 32](#), particularly [Section 32.11.3](#).

## **A.45 Getting All the Work Done**

- Climb out of the hole—[Chapter 1](#).
- Improve your time management through classes and books—[Section 50.9](#).
- Use a console server so that you aren't spending so much time running back and forth to the machine room—[Sections 26.5](#), [14.3](#), and [34.9.2](#).
- Batch up similar requests; group all tasks that require being in a certain part of the building or the same tools.
- Start each day with project work, not by reading email.
- Make informal arrangements with your co-workers to trade being available. Then find an empty conference room and get uninterrupted work done for a couple of hours.

## **A.46 Reducing Stress**

- Take those vacations! (A three-day weekend is not a vacation.)
- Take a vacation long enough to learn what hasn't been documented well. It's better to find those issues when you're returning in a few days than when you're (heaven forbid) hit by a bus.
- Take walks; get out of the area for a while.

- Don't eat lunch at your desk.
- Don't forget to have a life outside of work.
- Get weekly or monthly massages.
- Sign up for yoga or meditation classes.
- Exercise. It helps to clear the mind.

## **A.47 Working as Part of a Team**

- Document processes well enough that the rest of the team can do them.
- Be on time for meetings and appointments.
- Communicate in civil terms.
- Be honest.
- Keep the team informed about what they're doing; stay up-to-date on what the rest of the team is doing.
- Take vacations and manage stress.
- Communicate using the channels the team has designated.
- Meet commitments. Under-promise and over-perform.
- Check your work.
- Be free with your knowledge. Be willing to teach others.
- Praise in public, criticize in private.
- Do not withhold knowledge in a passive-aggressive attempt to get your way.

## **A.48 Managing SAs**

- Clearly communicate priorities.
- Get enough budget to meet goals.
- Give feedback that is timely and specific.
- Give the team permission to speak freely in private in exchange for using decorum in public—[Section 49.1.2](#).

## **A.49 Working as an SA**

- Do the best job you can.
- Treat customers well—[Chapter 49](#).
- Get things done on time, under budget.

- Learn from mistakes.
- Ask for help—[Section 52.3](#).
- Under-promise and over-perform.
- Set an honest status for milestones as projects progress.
- Participate in budget planning.
- Have high ethical standards—[Section 47.2](#).
- Take at least one long vacation per year—[Section 52.4](#).
- Keep on top of technology changes—[Section 52.5](#).

## **A.50 Reporting Up the Management Chain**

- Provide access to monitoring and reports so that the boss can update himself or herself on status at will.
- Provide budget information in a timely manner.
- Provide pessimistic time estimates for requested projects. Under-promise and over-deliver.
- Provide honest status information on milestones as projects progress.
- Provide a reasonable amount of stability.

## **Appendix B. The Many Roles of a System Administrator**

This appendix is heavy on philosophy. If that turns you off, you can skip it, but we believe that it will help you think about your place in the universe or at least your role within your company, organization, or SA team. Examining your own role within an organization helps you focus, which helps you do a better job. It can give you a long-term perspective on your career, which can help you make the big career decisions necessary for having a happy and successful life.

This appendix can also give your organization a framework for thinking about which roles it wants you to play. Each of the roles discussed in this appendix in some way affects your organization. This is by no means a complete list, but it is a very good starting point. You should use this list to consider which roles are missing in your organization and perhaps to start on a quest to fill them.

It is interesting to think about which and how many of these roles you are asked to play as your career moves forward. The list can help you plan your career. Some entry-level SAs are asked to play single roles and grow into more roles as they gain experience. Sometimes, SAs start out flooded with many roles and specialize as time goes on.

A small site may require its single SA to take on many roles. As the organization grows, certain roles can be transferred to newly hired SAs. Sometimes, you may discover that you don't enjoy a particular role and look to avoid it when you change jobs. Thinking about these roles may also help guide your career with respect to which kinds of companies you decide to work for: Small companies tend to require people to fill multiple roles, larger companies tend to require people to specialize, and mega-corporations have people so specialized that it can seem bizarre to outsiders. Technology companies respect and reward those who play the role of pushing for new technology, whereas other companies often discourage too much change.

### **B.1 Common Positive Roles**

Some roles within a company are more critical than others; some are good and some are bad. Here we list many common roles and explain how they provide value to the company, how people derive satisfaction from being in the role, and what customers tend to expect from them.

### **B.1.1 The Installer**

Some people view an SA as the person who installs “stuff.” This is the role that customers see most often, so it is most often associated with the career of system administration. The customer rarely sees the other, possibly more critical positions, such as the people who design the infrastructure.

The value this role provides to the company is the ability to follow through and see that the job gets done. SAs working as installers are often the final and most critical link in the deployment chain.

When installation is being done on a large scale, the item that is being installed is usually preconfigured at some central location. Installers are trained on the specific situations they are expected to see and have a second-tier resource to call on if they come across an unexpected situation. In that case, the kind of person who makes a good installer is one who enjoys meeting and helping the customers and gets satisfaction from doing the same task well many times over. In smaller deployments, the installer is often expected to be a higher-skilled person because more unexpected situations will be encountered.

When you are the installer, it is important to be friendly and polite. The installer is often the public face of the IT organization; people will assume that the entire organization acts the same way that you do.

### **B.1.2 The Repair Person**

Things break. Some people view all SAs as the repair people. Just as people call a dishwasher repair person when their dishwasher breaks, so they call a computer repair person when their computer breaks. SAs also repair bigger and sometimes more nebulous things, such as “the Internet” and “the database.” Whether the real problem is simply a broken cable or a much larger problem is of little interest to the customer.

The value this role provides the company is the ability to bring the company back to life when technological problems stall a business. Repair people receive satisfaction from knowing they’ve helped one person or the entire company. They enjoy the challenge of a good puzzle or mystery.

When you are the repair person, customers want to know that you are concerned about their problems. They want to feel as though their problems are the most important problems in the world.

### **B.1.3 The Maintainer**

The maintainer is the person who keeps previously built systems going.

Maintainers are very good at following the instructions presented to them, either in a written manual or through training. They do not seek to improve the system; they are willing to maintain it as it is.

The value this role provides the company is in the ability to bring stability to the environment. These people are not going to break things trying to improve them or replace them, nor do they spend all day reading magazines about new things to install. Once companies spend money to install something, they need it to be stable long enough to pay for itself before it is replaced with something newer.

Maintainers receive satisfaction from knowing that their work is part of the big picture that keeps the organization working. They tend to be glad that they aren't the people who have to figure out how to design and install the next generation of systems and may even have disdain for those who wish to replace their stable system with something new.

When you are the maintainer, you are expected to do two opposing things: maintain the stability of the customer's world, and be flexible when they seek customizations.

#### **B.1.4 The Problem Preventer**

A role that is invisible to most customers is the problem preventer, who looks for problems and fixes them before they become visible. Problem preventers do the behind-the-scenes planning and preventive maintenance that keeps problems from occurring at all. A good problem preventer collects metrics to find trends but also has an ear to the ground to know which future problems may arise.

The value that problem preventers provide to a company is that they avert problems, which is less costly than fixing problems when they happen.

Problem preventers receive satisfaction from knowing that their work prevented problems that no one even knows could have happened. Their joy is private. They enjoy thinking in the longer term rather than getting immediate satisfaction from solving an emergency.

Typical customers do not know that this person exists, but their management does. The managers expect this person to have the same priorities that they do.

#### **B.1.5 The Hero**

The SA can be the hero who saves the day. Like the fire fighter who pulls people out of a burning building, the hero receives adulation and praise. The network was down, but now it is up. The demo wasn't going to be ready, but the SA

worked all weekend to bring the network to that part of the building. Heroes get satisfaction out of their jobs from the praise they receive after the fact.

The value to the company of heroes is huge: Management always rewards a hero. Ironically, problem preventers often must struggle to get the same positive visibility, though their contributions may be as or more valuable than the hero's feat.

Heroes receive satisfaction from knowing that they hold the keys to some knowledge that the company could not live without. The hero role is not one that promotes a healthy non-work life. Heroes give up nights, weekends, and vacations, often with no notice. A personal life takes second priority. Eventually, heroes burn out and become martyrs, unless management finds some way to help them manage their stress.

Customers expect the hero to be anywhere at any time. Customers would prefer to deal only with the hero, because this superstar has become someone on whom they can rely. However, customers need to learn that if they get everything they want, the hero will burn out. New heroes take a while to find.

### **B.1.6 The Go-To Person**

The go-to person has gained the reputation of being the expert for a particular technology or service. Go-to people are a little like heroes, but are more coordinated and more infrastructure related. This is the person on whom everyone relies when a question arises or a problem requires deep knowledge. Management knows that the go-to person will get to the bottom of the problem, work out the underlying problems, and fix them. It could be an infrastructure issue, a process issue, or the need to understand the context of why something was designed a particular way.

The value of having a go-to person around is to get things done when others can't.

Like heroes, go-to people can burn out if overused. Nevertheless, when they do the jobs, they get the satisfaction of knowing that their solutions will become part of the standard procedures going forward.

Customers expect go-to people to follow through when they agree to solve problems and to be able to give accurate time estimates, or at least periodic status updates until time estimates can be given.

### **B.1.7 The Infrastructure Builder**

A corporate network depends on a lot of infrastructure: DNS, directories,

databases, scripts, switches, and so on. None of this is seen by the typical customer, except when an outage is explained after the fact mysteriously—for example, “It was a problem with the DNS server.”

The larger the company, the more valuable infrastructure builders become. A good infrastructure is like a solid foundation on which a house can be built. You can build a house on a shaky foundation and adjust for it with more complicated and costly house designs, but in the long run it is cheaper to have started with a solid foundation. A tiny company has almost no infrastructure. Larger companies get benefits from amortizing the cost of a quality infrastructure over larger and larger customer bases. When small companies grow to become large companies, often what makes this expansion go smoothly is having had the foresight to employ SAs who “think big” about infrastructure.

Infrastructure builders get satisfaction from doing long-term planning, taking existing systems and improving them, scaling large systems into humongous systems, and overhauling legacy systems and replacing them with newer systems. Infrastructure builders are proud of their ability to not only build extremely huge systems, but also coordinate elegant ways to transition to them.

When you are the infrastructure builder, you have two groups of customers. The general customer population wants the computer infrastructure to be reliable and wants new infrastructure to be deployed yesterday. The other customers are the SAs whose systems sit on top of the infrastructure you are building. The SAs want documentation and an infrastructure that is reliable and easy for them to understand, and they want it *now*, because when you miss a deadline, it makes their projects late, too.

### **B.1.8 The Policy Writer**

Policies are the backbone of IT. They communicate the wishes of the top corporate officials and dictate how things should be done, tell when they should be done, and explain why they are done. SAs are often asked to write policies on behalf of management. Social problems cannot be solved by technology. Some social problems can be solved only by written policy.

The value to the company of policy writers is that they solve some problems and prevent new ones. Policies are a communication tool. As a company grows, communication becomes more difficult and more important.

Policy writers gain satisfaction from knowing that their knowledge, skills, and personal experiences contributed to a policy that improved an organization. They also enjoy being facilitators who can obtain buy-in from many different



communities.

When you are the policy writer, seek customer input. This should be done at the beginning of the process. It disempowers people to ask for their opinion after the major decisions have been made. Your willingness to listen will be appreciated.

### **B.1.9 The System Clerk**

System clerks have very little power or decision-making responsibilities. These SAs are given instructions to be followed, such as “Create an account for Fred” or “Allocate an IP address.” If the system clerk works as an assistant to a higher-level SA, this can be a beneficial arrangement. In fact, it is an excellent way to start a career. However, when a system clerk reports to a nontechnical managers, that manager may not understand the clerk’s limits and can get frustrated when the clerk is not able to tackle things outside his or her normal duties.

The value that the system clerks provide to the company is that they perform the tasks that would otherwise distract senior SAs from more specialized tasks. They also provide coverage for SAs when they are away. In addition, a system clerk is an excellent candidate to fill a more senior SA position as it opens. The clerk already knows the environment, and the hiring manager knows the clerk’s personality. However, if the environment has no senior SAs, the clerks can often be scapegoats for a bad computing environment, when the real problem is management’s lack of understanding about how to manage technology.

The clerk receives satisfaction from a job well done, from learning new skills, and from looking forward to the excellent growth path ahead.

When you are the clerk, you will be expected to perform requests immediately, whether that is reasonable or not. You may not have the power to reject unreasonable requests. [Chapter 49](#), “[Perception and Visibility](#),” has more information about dealing with this situation.

---

#### **Case Study: Site with Only System Clerks**

A site needs a balance of senior-level SAs and clerks. There once was a site that had only system clerks. Their training included rudimentary Unix skills: perform backups, create accounts, allocate IP addresses and IP subnets, install new software, and add new hosts. The clerks fell victim to the “we can always add one more” syndrome: New allocations were blindly made as requested, with no overall plan for increasing capacity. For example, a new host would be added to a subnet without any network-

capacity planning. This worked for a while, but eventually led to overloaded subnets.

Customers complained of slow networks, but the clerks did not have the network engineering skills to fix the problem. Customers solved this problem themselves by requesting private subnets to gain their own private dedicated local bandwidth. The clerks would happily allocate a new IP subnet, and users would connect it to the rest of the network via a routing-enabled workstation with two NICs. These interconnections were unreliable because hosts route packets slowly, especially when they become overloaded. The more overloaded the main networks became, the more dedicated subnets were created.

Eventually, much of the slowness of the network was caused by the slow interconnections between these private pools of bandwidth. The company's compute servers also suffered from the same lack of capacity planning. The customers installed their own compute servers, even though the performance problems they were trying to work around were most likely related to the slow host-based routing. These new, fast servers overpowered the 10 Mbps network, particularly because they were often an order of magnitude faster than the hosts doing the routing.

By the time the organization hired a senior-level SA, the network was a swamp of unreliable subnets, badly configured compute servers, and antique file servers. The network had 50 subnets for about 500 users. It took nearly 2 years to clean up the mess and modernize the network.

---

### **B.1.10 The Lab Technician**

The lab technician is an SA for highly specialized equipment. For example, in a chemical research firm, the lab technician may be responsible for a small network that connects all the scopes and monitoring devices. At a telecommunications manufacturer, the lab technician may maintain all the equipment in a protocol-interoperability room, which would include one of every version of a product, the competition's products, and a suite of traffic generators. The lab technician is responsible for installing new equipment, integrating systems together for ad hoc projects (often all projects are ad hoc), and interpreting customer specialties to translate their needs into the tasks that must be performed. The lab technician usually manages a small network or group of networks that connect to the main corporate network and depend on the main corporate network for most services; a smart lab technician also makes

friends in the corporate services area to pick their brains for technical knowledge.

The value lab technicians provide is that they enable researchers to focus on higher-level issues such as design and analysis of experiments. Lab technicians also add value through their vast knowledge base of technical information.

Lab technicians derive satisfaction from successfully completing experiments or demos on time. However, if they do not get direct congratulations from the researchers they serve, they may grow resentful. Lab technicians need to remember that their researchers are grateful, whether they express it or not. Researchers will have technicians who stay with them longer if the technicians are included in recognition ceremonies, awards, dinners, and so on.

When you are the lab technician, let customers know that something can be done, not how it will be done. They want their requirements met, though it is your responsibility to draw out of them what those requirements are. Active listening skills can greatly help in this area.

### **B.1.11 The Product Finder**

The product finder reads every technology review magazine, web site, and blog so that when someone asks, “Is there a software package that compresses widgets?” the product finder can recommend not only a list of widget compressors, but also ways to determine which is the most appropriate for the particular application. The product finder also knows where to find such a product or where to look for one.

The value this role provides to a company is that the product finder is able to keep the company on top of what’s new. Managers should not watch this kind of person closely, because they will be appalled to discover that the product finder spends half his or her workday surfing the web and reading magazines.

Managers must weigh that against the time this person saves for everyone else.

Product finders receive satisfaction from having all the right resources. These people can be annoying to others in the group, even those whom they help, because everyone would prefer to have the time to surf the web and keep in touch, but most people (necessarily) have other priorities.

When you are the product finder, provide customers with summaries rather than details. If you provide them with every detail that you’ve learned on the subject in a long rambling story that takes hours to read, they will shy away from you. Be concise.

### **B.1.12 The Solution Designer**

Solution designers play a key role in a company. On hearing of a problem, they soon have a solution that is better than anyone else could have provided. This may solve a small issue, such as installing an e-fax server to make it easier to send faxes, or it may resolve a large issue, such as creating an electronic version of a paper process. Unlike product finders, solution designers are more likely to build something from scratch or to integrate some smaller packages.

The value to the company of solution designers is their ability to remove roadblocks and simplify bureaucratic processes.

The solution designer receives satisfaction from knowing that his or her solutions are used, because usage indicates that people like it.

When you are the solution designer, show customers their aspect of the problem solved, not what you may perceive as the problem or what would save the company money. For example, if expense reports are faxed to headquarters (HQ), you might create a way for the data to be entered electronically so that HQ doesn't have to retype all the data. However, your customers aren't helped by saving time at HQ; they simply want the preparation to be made easier. That would be solved with a better user interface or a system that could download their corporate credit card bill off the service provider's web site. Your customers wouldn't even care if the output was then e-faxed to HQ for manual reentry.

### **B.1.13 The Ad Hoc Solution Finder**

The ad hoc solution finder can, on an emergency basis, create a solution to a seemingly impossible problem. This is the person who magically yet securely gets network connectivity to the moon for your big demo to the moon people. Ad hoc solution finders may know more about the tools than the average person who uses the tools, possibly from dissecting them. Unlike the hero, who usually puts out fires by fixing problems, this person builds solutions.

The value to the company of ad hoc solution finders is their ability to find solutions that work around the fact that the technology is not as flexible as some special situations require or that the corporate network has weaknesses that they have not invested in fixing. The former is a situation that gets better over time. The latter indicates a lack of proper technology management.

The ad hoc solution finder receives satisfaction from saving the day. Like the hero, the ad hoc solution finder can get burned out from being overloaded.

When you are the ad hoc solution finder, be prepared to pull off miracles, and

don't remind the customer that the emergency could have been prevented through better planning by the company, which is rarely their fault.

### **B.1.14 The Unrequested Solution Person**

Some SAs find themselves providing solutions that weren't requested. This can be a good thing and a bad thing. One SA was rewarded for installing a paperless fax system that wasn't requested but soon became a major productivity enhancement. It was based on free software and used the existing modem pool, so the tangible cost was zero. This same SA was once reprimanded for spending too much time on "self-directed projects" and was encouraged to focus on his assigned tasks.

The value to the company of unrequested solution people is that they are usually close to the customers and positioned to see needs that upper management wouldn't see or understand. These SAs may also be more aware of new products than their less technical customers.

Individuals in this role receive satisfaction from discovering that their guesses of what might be useful turn out to be correct.

When you are in this role, you have to guess correctly what will or won't be useful to customers; talking with them regularly at appropriate times is critical. They will be concerned that these new projects don't interfere with your assigned project's deadlines, especially when that would result in them missing their own deadlines. Management will be concerned about the cost of your time and of any tangible costs, especially when an unrequested new service does not get used.

### **B.1.15 The Oncall Expert**

Oncall experts are always available to give advice and have established themselves as knowledgeable in all or most aspects of the system. Sometimes the oncall expert has a narrow focus; other times, he or she is an all-around expert.

The value to the company of oncall experts is that people have someone to call when they need advice, whether they need an exact answer or simply a good starting point for research.

The oncall expert receives satisfaction from helping people and from the ego trip that is inherent to the role. Because technology changes quickly, the oncall expert requires time to maintain knowledge, whether that is time spent reading magazines, networking at conferences, or experimenting with new products.

When you are the oncall expert, you must remember to help people help themselves. If you don't, you will find yourself overcommitted.

### **B.1.16 The Educator**

The educator teaches customers to use the services available. The educator may stop by to fix a problem with a printer but will also stay to teach the customer how to better use the spreadsheet software. This person probably writes the majority of the user documentation.

Educators are valuable to the company because their work results in people working more efficiently with the tools they have. The educator has close interactions with customers and therefore learns which problems people are having. He or she becomes a resource for finding out what the customers need.

The educator receives satisfaction from knowing that his or her documentation is used and appreciated, and from knowing that people work better because of those efforts.

When you are the educator, learn to understand your customers' jobs, how they work, and, most important, what it is in their tools that they find confusing. They want documentation that answers the questions they have, not the questions that the developers think are important.

### **B.1.17 The Policy Enforcer**

The policy enforcer is responsible for saying no when someone wants to do something that is against policy and for shutting down violators. The policy enforcer depends on two tools equally: written policies and management support. Policies must be written and published for all to see. If the policies are not written, enforcement will be inconsistent because policy enforcers will have to make up the rules as they go along, and their peers may enforce different ideas of what is right and wrong. The second tool is management support. The policy has no teeth if management bends the rules every time someone requests an exception. A manager shouldn't sign off on a policy and then continually sign off on requests for exceptions. Often, the policy enforcer has the authority to disconnect a network jack if the violation is creating a global problem and the violator cannot be contacted in a reasonable amount of time. If the management does not support the enforcer's decision, the enforcer can't do his or her job. If management approves a policy but then permits an exception after the enforcer says no, the enforcer loses authority and possibly the motivation to continue being the enforcer.

The value the policy enforcer provides to a company is that company policies are carried out. Lack of follow-through on an important policy defeats the point of having a policy.

The policy enforcer receives satisfaction from knowing that he or she is actively trying to keep the company following the direction set by management.

Customers want to get their jobs done and don't understand why so many roadblocks (policies) are preventing them from doing that. When you are the policy enforcer, rather than saying no, it can be more useful to help customers by understanding what they are trying to achieve and helping them reach that goal and stay within policy. If you do not want to be in this role but feel trapped in it, you might consider assertiveness training or such books as *When I Say No, I Feel Guilty* ([Smith 1975](#) and [2000](#)).

---

### **A Policy with Exceptions**

A site had a security policy that created a lot of extra work for anyone who wanted to abide by it. For a web site to be accessible from outside the firewall, the site had to be replicated on the outside rather than by poking a hole in the firewall to let outsiders access the internal host. This replicated site could not make connections back into the company. If it needed access to an internal service, such as a database, that service also had to be replicated. Making a service completely self-sufficient was very difficult. Therefore, when the policy enforcer rejected a request, the employee would cry to management, and an exception would be granted. Eventually, enough holes were poked in the firewall that the policy didn't mean anything.

The policy enforcer proposed a revision to the policy that simply reflected management's behavior: Holes would be poked if the cost of replication would exceed a certain number of hours of work. Management was in a furor at the proposal because that was not how it wanted security to be done. The policy enforcer pointed out all the exceptions that management had made. Although old exceptions were grandfathered, management became much better at supporting the policy enforcer after the revision. If management wasn't going to support the policy, the policy enforcer shouldn't have to, either.

---

### **B.1.18 The Disaster Worrier**

Someone in the group should be worried about things going wrong. When a

solution is being proposed, this person asks, “What is the failure mode?” Of course, the disaster worrier can’t drive all decisions, or projects will never be completed or will be over budget. This person needs to be balanced by an optimist. However, without someone keeping an eye out for potential disasters, a team can create a house of cards.

The value to the company of the disaster worrier is not felt until there is an emergency. For example, half the system is failing, but the other half keeps working because of controls put in place. General system robustness can be the result of this person.

The disaster worrier receives satisfaction from ensuring safety and stability.

When you are in this role, others around you may get tired of your constant push for belts and suspenders. It is important to pick your battles, rather than have an opinion at every turn. No one likes to hear such laments as “That wouldn’t have failed if people had listened to me” or “Next time, you won’t be so quick to ignore me!” It may be better to share responsibility rather than place blame, and refer to future improvement rather than gloat about your expertise: “In the future, we need to write scripts that handle disk-full situations.” Gentle one-on-one coaching is more effective than public bemoaning.

### **B.1.19 The Careful Planner**

The careful planner takes the time to plan each step of the project in which he or she is involved. The careful planner builds good test plans and is never flustered when things go wrong, because he or she has already figured out what to do.

The value to the company of the careful planner is that he or she completes important tasks reliably and flawlessly.

This person derives satisfaction from completing a task and knowing that it is really finished, and from watching the first customers use it without a hitch. Careful planners take pride in their work.

When you are in this role, others come to rely on your work being flawless. You are often given the tasks that cannot afford to fail. Continue to work as you always did, and don’t let the importance of the tasks weigh you down. Be aware that your meticulous work takes time and that others are always in a hurry and may get agitated watching you work. Make sure that you develop a talent for predicting how long you will need to complete a task. You don’t want to be seen as someone who couldn’t meet a deadline if it walked up and introduced itself.

### **B.1.20 The Capacity Planner**



The capacity planner makes the system scale as it grows. This person notices when things are getting full, running out, or becoming overloaded. *Good* capacity planners pay attention to utilization patterns and are in tune with business changes that may affect them. *Great* capacity planners install systems that do this monitoring automatically and produce graphs that predict when capacity will run out. Vendors can help capacity planners by documenting data that they would find useful, such as how much RAM and disk space are required as a function of the number of users.

The value to the company of capacity planners is that they prevent traffic jams. This is another role that goes unnoticed if the job is done properly. This person also helps the company fix the correct problem the right way. (Too many times, departments try to speed up a server by adding more RAM when the real problem is an overloaded network connection, or vice versa.)

The capacity planner receives satisfaction from knowing that problems are prevented and that people heed warnings, and from finding the real source of problems.

When you are the capacity planner, be focused on the accurate collection of data as the way to best predict the future. It is your job to justify costs. As always, explaining things in the customer's language is critical.

### **B.1.21 The Budget Administrator**

The budget administrator keeps tabs on how much money is left in the budget and helps write the budget for next year. This person knows what the money is meant to be spent on, when it is meant to be spent, and how to make the budget stretch farther.

The value to the company of the budget administrator is to keep SA expenses under control, ensure that the tasks that need doing are funded (within reason) even if they are unexpected, and provide reliable figures so management can perform financial planning for the coming year.

The budget administrator receives satisfaction from staying within budget and still managing to fund extra, important projects that were not budgeted for.

When you are the budget administrator, stay in budget, prepare a good budget plan for the next year, accurately evaluate what the most important projects are, make sure that all the critical tasks have funding, and show how money spent is benefiting the company.

### **B.1.22 The Customer's Advocate**

The customer's advocate can help customers speak up for their needs. The customer's advocate is the translator and lobbyist positioned between customers and their management. The advocate doesn't simply recommend a solution, but also coaches the customer on how to sell the idea to management and stands by during the presentation in case the customer needs help.

The value to the company of the customer's advocate is to help the customers get what they need despite red tape and communication barriers.

The advocate receives satisfaction from knowing that he or she has helped someone. Advocates also know that by interfacing with management, they are able to put their SA teams in a good light and to perform the role of the helpful facilitator. Often, a customer gets what he or she needs by working with the system rather than going around it. This is especially valuable if the advocate also created the system.

When you are the advocate, customers want you to understand them before you start suggesting solutions. The customers want you to understand their technical needs as well as soft issues, such as schedules and budgets.

### **B.1.23 The Technocrat**

The technocrat is the advocate for new technology. When a system needs to be repaired or replaced, the technocrat puts more value in the new system because it is new, even if it still has bugs. The technocrat disdains those who seek comfort in old systems that may be "good enough." The technocrat can provide good counterbalance to the disaster worrier.

The value to the company of the technocrat is that he or she prevents the company from becoming technically stagnant.

The technocrat receives satisfaction from being surrounded by the latest new technology—dare we say new-toy syndrome.

When you are the technocrat, focus on the real value of a solution rather than assuming that newer is better.

### **B.1.24 The Salesperson**

The salesperson is not limited to selling tangible items. The salesperson may be selling a particular policy, new service, or proposal. He or she may be selling the SA team itself, either to upper management or to the customers. A salesperson is concerned with identifying the needs of customers and then convincing them that what he or she has to sell meets those needs. New services are easier to sell if the customers were involved in the specification and selection process.

The value to the company of the salesperson is that he or she makes the SA team's job easier. A great system that is never accepted by the customers is not useful to the company. A great policy that saves the company money is not helpful if the customers work around it because they don't understand the benefits.

The salesperson receives short-term satisfaction from "making the sale," but for real, lasting satisfaction the salesperson must develop a relationship with the customers and feel that he or she truly helps the customers in a meaningful way.

When you are the salesperson, understand and empathize with the needs of your customers. They want to be talked with, not to.

### **B.1.25 The Vendor Liaison**

The vendor liaison maintains a relationship with one or more vendors. The vendor liaison may know a vendor's product line better than anyone else in the group and be privy to upcoming products. The vendor liaison knows the vendor's policies, procedures, and terminology and how to use them to get the company's needs met when dealing with the vendor. The vendor liaison is a resource for the other SAs, thus saving calls to the vendor's salesperson.

The value to the company of the vendor liaison is having someone who understands and is dedicated to the company's needs when dealing with a vendor. Having a single point of contact saves resources.

The vendor liaison receives satisfaction from being the expert whom everyone respects, from being the first to know about vendor news, and from collecting the free lunches and shirts he or she receives.

When you are the vendor liaison, customers want you to be all-knowing about the vendor, open-minded about competing vendors, and a harsh negotiator when haggling over prices.

### **B.1.26 The Visionary**

The visionary looks at the big picture and has a vision of where the group should go. These individuals stay in contact with customers and business leadership to not only understand broad company goals, but also understand where technology is going and which operational improvements are possible.

The value to the company of the visionary is keeping the group focused on what's next.

Visionaries receive satisfaction when they look back over the years and see

that in the long term, they made a difference. All those incremental improvements accumulated to meet major goals.

When you are the visionary, customers want to know what's happening next and may not be too concerned with the long term. Your team's reputation for being able to execute a plan affects your ability to sell your vision to the customers.

### **B.1.27 The Mother**

The mother nurtures the customers. It's difficult to explain this role except through example. One SA spent her mornings walking through the halls, stopping by each person's office to see how things were. She would fix small problems and note the bigger problems to handle in the afternoon. She would answer many user-interface questions that a customer might have felt were too small to ask the helpdesk. The customers were making a big paradigm change (from X Terminals to PCs running X Terminal emulators), and this mothering was exactly what they needed. In her morning walks, this SA would answer hundreds of questions and resolve dozens of problems that would otherwise have been tickets submitted to the helpdesk. The customers got very used to this level of service and soon came to rely on her morning visits as part of what kept them productive.

The value to the company of the mother is his or her high degree of hand-holding, which can be critical at times of great change or with nontechnical customers. The personal contact also ensures a more precise understanding of the customers' needs.

The mother receives satisfaction from the personal relationships he or she develops with his or her customers.

When you are the mother, ensure your customers' immediate needs are being met and put less emphasis on the long-term strategy. You must remember to keep an eye on the future and not get too absorbed in the present.

### **B.1.28 The Monitor**

The monitor notices how well things are running. Sometimes, the monitor uses low-tech methods, using the same services that his or her customers use. Although the SAs may have a private file server, this person stores his or her files on the file server that the customers use, so the monitor can "feel their pain." As this person becomes more sophisticated, the monitor automates his or her monitoring but then watches the monitoring system's output and takes the

time to fix things rather than simply clear the alarms.

The monitor's value to the company is that problems are noticed before customers start complaining. This can give the perception of a trouble-free network.

The monitor receives satisfaction from being the first to notice a problem, from knowing that he or she is working on fixing a problem before customers report it, and from knowing that problems are prevented by monitoring capacity issues.

When you are the monitor, customers most likely don't know that you exist. If they did, they would want your testing to simulate their real workloads; that is, they would want end-to-end testing. For example, it isn't good enough to know that a mail server is up. You must test that a message can be submitted, relayed, delivered, and read.

### **B.1.29 The Facilitator**

The facilitator has excellent communication skills and tends to turn impromptu discussions into decision-making meetings. The facilitator is often asked to run meetings, especially large meetings in which keeping focus can be difficult.

The facilitator adds value by optimizing processes. The facilitator may not take on a lot of action items but gets groups of people to agree to what needs to be done and who is going to do it. The facilitator keeps meetings efficient and fun.

The facilitator receives satisfaction from seeing people come to agreement on goals and taking the initiative to see the goals completed.

When you are the facilitator, the other members on your team want you to facilitate all their discussions, but it is more important to teach everyone to be a good facilitator. Other people will naturally emulate your style, but more direct coaching and training can create an environment in which everyone is a better facilitator. Meetings run better when everyone in the room understands facilitation concepts and practices.

### **B.1.30 The Customer/SA**

Sometimes a customer is also an SA, perhaps because that person has certain responsibilities that require privileged access or used to be an SA and retains some of those responsibilities.

The value that a customer/SA provides to the company is to distribute SA

responsibilities, thereby reducing the work for the IT team. However, without clear boundaries delineating what these individuals are and aren't supposed to do, it can create confusion and problems. In situations in which there is only a single SA, a customer/SA can help with daily operational issues and provide vacation coverage. Having an additional person who can change backup tapes, create user accounts, and solve the most frequent problems can be very useful.

The customer/SA receives satisfaction from the role if he or she holds SAs or superuser access in high esteem. Alternatively, the customer/SA may receive some kind of "combat pay" for being cross-trained.

When you are the customer/SA, understand the boundaries set by the SA team so that your "help" is not interfering with their plans. Follow the procedures and policies set forth. Uphold the same ethical practices as the SA team.

### **B.1.31 The Helpdesker**

The helpdesker views his or her job as being centered on helping people. This individual reacts to customer requests and is not as interested in technical infrastructure and back-office work. The helpdesker views his or her job as helping customers with the system, and may prefer that things do not change because that would interfere with the processes the helpdesker has established.

The helpdesker adds value to the company by being the day-to-day human interface to the full SA team. Most customers never see any back-line support.

Helpdeskers receive satisfaction from the personal relationships that they develop and the warm glow of knowing that they have helped real live humans.

When you are the helpdesker, you have to deal with customers' issues on their schedules. If they perceive the situation to be an emergency, they expect you to stop everything and help them.

### **B.1.32 The Policy Navigator**

The policy navigator understands the rules and regulations of the bureaucracy and can help others navigate through or around them. The navigator can help someone get through a process or work around a policy without violating it.

The policy navigator adds value to the SA team by getting things done more quickly when having to deal with the system, and, when needed, working around the system without getting into hot water.

The policy navigator receives satisfaction from knowing that his or her connections and knowledge have contributed to a project.

When you are the policy navigator, you are expected to get things done, regardless of whether you stay within the system. This can put you in a difficult situation when the customer thinks it is easier to violate policy or work around the system.

## **B.2 Negative Roles**

The following sections describe some negative roles that you want to avoid.

### **B.2.1 The Bleeding Edger**

Sometimes, an SA can be so excited by new technology that he or she seeks to unleash it on the customers before it is ready. Rather than being on the leading edge, he or she keeps the company on the *bleeding edge*. In this case, the customers always seem to be enduring the pain of buggy new services.

### **B.2.2 The Technology Staller**

The counterbalance to the bleeding edger is the person who stalls on the use of any new technology. The technology staller sometimes has a favorite excuse, such as not being satisfied with the backout plan, or suggesting that a technology is too new and untested. The technology staller is happy with the current OS release, the current OS distribution, the current brand of computers, amount of bandwidth, and type of network topology. The staller doesn't see the lack of technology refresh and the problems that it has created. Ironically, this person used to be on the cutting edge but has now become stuck in a rut. The technology staller may have been the person who eschewed the mainframes and adopted Unix, or who laughed at the workstation users who wouldn't adopt PCs. However cutting edge he or she once was, the technology staller has found something he or she is comfortable with and now has become the person the staller used to mock years ago.

### **B.2.3 The SA Who Cried Wolf**

The SA who cried wolf is worried about things that aren't happening or are unlikely to happen. Much of system administration is about managing risks, but this person thinks that all risks are unacceptable. This person slows projects from getting off the ground. This person predicts doom or failure without hard facts to back it up. Sometimes, such people are simply uncomfortable with anything they aren't in control of or about which they aren't educated. Sometimes, such a person will waste a lot of time working on problems that aren't even on the radar and ignore more pressing problems. The biggest danger with this person is that

when he or she is correct, he or she will be ignored.

### **B.2.4 The Cowboy**

The cowboy rushes into fixing systems or implementing new services without proper planning, thinking through the consequences, or developing a backout plan. The cowboy does not bother to ask his or her manager beforehand or to check with the customers, does not test his or her work properly to verify that it is working, and goes home without telling anyone what he or she has done. Cowboys see themselves as talented and fast-working, and think that others try to put too much red tape in the way and are unappreciative of their talents. The cowboy doesn't document anything and simply knows that he or she is invaluable to the company.

---

#### **Why Plan When You Can Get Work Done?**

A midsize computing hardware manufacturer had a cowboy in a senior SA position. His management had brought in a consulting group to re-architect the network and to build a transition plan from the old network to the new one. The plan was agreed on, the equipment was ordered, a schedule was laid out and agreed on with the customers, and test plans were being built with the customers. The day that the new equipment arrived, the cowboy stayed late, ripped out the old network equipment, and installed the new gear. He did not use the new architecture; he ignored all the components of the transition plan that were intended to solve unrelated problems that could be solved by the process of the transition; and he did not do much, if any, testing.

The next day, when people returned to work, many of them found that they had no network connection—including the CEO, the entire customer-support department, the cowboy's management chain, and many of the engineers. The helpdesk and the rest of the SA group had no idea what had happened, because the cowboy had not told anyone, and he didn't bother to arrive at work until much later in the day. After all, he had stayed late!

He was still proud of what he had done and unrepentant because he viewed the transition plan and test plan as a big waste of time and money. He enjoyed showing that he could do it on his own in a few hours, when the consultants were taking so much time over it. He was oblivious to the cost of the major outage that he had caused and the damage he had inflicted on the SA group's reputation.



---

## **B.2.5 Slaves, Scapegoats, or Janitors**

Sometimes, SAs are in the role of being slaves, scapegoats, or janitors. Slaves are expected to do tasks without question, even if they might be able to suggest better processes if they were filled in on the big picture. Sometimes, others use SAs as scapegoats. In that case, all bad things that happen are blamed on the SAs. The SAs are blamed for a project being late, even if the customers weren't communicating their needs. Sometimes, SAs are thought of as janitors—people who aren't valuable to the company's direct business, but rather are unskilled workers who are purely overhead.

All three of these roles are problems that tend to result from managers not understanding the purpose of SAs in their own organization. However, it is the SAs' responsibility to fix these problems by increasing communication and working on their team's visibility within the organization.

## **B.3 Team Roles**

Within an SA team, people play many different roles. The diversity helps keep the team balanced. With the exception of the martyr, every team needs people to fulfill these roles. There are other roles that exist within an SA team, but these are the ones that deserve highlighting.

### **B.3.1 The End-to-End Expert**

The end-to-end expert understands the technology being used, from the lowest to the highest levels. The end-to-end expert is critical for solving both obscure problems and major outages. Obscure problems usually are the result of multiple simultaneous failures or strange interactions among different areas, and this role requires expertise in all areas to be able to solve them. Major outages affect multiple subsystems and require a deep understanding of the overall architecture to pinpoint the real problem.

### **B.3.2 The Outsider**

During a prolonged outage, the people working on the problem sometimes get the SA equivalent of writer's block. They're going in circles, unable to make a significant improvement in the situation. The role that is most useful here is the outsider who brings a fresh viewpoint to the situation. By making everyone explain what has happened so far, people often realize the solution. Sometimes, the role of this person is to encourage the others to seek outside help or escalate

the problem to higher authorities or to vendor support. At other times, this person's role is to simply recognize that it is time to give up: There is a backout plan and it should be implemented.

### **B.3.3 The Level-Focused Person**

Another role is the person who is very conscious of people's ranks or levels. Some people always try to solve problems at their own level. Technicians think they need to work harder. Programmers think that they need to introduce new software to fix the problem. However, an important role is the person who has an understanding of all the levels, including management, and can judge where best to solve a problem.

After weeks of trying to solve a problem, the level-focused person suggests that it would be less expensive and possibly more effective to simply get a higher level of management to announce that a certain practice is not allowed. It may be better to find the first manager up the organization chart who outranks the two bickering customers and give the problem to that person to solve. The level-focused person is also the person most likely to quote the old Internet adage, "Technology can't solve social problems," and then seek a policy change.

### **B.3.4 The Martyr**

The martyr feels that nobody else has as much work as he or she does. Resentful of the fact that nobody else works the long hours that he or she does, and unhappy with his or her lack of a social life or financial success, the martyr is unable to understand how other people can "make it" when they do so little. This person spends a lot of time bemoaning the problems of the world—mainly his or her own world. This can happen as a result of simple burnout or, more complexly, low self-esteem.

---

#### **Burnout and Depression**

Burnout happens when a person does not balance work with recreation. Sadly, in today's fast-paced, modern culture, some people grow up not learning the importance of relaxation. They feel that they must always be "at work." This extreme work ethic may be the secret to their success, but without the mental equivalent of preventive maintenance, it leads to burnout. At that point, working harder merely makes the situation worse. People who have reached this extreme may not be helped by a quantity of time off, and they might be even more resentful if they have nothing to do.

Self-esteem is something we learn (or don't learn) when we are young. Cognitive theorists believe that our mental state—whether we are, by default, happy or sad—is not a sign of whether good or bad things are happening to us but of how we react to whatever is happening to us. We can even be unhappy in response to good events if our self-esteem has been damaged to the point that no matter what happens, we feel worthless. When this happens, we corrupt good news and events, changing them into things to worry about: “He liked my work on that project! What if I can’t meet those expectations every time? What if my co-workers become jealous and resent me?” Various therapies are designed to help people in this situation.

If you feel yourself getting burned out or depressed, seek help. Most companies have an employee assistance program (EAP) that will provide a counselor who is easy to talk to and can help you evaluate your own situation. A counselor can also refer you to other places you can get help. If you prefer to go the self-help route, we recommend books such as *The Feeling Good Handbook* by Burns ([1999](#)).

---

### **B.3.5 Doers of Repetitive Tasks**

Some personality types lend themselves to doing repetitive tasks. These people should be valued because they solve immediate problems. This book can't stress enough the importance of automation, but some tasks (such as physically delivering new machines) can't be automated, or are not repeated enough to make their automation cost-effective. This role is an important one for the team. This person can take on the load of repetitive tasks from higher-skilled people. These smaller tasks are often excellent training for higher-level tasks.

### **B.3.6 The Social Director**

The social director boosts team spirit by finding reasons for the group to celebrate together; people's birthdays, anniversaries, and hiring dates are just a few. Getting people to relate to one another in a non-work context can build team cohesion. The key to success in this position is to make sure that people don't feel imposed on and to make sure that you don't overdo it and risk the boss considering you a time waster.

---

#### **Monthly Birthday Lunch**

One SA team had a policy of going out to lunch once a month. The people

with birthdays that month didn't have to pay and were responsible for coordinating the next month's lunch. It was a valuable team-building tradition. Because of a couple of factors, the tradition stopped for a year. To restart the process, the team held a lunch for anyone who had a birthday in the last 12 months, and the boss paid the entire bill.

---

### **B.3.7 Captain Break Time**

During an emergency situation, it is key to have someone who notices when people are becoming exhausted and encourages them to take a break. People may feel that the task is too important to stop working on it, but Captain Break Time realizes that nothing is getting done and that a break might refresh people. Tom is known to disappear and return with pizzas and drinks. Often, by walking away from a problem for a time, people will think about it differently and come up with better solutions.

## **B.4 Summary**

We hope that as you read this appendix you saw yourself in one or more of these roles. Now you can be conscious of what motivates you and how you fit into your team.

This discussion may help you realize that you play many roles in your organization and may play a special role on your team. You wear many hats. As you wear each hat, you can be conscious of the value you should be providing to the company, what motivates you, and what customers expect from you.

Maybe you learned something good or bad about yourself. Maybe you learned something new about yourself or validated feelings that you already had. It may convince you to seek personal improvement in an area of your life. The next time you consider a career change, you might spend more time thinking about the roles you like to be in or the roles that match your skills, and compare those to the roles you will be asked to play in the new position. Don't be worried if every role appeals to you. Maybe your destiny in life is to have many roles, or maybe as you gain more knowledge and experience, you will find new clarity in your feelings about these things.

Maybe this appendix made you realize that your organization is in dire need of one of the roles described here. Maybe you've noticed something missing from your SA team. Maybe you should try to take on that role more often, encourage others to take on that role, or look for those skills the next time you hire someone. You may realize that your organization has too many people in a

particular role, which indicates that you need balance. Maybe this appendix helped you notice that a certain person is fulfilling a negative role and needs coaching to change.

Maybe this appendix made you realize that people on your team all play too many roles or don't play enough roles or the roles aren't balanced.

Most of all, we hope that this discussion helped you understand the people on your team who are not like you. Now you can appreciate the value they bring to the team. A strong team includes people with different skills and from various backgrounds; each brings something unique to the table. Those differences do not divide a team, but rather make it stronger. For that to happen, the team must be conscious of the differences and take time to value them rather than run from them.

## **Exercises**

1. In which roles do you see yourself? (Don't be humble.) Do you want to remain in these roles? Which unfamiliar roles appeal to you?
2. In which roles did you see yourself early in your career? How is that different from the roles you are in now?
3. Which roles do you see yourself fulfilling in the future?
4. Which personal changes would you like to make so that you can better serve in your role or move into new roles?
5. Which roles do you not like? Why? How are they different from the roles you are currently fulfilling?
6. Determine which role each person on your team portrays. If you feel it wouldn't destroy your friendship, have the team members do the same thing, and share the lists. How much did the therapy bills cost?
7. Which roles are missing from your team? How can you develop those roles in your team?
8. Which roles do you feel should be added to this list? How are they valuable to the organization or team, or in which ways are they negative roles? What are the motivators for these roles? What do customers expect from them?

# Bibliography

- Abbott, M. & Fisher, M. (2010). *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise*, Addison-Wesley.
- Anderson, D. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*, Blue Hole Press.
- Andrews, M. & Whittaker, J. (2006). *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*, Addison-Wesley.
- Andreyev, A. (2014). Introducing data center fabric, the next-generation Facebook data center network.  
<http://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-datacenter-network/>
- Benson, J. & Barry, T. D. (2011). *Personal Kanban: Mapping Work | Navigating Life*, Modus Cooperandi Press. <http://www.personalkanban.com/>
- Bent, Jr., W. H. (1993). System administration as a user interface: An extended metaphor, *Systems Administration (LISA VII) Conference*, USENIX, pp. 209–212.
- Bentley, J. L. (2000). *Programming Pearls, 2nd ed.*, ACM Press Series, Addison-Wesley.
- Beyer, B., Jones, C., Petoff, J. & Murphy, N. R. (2016). *Site Reliability Engineering: How Google Runs Production Systems*, O'Reilly.
- Blanchard, K. & Johnson, S. (2015). *The One Minute Manager*, William Morrow.
- Braden, R. T. (1989). RFC 1123: Requirements for Internet hosts—application and support. See also STD0003. Updates RFC0822 (Crocker 1982). Updated by RFC2181 (Elz & Bush 1997). Status: Standard.
- Brechner, E. (2015). *Agile Project Management with Kanban*, Best Practices Series, Microsoft Press.
- Burgess, M. (2000). *Principles of Network and System Administration*, Wiley.
- Burns, D. D. (1992). *Feeling Good: The New Mood Therapy*, Avon Books.
- Burns, D. D. (1999). *The Feeling Good Handbook*, Plume.

- Chalup, S. R. (2000). Technical interviews, *;login:* **25**(8): 52–61.
- Chalup, S. R., Hogan, C., Kulosa, G., McDonald, B. & Stansell, B. (1998). Drinking from the fire(walls) hose: Another approach to very large mailing lists, *Twelfth Systems Administration Conference (LISA '98)*, USENIX, p. 317.
- Clarke, R. (2004). *Against All Enemies: Inside America's War on Terror*, Free Press.
- Coleman, T. S. (2011). *A Practical Guide to Risk Management*, Research Foundation of CFA Institute.
- Corman, J. (2015). Continuous acceleration: Why continuous everything needs a supply chain approach.  
<http://www.usenix.org/conference/lisa15/conference-program/presentation/corman>
- Crawley, D. (2013). *The Compassionate Geek: How Engineers, IT Pros, and Other Tech Specialists Can Master Human Relations Skills to Deliver Outstanding Customer Service*, 3rd ed., soundtraining.net.
- Crittenden, J. (1995). [Episode 2F10] And Maggie Makes Three, *The Simpsons*.  
<http://www.simpsonsarchive.com/episodes/2F10.html>
- Crocker, D. (1982). RFC 822: Standard for the format of ARPA Internet text messages. See also STD0011. Obsoletes RFC0733. Updated by RFC1123, RFC1138, RFC1148, RFC1327, RFC2156. Status: Standard.
- Curtin, M. (1998). Snake oil warning signs: Encryption software to avoid.  
<http://www.interhack.net/people/cmcurtin/snake-oil-faq.html>
- Darmohray, E. T. (2012). *Job Descriptions for System Administrators*, 3rd ed., Short Topics in System Administration 22, The USENIX Association.
- Davis, C., Vixie, P., Goodwin, T. & Dickinson, I. (1996). RFC 1876: A means for expressing location information in the domain name system. Updates RFC1034, RFC1035. Status: Experimental.
- Dijker, B. L. (1999). Round and round we go, *;login:* **24**(2): 8.
- Dijkstra, E. W. (1975). Guarded commands, nondeterminacy and formal derivation of programs, *Communications of the ACM* **18**(8): 453–457.  
<http://doi.acm.org/10.1145/360933.360975>
- Doran, G. T. (1981). There's a S.M.A.R.T. way to write management's goals and objectives, *Management Review* **70**(11): 35–36.



- Edwards, D. & Shortland, A. (2012). Integrating DevOps tools into a service delivery platform.  
<http://dev2ops.org/2012/07/integrating-devops-tools-into-a-service-delivery-platform-video>
- Elz, R. & Bush, R. (1996). RFC 1982: Serial number arithmetic. Updates RFC1034, RFC1035. Status: Proposed Standard.
- Elz, R. & Bush, R. (1997). RFC 2181: Clarifications to the DNS specification. Updates RFC1034, RFC1035, RFC1123. Status: Proposed Standard.
- Evard, R. (1997). An analysis of UNIX system configuration, *Eleventh Systems Administration Conference (LISA '97)*, USENIX, p. 179.
- Fall, K. R. & Stevens, W. R. (2012). *TCP/IP Illustrated, Volume 1: The Protocols, 2nd ed.*, Addison-Wesley.
- Fine, T. A. & Romig, S. M. (1990). A console server, *LISA IV Conference Proceedings*, USENIX, pp. 97–100.
- Finke, J. (1994a). Automating printing configuration, *LISA VIII Conference Proceedings*, USENIX, pp. 175–183.
- Finke, J. (1994b). Monitoring usage of workstations with a relational database, *LISA VIII Conference Proceedings*, USENIX, pp. 149–157.
- Finke, J. (1995). SQL\_2\_HTML: Automatic generation of HTML database schemas, *Ninth Systems Administration Conference (LISA '95)*, USENIX, pp. 133–138.
- Finke, J. (1996). Institute white pages as a system administration problem, *10th Systems Administration Conference (LISA'96)*, USENIX, pp. 233–240.
- Finke, J. (1997). Automation of site configuration management, *Eleventh Systems Administration Conference (LISA '97)*, USENIX, p. 155.
- Fowler, M. (2012). SnowflakeServer.  
<http://martinfowler.com/bliki/SnowflakeServer.html>
- Fowler, M. (2016). InfrastructureAsCode.  
<http://martinfowler.com/bliki/InfrastructureAsCode.>
- Fulmer, R. & Levine, A. (1998). AutoInstall for NT: Complete NT installation over the network, *Large Installation System Administration of Windows NT Conference*, USENIX, p. 27.
- Gladstone, B. & Garfield, B. (2012). A new whistleblower law, *On The Media*.  
<http://www.wnyc.org/story/253973-new-whistleblower-law/>



- Gladstone, B. & Garfield, B. (2015). Word watch: Whistleblower, *On The Media*. <http://www.wnyc.org/story/word-watch-whistleblower>
- Gladwell, M. (2007). *Blink: The Power of Thinking Without Thinking*, Little, Brown.
- Goldfuss, A. (2015). Scalable meatfrastructure: Building stable DevOps teams. Presented at USENIX LISA 2015. <http://www.usenix.org/conference/lisa15/conference-program/presentation/goldfuss>
- Guth, R. & Radosevich, L. (1998). IBM crosses the Olympic finish line, *InfoWorld*.
- Harris, D. & Stansell, B. (2000). Finding time to do it all, *;login*: 25(6): 72–78. <http://www.conserver.com/consoles/>
- Harris, T. (2002). The case of the 500mile email. <http://www.ibiblio.org/harris/500milemail-faq.html>
- Harris, T. (2004). A new approach to scripting, *Proceedings of the 18th Conference on Systems Administration, (LISA 2004)*.
- Humble, J. & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, A Martin Fowler Signature Book, Addison-Wesley.
- Hunter, T. & Watanabe, S. (1993). Guerrilla system administration: Scaling small group systems administration to a larger installed base, *Systems Administration (LISA VII) Conference*, USENIX, pp. 99–105.
- Johnson, S. (1991). *The One Minute Sales Person*, Avon Books.
- Kernighan, B. W. & Pike, R. (1999). *The Practice of Programming*, Addison-Wesley.
- Kim, G., Behr, K. & Spafford, G. (2013). *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*, IT Revolution Press.
- Kim, G., Humble, J., Debois, P. & Willis, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*, IT Revolution Press.
- Klein, D. V., Betser, D. M. & Monroe, M. G. (2014a). Making “push on green” a reality. Presented at USENIX LISA 2014. <http://www.usenix.org/conference/lisa14/conference-program/presentation/klein>

- Klein, D. V., Betser, D. M. & Monroe, M. G. (2014b). Making “push on green” a reality: Issues & actions involved in maintaining a production service, *;login:* **39**(5): 26–32.
- Koren, L. & Goodman, P. (1991). *The Hagglers Handbook: One Hour to Negotiating Power*, W. W. Norton.
- Kruger, J. & Dunning, D. (1999). Unskilled and unaware of it: How difficulties in recognizing one’s own incompetence lead to inflated self-assessments, *Journal of Personality and Social Psychology* **77**(6): 1121–1134.
- Lakein, A. (1996). *How to Get Control of Your Time and Your Life*, New American Library.
- Lear, E., Fair, E., Crocker, D. & Kessler, T. (1994). RFC 1627: Network 10 considered harmful (some practices shouldn’t be codified). Obsoleted by BCP0005, RFC1918 (Rekhter, Moskowitz, Karrenberg, de Groot & Lear 1996). Status: Informational.
- Leber, J. (1998). *Windows NT Backup and Restore*, O’Reilly.
- Levinson, M. (2016). *The Box: How the Shipping Container Made the World Smaller and the World Economy Bigger*, 2nd ed., Princeton University Press.
- Libes, D. (1990). RFC 1178: Choosing a name for your computer. See also FYI0005. Status: Informational.
- Limoncelli, T. (1998). Please quit, *;login:* **23**(5): 38.
- Limoncelli, T. A. (1999). Deconstructing user requests and the nine step model, *Proceedings of the 13th USENIX LISA Conference*, p. 35.
- Limoncelli, T. A. (2006). *Time Management for System Administrators*, O’Reilly.
- Limoncelli, T. A. (2012). OpenFlow: A radical new idea in networking, *Queue* **10**(6): 40:40–40:46. <http://queue.acm.org/detail.cfm?id=2305856>
- Limoncelli, T. A. & Cerf, V. G. (2011). Successful strategies for IPv6 rollouts: Really!, *Communications of the ACM* **54**(4): 44–48. <http://queue.acm.org/detail.cfm?id=1959015>
- Limoncelli, T. A., Hogan, C. J. & Chalup, S. R. (2017). *The Practice of System and Network Administration, Volume 1*, 3rd ed., Addison-Wesley.
- Limoncelli, T., Reingold, T., Narayan, R. & Loura, R. (1997). Creating a network for Lucent Bell Labs Research South, *Eleventh Systems Administration Conference (LISA ’97)*, USENIX, San Diego, California, p.

123.

- Lions, J. (1996). *Lions' Commentary on UNIX 6th Edition, with Source Code, Peer-to-Peer Communications*.
- MacKenzie, R. A. (1997). *The Time Trap: The Classic Book on Time Management*, 3rd ed., AMACOM.
- Maniago, P. (1987). Consulting via mail at Andrew, *Large Installation System Administrators Workshop Proceedings*, USENIX, pp. 22–23.
- Mathis, M. (2003). The case for raising the Internet MTU, Cisco200307.  
<http://staff.psc.edu/mathis/MTU/index.html>
- McEntee, K. (2012). Netflix's transcoding transformation.  
<http://youtu.be/B1RMcCqEKwE>
- McKenty, J. (2013). Press release: Piston cloud turns out the lights in the software-defined data center with Piston Enterprise Openstack 2.0.  
<http://www.marketwired.com/press-release/piston-cloud-turns-out-lights-software-defined-data-center-with-piston-enterprise-openstack-1776772.htm>
- McKinley, D. (2012). Why MongoDB never worked out at Etsy.  
<http://mcfunley.com/why-mongodb-never-worked-out-at-etsy>
- McKusick, M. K., Neville-Neil, G. V. & Watson, R. N. M. (2014). *The Design and Implementation of the FreeBSD Operating System*, 2nd ed., Addison-Wesley.
- McNutt, D. (1993). Role-based system administration or who, what, where, and how, *Systems Administration (LISA VII) Conference*, USENIX, pp. 107–112.
- Meggs, R. (2012). Making it virtually easy to deploy on day one.  
<http://codeascraft.com/2012/03/13/making-it-virtually-easy-to-deploy-on-day-one/>
- Menter, E. S. (1993). Managing the mission critical environment, *Systems Administration (LISA VII) Conference*, USENIX, pp. 81–86.
- Morgenstern, J. (1998). *Organizing from the Inside Out: The Foolproof System for Organizing Your Home, Your Office, and Your Life*, Henry Holt.
- Palfreman, J. (1997). Why the French Like Nuclear Energy, *Frontline*.  
<http://www.pbs.org/wgbh/pages/frontline/shows/react>
- Paterson, R. J. (2000). *The Assertiveness Workbook: How to Express Your Ideas and Stand Up for Yourself at Work and in Relationships*, New Harbinger

Publications.

- Phillips, G. & LeFebvre, W. (1998). *Hiring System Administrators*, Short Topics in System Administration 5, The USENIX Association.
- Plonka, D. & Tack, A. J. (2009). An analysis of network configuration artifacts, *23rd Systems Administration Conference (LISA'09)*, USENIX Association, pp. 79–91. <http://www.usenix.org/conference/lisa-09/analysis-network-configuration-artifacts>
- Powers, Dr. P. & Russell, D. (1993). *Love Your Job!: Loving the Job You Have, Finding a Job You Love*, O'Reilly.
- Preston, W. C. (1999). *Unix Backup and Recovery*, O'Reilly.
- Punyon, J. (2015). Providence: Failure is always an option. <http://jasonpunyon.com/blog/2015/02/12/providence-failure-is-always-an-option/>
- Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J. & Lear, E. (1996). RFC 1918: Address allocation for private internets. See also BCP0005. Obsoletes RFC1627, RFC1597. Status: Best Current Practice.
- Richardson, J. R. & Gwaltney, Jr., W. A. (2005). *Ship It! A Practical Guide to Successful Software Projects*, Pragmatic Bookshelf.
- Ries, E. (2009). Minimum viable product: A guide. <http://www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html>
- Ringel, M. F. & Limoncelli, T. A. (1999). Adverse termination procedures, or How to fire a system administrator, *Proceedings of the 13th USENIX LISA Conference*, USENIX, p. 45.
- Russinovich, M., Solomon, D. A. & Ionescu, A. (2012). *Windows® Internals, Part 1, 6th ed.*, Developer Reference, Microsoft Press.
- Schafer, P. (1992). bbn-public: Contributions from the user community, *Systems Administration (LISA VI) Conference*, USENIX, pp. 211–213.
- Schneier, B. (2007). A security market for lemons. [http://www.schneier.com/blog/archives/2007/04/a\\_sec](http://www.schneier.com/blog/archives/2007/04/a_sec)
- Singh, A. *et al.* (2015). Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network, *Sigcomm '15 Proceedings*.
- Smallwood, K. (1992). SAGE views: Whither the customer?, *login*: **17**(5): 15–16.

- Smith, M. J. (1975). *When I Say No, I Feel Guilty*, Bantam.
- Smith, M. J. (2000). *When I Say No, I Feel Guilty, Vol. II: For Managers and Executives*, A Train Press.
- Spolsky, J. (2007). *Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent*, Apress.
- Stern, H., Eisler, M. & Labiaga, R. (2001). *Managing NFS and NIS*, 2nd ed., O'Reilly.
- Stoll, C. (1989). *The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage*, Doubleday.
- Toigo, J. (2003). *Disaster Recovery Planning: Preparing for the Unthinkable*, 4th ed., Prentice Hall.
- Viega, J., Warsaw, B. & Manheimer, K. (1998). Mailman: The GNU mailing list manager, *Twelfth Systems Administration Conference (LISA '98)*, USENIX, p. 309.
- Watters, J. (2010). *The Business Continuity Management Desk Reference: Guide to Business Continuity Planning, Crisis Management & IT Disaster Recovery*, Leverage Publishing.
- Zwicky, E. D., Simmons, S. & Dalton, R. (1990). Policy as a system administration tool, *LISA IV Conference Proceedings*, USENIX, pp. 115–124.

# Index

AAA (authentication, authorization, and accounting), [719–720](#)

Abbott, M., [322](#)

Academic institutions, removing SAs at, [1013–1014](#)

Acceptable use policies (AUPs), [408](#), [874–875](#)

Acceptance testing

- data migration, [368](#)

- definition, [60](#)

- software upgrades, [295](#)

- staging areas, [342](#)

Access

- console, [475–476](#)

- datacenter requirements, [455–456](#)

- email servers, [733](#)

- fired system administrators, [1007–1011](#)

- maintenance windows, [621](#)

- managing, [433](#)

- monitoring systems, [685–686](#)

- nameservices policy, [721–723](#)

- preventing, [81](#)

- printer, [752–753](#)

- privileged-access code of conduct, [877–878](#)

- after server upgrades, [606](#)

- service launches, [349](#)

- software repositories, [835](#)

- workstations vs. servers, [247](#)

Accidental file deletions, restores for, [796–797](#)

Accomplishment walls, [972](#)

Accountability in service requirements, [287](#)

Accounting in workstations disposal, [131](#)

Accounts, workstation, [86–88](#)

Accuracy in documentation, [553–554](#)

Achievable goals in S.M.A.R.T., [338](#)

## ActiveDirectory (AD)

- GPOs, [122](#)

- Kerberos authentication system, [300](#)

- LDAP, [718](#)

- workstations, [86](#)

Active installation of packages, [830](#)

Active items in Kanban, [8](#)

Active Listening, [950](#)

- mirroring, [951–952](#)

- reflection, [953–954](#)

- summary statements, [952–953](#)

AD. *See* [ActiveDirectory \(AD\)](#)

Active monitoring, [682–683](#)

Ad hoc solution finder role, [1097](#)

Ad hoc support tier, [167](#)

## Addresses

- email, [731](#)

- IP. *See* [IP addresses](#)

Adds in datacenters, [466](#)

Administrative networks, [257–258](#)

“Adverse Termination Procedures,” [1005](#)

## Advertising

- helpdesks, [500–501](#)

- jobs, [370](#), [980](#), [983](#)

After-hours support coverage, [499–500](#)

*Agile Project Management with Kanban*, [12](#)

## Alerts

- cascading, [684–685](#)

- real-time monitoring, [679–682](#)

## Aliases

- IP addresses, [315](#)

- service names, [314–315](#)

## Alignment

- asset depreciation, [163–164](#)

- dependencies, [311–313](#)

- priorities with customer expectations, [920–922](#)
- All eggs in one basket server hardware strategy, [222–224](#)
- Allman, Eric, [731](#)
- Amazon AWS service, [56](#), [238](#)
- Amazon Elastic Compute Cloud (Amazon EC2), [39](#)
- Amber zone servers, [412](#)
- Amicable terminations, [1012](#)
- Analogies, [37–39](#)
- “Analysis of Network Configuration Artifacts,” [436](#)
- “Analysis of Unix System Configuration,” [117–118](#)
- Anderson, D., [12](#)
- Andreyev, A., [412](#)
- Announcements
  - changes, [486](#)
  - major outages, [522–523](#)
  - server upgrades, [602–603](#)
- Anonymizing redirection services, [562–563](#)
- Answer files in OS installation, [149](#)
- Antivirus software, [95–97](#)
- Anycast addresses, [715](#)
- APIs (application program interfaces), [56](#)
- Apollo program, [336](#)
- Appearances, [914–915](#)
- Apple operating systems, [83](#)
- Appliances, server, [241–242](#)
- Application control software, [95](#)
- Application layer in OSI model, [400–401](#)
- Application orchestration, [68](#)
- Application program interfaces (APIs), [56](#)
- Applications
  - datacenter requirements, [454](#)
  - fired SA access, [1009](#)
  - response time monitoring, [688–689](#)
  - updating, [123–128](#)
  - web servers, [862](#), [864](#)



workstation service definition, [159](#)

Appreciation, employee, [1001](#)

Approval committees for launches, [356–357](#)

Approvals for ready lists, [338](#)

APT repository systems, [829](#)

## Architecture

centralization, [645](#)

customer care, [525](#)

network. *See* [Network architecture](#)

print service, [751–754](#)

service, [305](#)

workstation. *See* [Workstation architecture](#)

## Archives

Docker Container format, [48–51](#)

restores for, [797–798](#)

## ARM architecture

CPUs, [270](#)

grid computing, [236](#)

*Art of Scalability*, [322](#)

Artifacts in packages, [831](#)

ASN.1 format, [678](#)

Assertions in ready lists, [338](#)

*Assertiveness Workbook*, [961](#)

## Assessments

greatness, [1020–1021](#)

levels, [1023–1024](#)

levels of improvement, [1030–1031](#)

operational. *See* [Operational assessments](#)

operational responsibilities, [1021–1022](#)

organizational, [1029–1030](#)

questions and look for's, [1025](#)

service, [1025–1028](#)

starting, [1031–1032](#)

## Assets

in refresh cycles, [163–164](#)

- in server hardware strategy, [225](#)
- Assigned device phase, [434](#)
- Assumer stereotype, [520](#)
- AT&T split, [375–376](#)
- Atomic operating system, [234](#)
- Atomic writes, [783](#)
- Attitude, [918–920](#)
- Attrition, [1080](#)
- Audience for change review boards, [577](#)
- Audit logs for configuration changes, [436](#)
- Audit trails, [433](#)
- AUPs (acceptable use policies), [408](#), [875](#)
- Authentication
  - databases, [1015–1016](#)
  - NACs, [405](#)
  - nameservices, [719](#)
  - wireless office networks, [408](#)
  - workstations, [87](#)
- Authentication, authorization, and accounting (AAA), [719–720](#)
- Authoritative DNS servers, [716–717](#)
- Authority
  - nameservices, [713](#)
  - namespaces, [706–708](#)
- Authorization
  - in AAA, [719–720](#)
  - centralization, [646–647](#)
  - workstation, [86–88](#)
- Auto manufacturers, variation handling by, [52](#)
- Automation, [865–866](#)
  - backups and restores, [801](#), [816–819](#)
  - benefits, [924–925](#)
  - build environments, [843–844](#)
  - consistency, [53](#)
  - deployment process, [437](#)
  - DNS/DHCP infrastructure software, [365](#)

- email service, [737](#)
- fire drill requests, [798](#)
- fix it once principle, [547–549](#)
- hardware life cycle, [467](#)
- myths, [74](#)
- nameservice changes, [724–726](#)
- OS installation, [4](#), [142–143](#), [148–152](#)
- rapid release, [360](#)
- real-time monitoring, [681–683](#)
- small batches, [26](#)
- updates with user approval, [127–128](#)
- web services, [867–868](#)
- workstation fleet logistics, [180](#)
- workstation updates, [93](#)

Autoscale features, [39](#)

#### Availability

- datacenter requirements, [452](#)
- maintenance windows, [633–634](#)
- monitoring, [676](#)
- redundancy. See [Redundant Array of Independent Disks \(RAID\)](#)
- wiring, [464](#)
- workstations vs. servers, [247](#)

back-office systems, [284](#)

Back-sliding in workstation standardization, [194](#)

Backgrounds of team members, [987–988](#)

Backing out of server upgrades, [605–606](#)

Backlog of items in Kanban, [8–9](#)

#### Backout plans

- change management, [573–574](#)
- server upgrades, [600](#)
- service conversions, [383–385](#)

#### Backups and restores

- backup automation, [816–819](#)
- case studies, [652](#), [806](#)

- centralization, [819](#)
- consumables, [809–815](#)
- corporate guidelines, [799–800](#)
- data-recovery SLAs and policies, [800–801](#)
- data storage, [775–777](#)
- disaster recovery, [298](#)
- high-availability databases, [809](#)
- media and off-site storage, [812–814](#)
- overview, [793–794](#)
- performance, [799–800](#)
- RAID, [276](#)
- restore issues, [815–816](#)
- restore reasons, [795–799](#)
- schedules, [801–807](#)
- servers, [223](#)
- service launches, [351](#)
- starting, [794–795](#)
- tape inventory, [811–812](#)
- technology, [820–821](#)
- time and capacity planning, [807–809](#)

Bagley, John, [687](#)

Balancing work and personal life, [966–967](#)

Bandwidth

- dedicated lines, [417](#)
- vs. latency, [330–332](#)
- for monitoring systems, [673](#)
- usage monitoring, [779](#)
- VLANs, [404–405](#)

Barry, T., [11](#)

Basic services, [284](#)

Batch work in DevOps, [17](#)

Beautiful snowflakes server strategy, [224–227](#)

Bell Labs

- AutoPatch system, [125](#)
- change management, [575](#)

- documentation, [554](#)
- dress code, [488](#)
- gradual roll-outs, [379](#)
- hiring system administrators, [981](#)
- IP network splitting, [182](#)
- layers vs. pillars approach, [377](#)
- naming standards, [702](#)
- NFS dependencies, [310](#)
- on-site spares, [260](#)
- onboarding, [202](#), [207](#)
- OS installation, [139–140](#)
- platform support, [165](#), [166–167](#)
- post-installation tasks, [151](#)
- printer names, [697](#)
- repositories, [846](#)
- split, [374–376](#)
- user IDs, [706](#)
- visibility, [927](#)
- workstation standardization case study, [196–198](#)

Benson, J., [11](#)

Best practices fro vendor, [308](#)

Beta repositories, [845](#)

Betser, D. M., [356](#)

Binary packages, [831](#)

Birthday lunches, [1111](#)

Blacklisting workstations, [95](#)

Blade servers, [237–238](#)

Blanchard, K. H., [960](#)

Bleeding edger role, [1107](#)

Blocks in disks, [760–761](#)

Blog states, [47](#)

Boot disks, mirroring, [306](#)

Booting

- maintenance windows, [622–624](#)
- PXE, [148](#)

- Bosses, firing, [1012–1013](#)
- Bot farms, [859](#)
- Bottlenecks, [18–20](#)
- Brain-teaser questions in interview process, [995](#)
- Brechner, E., [12](#)
- Bring your own device (BYOD), [110](#)
  - costs, [112](#)
  - pros and cons, [111](#)
  - security, [111–112](#)
  - strategies, [110–111](#)
  - usability, [112–113](#)
- Bubble-up dynamic backup schedules, [806](#)
- Budget administrator role, [1102](#)
- Budgeting
  - CMDB for, [184](#)
  - datacenter, [453](#)
- Bugs
  - classes, [533](#)
  - debugging. *See* [Debugging](#)
  - ready lists, [337](#), [339](#)
- Build and initialize workstation processes, [118](#)
- Build environments, [843–845](#)
- Building
  - datacenters, [450](#)
  - monitoring systems, [673](#)
  - sites, [1067](#)
- Bulk licenses, [880](#)
- Bulk purchases for servers, [228–235](#)
- Bundling cables, [469](#)
- Bureaucrats, dealing with, [944–945](#)
- Burger King, [51](#)
- Burnout, [1110–1111](#)
- Burns, David, [964](#), [976](#)
- Business Continuity Management Desk Reference*, [387](#)
- Business continuity planning, [393](#)

Business products for workstations, [104](#)  
Business requirements for datacenters, [452](#)  
Business view for cloud-based services, [239](#)  
Buy-in and approval  
    maintenance windows, [613–615](#)  
    in service requirements, [287](#), [292](#)  
Buy in bulk and allocate fractions server strategy, [228–235](#)  
Buzzword-compliant tools, [537](#)  
BYOD. See [Bring your own device \(BYOD\)](#)  
BYOD-lite strategy, [111](#)  
BYOD-mixed model strategy, [110](#)  
BYOD-only strategy, [110](#)

## Cables

    labeling, [439–440](#)  
    patch, [468–472](#)  
    separating, [470–471](#)

## Caches

    CPUs, [271–272](#)  
    disks, [760–761](#)  
    DNS servers, [715–716](#)  
    software repositories, [836–837](#)

Cadence changes in onboarding, [212](#)

Calendars, launch, [348](#)

Call-volume ratios, [489](#)

CamelCase, [558](#)

## Candidates

    interview process, [991–994](#)  
    technical interviewing, [994–998](#)

Capability Maturity Model (CMM), [1023–1024](#)

## Capacity

    datacenters, [456](#), [459–465](#)  
    disks, [766](#)  
    monitoring, [676](#)  
    nameservices, [713–714](#)

- stranded, [228](#)
- VMs, [232–233](#)
- Capacity planner role, [1101](#)
- Capacity planning (CP)
  - backups and restores, [807–809](#)
  - operational assessments, [1043–1044](#)
  - operational responsibilities, [1022](#)
  - performance, [327–328](#)
  - software repositories, [840](#)
- Captain Break Time role, [1111](#)
- Cards, Kanban, [9](#), [11](#)
- Careful planner role, [1101](#)
- Carpenter analogy, [23–24](#)
- Carpenter principle, [545–547](#)
- Cartridges, toner, [756–757](#)
- Carts in datacenters, [478](#)
- CAs (certification authorities), [862](#)
- Cascading alerts, [684–685](#)
- “Case of the 500-Mile Email,” [539](#)
- Cassandra distributed storage systems, [251](#)
- Catalog of printers, [754–755](#)
- Catch-up sync for workstations, [91](#)
- Categorizations, launch, [355–356](#)
- Causes vs. symptoms in debugging, [531](#)
- CD (continuous delivery) in SDLC, [16](#)
- CDP (continuous data protection), [787](#)
- Cellular communications for maintenance windows, [627](#)
- CentOS Puppet Modules, [63–64](#)
- Central switches with patch panels, [409–410](#)
- Centralization, [639](#)
  - antivirus software control, [96](#)
  - approaches and hybrids, [642–643](#)
  - backups and restores, [819](#)
  - management, [293](#)
  - print service, [750](#)



rationale for, [640](#)

Centralization recommendations, [645](#)

architecture, [645](#)

authorization, [646–647](#)

communications, [651–652](#)

data management, [652](#)

datacenters, [649](#)

extranet connections, [647–648](#)

infrastructure, [648–656](#)

IP address space management, [650](#)

lab environments, [656](#)

leakage prevention, [648](#)

logging, [653–654](#)

monitoring, [653](#)

namespace management, [650–651](#)

networking, [649–650](#)

purchasing, [655–656](#)

security, [645–648](#)

support, [654–655](#)

Centralized models

description, [639](#)

funding, [895–896](#)

nameservice management, [726–728](#)

system administration, [903](#)

Centralized platform strategies, [642–643](#)

Centralized support, [444–445](#)

Centralizing services,

current solution, [660–661](#)

decentralization situations, [665–667](#)

DNS and DHCP, [664](#)

excellence in, [663](#)

fixing problems, [662](#)

low-hanging fruit, [664](#)

management support, [662](#)

monitoring, [685–686](#)

- overview, [659](#)
- plans, [661–662](#)
- slow pace, [663–664](#)
- Cerf, Vint, [426](#), [427](#)
- Certificates
  - fired system administrators, [1010–1011](#)
  - NACs, [405](#)
  - web services, [860–862](#)
- Certification authorities (CAs), [862](#)
- Certified engineering designs, [308](#)
- CFOs (chief financial officers), [897–898](#)
- CGI queries, [863](#)
- CGI servers, [852](#)
- Chalup, S. R., [980](#)
- Change completion deadlines for maintenance windows, [628](#)
- Change logs for server upgrades, [589](#)
- Change management (CM) and changes
  - case studies, [575](#), [579](#), [584–585](#)
  - change classifications, [571–572](#)
  - change request, [577–578](#)
  - change review boards, [568–569](#), [577–579](#)
  - communication, [576–577](#)
  - configuration, [436](#)
  - conflicting changes, [583](#)
  - datacenters, [453](#), [466](#)
  - freezes, [579–581](#)
  - Git, [583–584](#)
  - IBM’s Olympics failure and success, [568](#)
  - maintenance window proposals, [617–620](#)
  - nameservice policies, [723–724](#)
  - nameservice procedures, [724–726](#)
  - onboarding process, [210](#), [212](#)
  - operational assessments, [1045–1046](#)
  - operational responsibilities, [1022](#)
  - overview, [567](#)

- process overview, [570](#)
- proposals, [570](#)
- reboot test anecdote, [582–583](#)
- revision histories, [583–585](#)
- risk discovery and quantification, [572–573](#)
- scheduling, [574–575](#)
- service launch lessons learned, [344](#)
- teams, [581–583](#)
- technical planning, [573–574](#)
- tracking, [57–59](#)
- web service content, [867](#)
- wired office networks, [403](#)
- workstations, [128–129](#)

Change review boards (CRBs)

- change classifications, [571–572](#)
- change requests, [577–578](#)
- DNS/DHCP infrastructure software, [364](#), [366](#)
- overview, [568–569](#)
- process overview, [570](#)
- tiered, [578–579](#)

Chaotic launch category, [355–356](#)

Chat channels for maintenance windows, [626](#)

Checklists

- accounting, [131](#)
- decommissioning, [131](#)
- delivery, [186–187](#)
- deployment, [437](#)
- documentation, [556–557](#)
- exit, [1007](#), [1011](#), [1014–1015](#)
- installation, [214](#)
- launches, [336](#), [347](#)
- new hires, [917](#)
- onboarding, [205–206](#)
- OS installation, [146](#)
- rapid release, [360](#)

- service, [588–591](#)
- service requests, [297](#)
- wiki systems, [1068](#)
- workstation fleet logistics, [186](#)
- Chief financial officers (CFOs), [897–898](#)
- Chief operating officers (COOs), [897–898](#)
- Chief technology officers (CTOs), [897–898](#)
- Chocolatey package format, [839](#)
- Chubby lock service, [29](#)
- CI (continuous integration)
  - build environments, [844](#)
  - SDLC, [16](#)
- CI/CD. See [Continuous integration and deployment \(CI/CD\)](#)
- Classification
  - changes, [571–572](#)
  - datacenter data, [722](#)
  - problems, [511](#)
- Clean workstation state, [118](#)
- Clients
  - DHCP, [122](#), [717–718](#)
  - DNS, [67](#)
  - NTP, [72](#)
  - remote access, [91](#)
  - software repositories, [841–843](#)
  - thin, [105–106](#)
  - VDI, [109–112](#)
  - web browsers, [851](#)
- Clonezilla product, [143](#)
- Cloning
  - OS installation, [143–145](#)
  - production environments, [362–363](#)
- Clos fabrics, [411–412](#)
- Closed services, [298](#)
- Closer stereotype, [521](#)
- Closets for datacenters, [449](#)

Closing step in incident reports, [519](#)

Cloud-based services

- cost benefits, [239–241](#)

- description, [239](#)

- in programmable infrastructure, [56](#)

- server hardware strategies, [238–241](#)

- software as a service, [241](#)

- WANs, [415](#)

- workstations, [91](#)

Clusters

- server hardware, [245](#)

- storage-area networks, [764](#)

- VMs, [229](#), [233](#)

CM. *See* [Change management \(CM\) and changes](#); [Configuration management \(CM\)](#)

CMDB. *See* [Configuration management database \(CMDB\)](#)

CMM (Capability Maturity Model), [1023–1024](#)

CMSs (content-management systems)

- documentation, [560–561](#)

- web services, [866–867](#)

CNAME records, [314](#), [696](#)

Code of conduct, [877–878](#)

Code of ethics, [875–876](#)

Cold spares

- description, [259–260](#)

- masters and slaves pattern, [322](#)

Coleman, T. S., [387](#)

Collaboration in infrastructure as code, [72–73](#)

Colocation (colo) centers

- datacenters, [449](#)

- Internet service, [905](#)

Color-coding labels, [471–472](#)

Command-line captures, [554–555](#)

Command-line editing, [546](#)

Comments in wiki systems, [561](#)

## Communication

- active listening, [950–954](#)
- centralization, [651–652](#)
- change management, [576–577](#)
- email service, [744–745](#)
- flash-cut conversions, [382](#)
- high-availability sites, [634](#)
- I statements, [950](#)
- maintenance windows, [625–627](#), [630](#)
- negotiation, [954–960](#)
- onboarding, [208–209](#)
- organizational structures, [891](#)
- overview, [949–950](#)
- server upgrades, [606–607](#)
- service conversions, [378–379](#)
- service launches, [343](#)

## Community building in DevOps, [17](#)

## Community resource, data storage as, [765–766](#)

## Community strings in SNMP, [677–679](#)

## *Compassionate Geek, The*, [960](#)

## Compatibility

- server upgrades, [591–592](#)
- software repositories, [841–842](#)

## Competition in negotiations, [956](#)

## Completed items in Kanban, [8](#)

## Completeness, automated updates for, [93](#)

## Compliance

- cloud-based services, [241](#)
- monitoring, [689–690](#)
- repositories for, [828](#)

## Complicated processes, documenting, [552](#)

## Compliments, [970–971](#)

## Comprehensive system testing for maintenance windows, [628–630](#)

## Computer closets, [449](#)

## Computer rooms

- datacenters, [449](#)
  - servers in, [253–254](#)
- Computers, influx of, [1079](#)
- Conference bridges for maintenance windows, [625–627](#)
- Conferences
  - professional development, [968](#)
  - recruiting at, [983](#)
- Confidence building, [1084](#)
- Confidential data, [722](#)
- Configuration
  - DNS clients, [67](#)
  - states, [44](#)
  - workstations, [84–86](#)
  - workstations vs. servers, [249](#)
- Configuration management (CM)
  - declarative vs. imperative, [64–65](#)
  - guards and statements, [66–67](#)
  - idempotency, [65–66](#)
  - infrastructure as code, [63–67](#), [71–72](#)
  - networks, [435–436](#)
  - software repository caches, [837](#)
  - workstations, [120–121](#)
- Configuration management database (CMDB)
  - documentation, [439](#)
  - service checklists, [588–589](#)
  - upgrades, [185–186](#)
  - workstation fleet logistics, [183–186](#)
  - workstation service definition, [160](#)
- Configured workstation state, [118–120](#)
- Conflicts
  - changes, [583](#)
  - namespaces, [698–699](#)
  - software repositories, [843](#)
- Connections
  - centralization, [647–648](#)

- Internet, [420–422](#)
  - web services, [860–862](#)
- Consented, informed, [873–874](#)
- Conserver software, [256](#)
- Consistency
  - automation, [53](#), [74](#)
  - nameservices, [712–713](#)
  - namespaces, [704–707](#)
  - OS installation, [138–141](#)
- Console
  - datacenters, [465](#), [475–476](#)
  - maintenance windows, [625](#)
- Consultants, [906–907](#)
- Consumables for backups and restores, [809–814](#)
- Consumer view for cloud-based services, [239](#)
- Consumer workstation products, [104](#)
- Containers
  - Docker Container format, [48–51](#)
  - server hardware strategies, [234–235](#)
- Content-management systems (CMSs)
  - documentation, [560–561](#)
  - web services, [866–867](#)
- Content on web, protecting, [863](#)
- Content scanning in email service, [742](#)
- Contention in software repositories, [840](#)
- Contents in documentation templates, [553–554](#)
- Context switches, [268–269](#), [272](#)
- Contingency planning, [393](#)
- Continuity requirements for datacenters, [452–453](#)
- Continuous data protection (CDP), [787](#)
- Continuous Delivery*, [16](#), [356](#)
- Continuous delivery (CD) in SDLC, [16](#)
- Continuous improvement, [182](#)
- Continuous integration (CI)
  - build environments, [844](#)



- SDLC, [16](#)
- Continuous integration and deployment (CI/CD), [354](#)
  - description, [4](#)
  - infrastructure as code, [62](#), [72](#)
  - launch categorizations, [355–356](#)
  - test ordering, [355](#)
- Continuous launch category, [355–356](#)
- Contractors, [906–907](#)
- Controlled mirrors, [827](#), [847–848](#)
- Controllers
  - domain, [651](#), [664](#), [718](#)
  - drive, [761](#), [783](#)
  - network, [425](#)
  - RAID, [275–276](#)
- Conversions. See [Service conversions](#)
- Cookies, [324–325](#)
- Cooling computer rooms, [253](#)
- Coordination in service monitoring, [685](#)
- COOs (chief operating officers), [897–898](#)
- Copy Exact philosophy, [546](#)
- Copyrights, [878–881](#)
- Cords, [277](#)
- CoreOS operating system, [234](#)
- Cores, CPUs, [267–270](#)
- Corporate culture
  - helpdesks, [488](#)
  - workstation standardization, [195–196](#)
- Corporate standards and guidelines
  - backups and restores, [794](#), [799–800](#)
  - logical design, [423–424](#)
  - overview, [422–423](#)
  - physical design, [424–425](#)
- Corrections in wiki systems, [561](#)
- Corruption in web services, [864](#)
- Costs

- backups and restores, [794](#), [819](#)
- BYOD, [112](#)
- cloud-based services, [239–241](#)
- controlling, [894–895](#)
- data storage, [759](#)
- decentralization, [641](#)
- decreasing, [1083](#)
- email service failures, [732](#)
- infrastructure as code, [59–60](#)
- laptops vs. desktops, [102](#)
- printer supplies, [756](#)
- printers, [750](#)
- RAID, [276](#)
- reliability, [774–775](#)
- SSDs, [90](#)
- tape, [810–811](#)
- VDI, [106](#)
- WANs, [417](#)
- workstations, [103–104](#)
- Coverage in customer support, [499–500](#)
- Cowboy role, [1107–1108](#)
- CP. See [Capacity planning \(CP\)](#)
- CPUs
  - monitoring systems, [673](#)
  - performance, [271–273](#)
  - servers, [267–270](#)
  - software repositories, [833](#)
  - workstations vs. servers, [246](#)
- Craft workers for incident reports, [517–519](#)
- Crashes, dealing with, [1072–1073](#)
- Crawley, D., [961](#)
- CRBs. See [Change review boards \(CRBs\)](#)
- Crimper tools, [468](#)
- Criteria
  - change management plans, [573](#)

- LRR, [345–349](#)
  - staging areas, [342](#)
- Critical DNS server upgrades, [607–608](#)
- Critical inner voice, [970](#)
- Critical paths, computers in, [285](#)
- Criticism, accepting, [965](#)
- Cross-shipping servers, [261](#)
- Cross-team reviews, [182](#)
- Crying wolf role, [1107](#)
- CTOs (chief technology officers), [897–898](#)
- Cuckoo's Egg*, [539](#)
- Cultural diversity in team members, [987–988](#)
- Culture
  - helpdesks, [488](#)
  - workstation standardization, [195–196](#)
- Culture of respect in documentation, [561](#)
- Culture of trust in assessments, [1029](#)
- Currentness workstation issues, [79–80](#)
- Customer care and support
  - architecture, [525](#)
  - case study, [524](#)
  - customer familiarity, [522](#)
  - friendliness, [488](#)
  - holistic improvement, [522](#)
  - incident reports. *See* [Incident reports](#)
  - marketing-driven, [511](#)
  - model-based training, [521–522](#)
  - organizational structures, [902–904](#)
  - outage announcements, [522–523](#)
  - overview, [485](#)
  - primary SA duty, [899](#)
  - time coverage, [499–500](#)
  - trend analysis, [523–524](#)
- Customer-focused hours, [7](#)
- Customer/SA role, [1105–1106](#)

- Customer service requirements, [286](#)
  - difficult requests, [290–291](#)
  - features, [288–289](#)
  - questions to ask, [289](#)
  - SLAs, [290](#)
- Customer-to-attendant ratios, [489](#)
- Customer-to-SA ratios, [893](#)
- Customers
  - datacenter access, [455–456](#)
  - dependency checks, [590](#)
  - face saving, [512–513](#)
  - helpdesk wait time, [489](#)
  - pleasing, [1077](#)
  - training, [524–525](#)
  - usage guidelines, [875](#)
  - venting about, [919](#)
  - vs. users, [3](#)
  - workstation standardization involvement, [192–193](#)
- Customer's advocate role, [1102](#)
- Customizability, decentralization for, [641](#)
- Cut-off dates for workstation standardization, [195](#)
- Cycles, backups, [801–807](#)
- Cylinders in disks, [760–761](#)
  
- D2D2T (disk-to-disk-to-tape) backups, [808](#)
- DAD (disk access density), [786–787](#)
- Daily tasks, [941](#)
- Damage limitation, [390–391](#)
- DAP (Directory Access Protocol), [86–87](#)
- Darmohray, E. T., [982](#), [984](#)
- DAS (directly attached storage), [764](#)
- Dashboards for workstation fleet logistics, [180](#)
- Data cables, separating, [470–471](#)
- Data gathering in historical monitoring, [674–675](#)
- Data in nameservices, [711–712](#)

## Data integrity

- all eggs in one basket strategy, [223](#)

- overview, [392–393](#)

- RAID disks, [250–251](#)

## Data leakage prevention (DLP), [648](#)

Data link layer in the OSI model, [400–401](#)

Data management in centralization, [652](#)

Data migration, launch with, [366–369](#)

Data pipelines, [783–784](#)

Data privacy requirements, [453–454](#)

## Data storage

- backups and restores, [775–777](#), [812–814](#)

- as community resource, [765–766](#)

- continuous data protection, [787](#)

- disk access density gap, [786–787](#)

- disk components, [760–761](#)

- drive speed considerations, [785](#)

- evaluating, [784–787](#)

- fragmentation, [785–786](#)

- future needs, [770–771](#)

- historical monitoring, [675–676](#)

- infrastructure size, [788](#)

- inventory and spares policy, [769–770](#)

- managing, [765–772](#)

- mapping groups, [768–769](#)

- migration, [619–620](#)

- monitoring, [777–779](#)

- overview, [759](#)

- performance, [780–784](#)

- pipeline optimization, [783–784](#)

- problems, [787–789](#)

- RAID disks, [761–764](#)

- real-time monitoring data, [676](#)

- reliability, [773–775](#)

- saturation, [789](#)

- as a service, [772–780](#)
- SLAs, [773](#)
- standards, [771–772](#)
- storage-needs assessments, [766–768](#)
- terminology, [760–764](#)
- timeouts, [788](#)
- VMs, [230–231](#)
- workstation disposal, [132](#)
- workstations, [89–93](#)

Database-driven web sites, [852](#)

## Databases

- high-availability, [809](#)
- nameservices, [720–721](#)

## Datacenters

- centralization, [649](#)
- moving, [1069–1070](#)
- nonstandardized, [649](#)
- server hardware as cattle, [41–43](#)

## Datacenters networks

- case studies, [408–409](#), [413](#)
- logical design, [412–413](#)
- overview, [408](#)
- physical infrastructure, [409–412](#)

## Datacenters operations

- capacity management, [459–465](#)
- case study, [480](#)
- console access, [475–476](#)
- console ports, [465](#)
- device decommissioning, [467–468](#)
- installations, [465–466](#)
- labeling, [471–474](#)
- life-cycle management, [465–468](#)
- maintenance, [466–467](#)
- moves, adds, and changes, [466](#)
- networks, [465](#)

- overview, [459](#)
- parking spaces, [480](#)
- patch cables, [468–472](#)
- power, [462–464](#)
- rack space, [461–462](#)
- tools and supplies, [477–479](#)
- wiring, [464–465](#)
- workbenches, [476–477](#)

#### Datacenters overview, [449](#)

- building, [450](#)
- hybrid, [451–452](#)
- outsourcing, [451](#)
- renting, [450–451](#)
- requirements, [452–456](#)
- software as a service, [451](#)

#### Dataflow analysis for scaling, [328–330](#)

#### Dataless configurations, [91](#)

#### DBAN (Derek’s Boot and Nuke) utility, [132](#)

#### DDI (DNS, DHCP, and IPAM) nameservice, [707](#)

#### DDoS (distributed denial of service) attacks, [421–422](#)

#### Deadlines

- maintenance windows, [628](#)
- ready lists, [341](#)

#### Death squads in Google, [225–226](#)

#### Debconf product, [142](#)

#### Debois, P., [17](#)

#### Debugging

- bug classes, [533](#)
- causes vs. symptoms, [531](#)
- change management, [574](#)
- end-to-end understanding of systems, [538–539](#)
- overview, [529](#)
- problem understanding, [529–531](#)
- service planning and engineering, [318](#)
- systematic, [532–533](#)

tools, [533–537](#)

workstations, [118](#)

workstations vs. servers, [248](#)

## Decentralization

delegation, [642–643](#)

description, [639](#)

DNS, [651](#)

rationale for, [640–641](#)

## Decentralized models

description, [639](#)

funding, [895–896](#)

system administration, [903](#)

## Decentralized services

appropriate, [665](#)

managing, [666–667](#)

monitoring, [665](#)

## Decision points

change management plans, [573–574](#)

service conversions, [384–385](#)

## Decision, precompiling, [942–943](#)

## Declarative programming language, [64–65](#)

## Decommissioning

datacenter devices, [467–468](#)

in life cycle, [434–435](#)

workstations, [131–133](#)

## Decoupling hostname from service name, [313–315](#)

## Decoupling state, [46–47](#)

## Dedicated lines, [417](#)

## Defaults

power of, [43](#)

VMs, [42–43](#)

web pages, [925–926](#)

## Defined assessment level, [1023–1024](#)

## Defining names, [694–698](#)

## Definition files in infrastructure as code, [62](#)



- Definitions, [287–288](#)
- Delegation, upward, [974–975](#)
- Deletions
  - accidental, [546](#)
  - restores for, [796–797](#)
- Delivery checklists, [186](#)
- Delivery phase in SDLC, [15](#)
- Delivery team in workstation fleet logistics, [177–178](#)
- Demarcation points, [414](#)
- Demilitarized zones (DMZs), [412](#)
- Department-specific repositories, [828](#)
- Departmental approach for refresh cycles, [162](#)
- Dependencies
  - checks on, [590](#)
  - hard, [309–310](#)
  - onboarding, [204–205](#)
  - service monitoring, [685](#)
  - soft, [310](#)
- Dependency engineering, [309](#)
  - dependency alignment, [311–313](#)
  - external dependencies, [309–310](#)
  - primary dependencies, [309](#)
- Dependency hell
  - servers, [224](#)
  - virtualization for, [229](#)
- Dependency matrices, [309–310](#)
- Deploying device phase, [434](#)
- Deployment
  - automated updates, [93](#)
  - IPv6, [427–428](#)
  - primary SA duty, [899](#)
  - SDLC, [15](#)
  - software, [437](#)
- Depreciation in refresh cycles, [163–164](#)
- Depression, [1110–1111](#)

Derek's Boot and Nuke (DBAN) utility, [132](#)

Descriptive names, [695](#)

*Design and Implementation of the FreeBSD Operating System*, [534](#)

Designs

- primary SA duty, [899](#)

- redundancy design patterns, [322–326](#)

- service planning and engineering, [307–309](#)

- vendor-certified, [308–309](#)

Desired workstation state, [120](#)

Desktops

- as cattle, [40](#)

- vs. laptops, [101–102](#)

- for servers, [248](#)

- SSDs for, [90](#)

Dev staging areas, [342](#)

Devices

- decommissioning, [467–468](#)

- labeling, [439](#)

- life cycle, [433–435](#)

- monitoring, [432](#)

- wired office networks, [404](#)

DevOps

- change management, [575](#), [579](#)

- description, [16](#)

- principles, [16–17](#)

- service launch. *See* [Service launch DevOps](#)

- term, [15](#)

- Three Ways, [211](#)

*DevOps Handbook, The*, [17](#)

DHCP. *See* [Dynamic Host Configuration Protocol \(DHCP\) service](#)

Diagnostics for office networks, [441](#)

Diameters of namespaces, [701–702](#)

diff program

- server upgrades, [593](#)

- text file differences, [518–519](#)

- Differential backups, [805–807](#)
- Difficult requests, [290–291](#)
- Dijker archive, [875](#)
- Dijkstra, Edsger W., [67](#)
- Directly attached storage (DAS), [764](#)
- Directories, shared, [557](#)
- Directory Access Protocol (DAP), [86–87](#)
- Directory Server, [86](#)
- Directory traversal, [863](#)
- Disabling maintenance windows access, [621–622](#)
- Disagreements, [915–916](#)
- Disaster preparedness, [1060–1061](#)
- Disaster recovery (DR)
  - damage limitation, [390–391](#)
  - data integrity, [392–393](#)
  - legal obligations, [389–390](#)
  - media relations, [394](#)
  - operational requirements, [298](#)
  - overview, [387–388](#)
  - preparation, [391–392](#)
  - redundant sites, [393–394](#)
  - risk analysis, [388–389](#)
  - security disasters, [394](#)
- Disaster Recovery Planning*, [387](#)
- Disaster worrier role, [1100](#)
- Discussions in DevOps, [17](#)
- Dishwasher spots, [1085](#)
- Disk access density (DAD), [786–787](#)
- Disk-to-disk-to-tape (D2D2T) backups, [808](#)
- Diskless storage, [89](#)
- Disks
  - backups, [820–821](#)
  - boot, [306](#)
  - capacity, [766](#)
  - drive speed considerations, [785](#)

- failures, [796–797](#)
- historical data, [675](#)
- monitoring, [778](#)
- RAID, [275–276](#)
- spinny components, [760–761](#)
- SSDs, [761](#)
- swap space, [273–274](#)
- workstation disposal, [132](#)

Disliked processes, documenting, [552](#)

Dispatchers, [854](#)

Disposal

- devices, [435](#)
- workstations, [130–134](#)

Disqus servers, [47](#)

Disruption in server upgrades, [597](#)

Distributed denial of service (DDoS) attacks, [421–422](#)

Distributed models, [639](#)

Distributed parity in RAID disks, [762–763](#)

Distribution systems vs. source packages, [834](#)

Diversity of team members, [987–988](#)

DLP (data leakage prevention), [648](#)

DMZs (demilitarized zones), [412](#)

DNS. *See* [Domain Name Service \(DNS\)](#)

DNS, DHCP, and IPAM (DDI) nameservice, [707](#)

Docker compose files, [48–49](#)

Docker Container format, [48–51](#)

Dockerfiles, [49](#)

Document retention policy for email, [744](#)

Documentation

- case study, [554](#)
- change management plans, [573](#)
- checklists, [556–557](#)
- content-management systems, [560–561](#)
- culture of respect, [561](#)
- email service, [744–745](#)

- findability, [559](#)
- infrastructure as code, [62](#)
- network operations, [437–440](#)
- off-site links, [562–563](#)
- overview, [551–552](#)
- policies, [1075–1076](#)
- print service, [754–755](#)
- purpose, [1075](#)
- roll-out issues, [560](#)
- selecting items for, [552–553](#)
- service planning and engineering, [318](#)
- sources, [554–556](#)
- taxonomy and structure, [561–562](#)
- templates, [553–554](#)
- uses, [562](#)
- wiki systems, [557–559](#)

Doer of repetitive tasks role, [1111](#)

Domain controllers, [651](#), [664](#), [718](#)

Domain Name Service (DNS)

- anycast, [715](#)
- authority, [715–717](#)
- centralization, [650–651](#), [664](#)
- client configuration, [67](#)
- CNAME records, [314](#)
- documentation, [439](#)
- dynamic updates, [716–717](#)
- email service, [735](#)
- infrastructure software, [363–366](#)
- nameservices, [714–717](#)
- root, [715](#)
- server upgrades, [607–608](#)

Doohan, James, [601](#)

Doran, George T., [338–339](#)

Dormant installation of packages, [830](#)

Double parity in RAID disks, [763](#)

- DR. See [Disaster recovery \(DR\)](#)
- Drawings with overlays, [438](#)
- Dressing, [915](#)
- Drive controllers, [761](#)
- Dual-boot configurations, [125](#)
- Dual star topology, [414](#)
- Dumb pipelining algorithms, [783](#)
- Dumpster diving, [884](#)
- Dunning, D., [998](#)
- Dunning–Kruger effect, [998](#)
- Duplex printing, [757](#)
- Dynamic backup schedules, [805](#)
- Dynamic configuration of workstations, [84–85](#)
- Dynamic DNS updates, [716](#)
- Dynamic Host Configuration Protocol (DHCP) service centralizing servers, [664](#)
  - importance, [86](#)
  - infrastructure software, [363–366](#)
  - IP addresses, [315](#)
  - nameservices, [717–718](#)
  - OS installation, [142–143](#)
  - workstations, [84–86](#), [122–123](#)
- E-commerce sites
  - helpdesks, [489](#)
  - organizational structures, [909](#)
  - privacy, [886](#)
- EAPs (employee assistance programs), [964](#), [1110–1111](#)
- Early-access staging areas, [342](#)
- Early value in small batches principle, [32](#)
- EBGP (Exterior Border Gateway Protocol), [420](#)
- Educator role, [1098–1099](#)
- Effort in server upgrades, [597–598](#)
- 8-mm tapes, [821](#)
- Einstein, Albert, [937](#)
- Electronic accomplishment walls, [972](#)

- Elements, workstation, [82](#)
- Elimination process in debugging, [532](#)
- Email service and email
  - automation, [737](#)
  - case studies, [732](#), [734](#)
  - communication, [744–745](#)
  - for documentation, [555](#)
  - encrypting, [743](#)
  - forwarding, [886](#)
  - generality, [736](#)
  - handling, [940–941](#)
  - high-volume list processing, [745–746](#)
  - migrating, [368](#)
  - monitoring, [738](#)
  - name conflicts, [699](#)
  - namespaces, [730–731](#)
  - nonstandard protocols, [736–737](#)
  - overview, [729](#)
  - privacy policy, [730](#)
  - proprietary, [300–302](#)
  - redundancy, [738–739](#)
  - reliability, [731–732](#)
  - retention policy, [743–744](#)
  - scaling, [739–742](#)
  - security, [742](#)
  - simplicity, [733–735](#)
  - spam and virus blocking, [735](#)
  - visibility, [930–932](#)
- Emergencies, defining, [8](#), [495–496](#)
- Emergency facility preparation, [392](#)
- Emergency failovers, improving, [26–29](#)
- Emergency Power Off (EPO) procedures, [624](#)
- Emergency response (ER)
  - operational assessments, [1039–1040](#)
  - operational responsibilities, [1021](#)

- Emergency services in wired office networks, [405–406](#)
- Emergency updates, [571](#)
- Emory University, [130](#)
- Employee assistance programs (EAPs), [964](#), [1110–1111](#)
- Employees
  - new hires, [180](#), [916–917](#)
  - onboarding. *See* [Onboarding](#)
  - retention, [1000–1001](#)
- Encryption
  - email service, [743](#)
  - web services, [860–861](#)
  - workstations, [95](#)
- End-to-end expert role, [1109](#)
- End-to-end monitoring tests, [687–688](#)
- End-to-end security vs. transport, [835](#)
- End-to-end understanding of systems, [538–539](#)
- End-user disruption in server upgrades, [597](#)
- End-user support in centralization, [655](#)
- Engineering, service. *See* [Service planning and engineering](#)
- Enhanced collaboration in infrastructure as code, [72–73](#)
- Enterprise products
  - servers, [245](#)
  - workstations, [104–105](#)
- Entropy process in workstations, [118](#)
- Entry criteria for staging areas, [342](#)
- Environmental issues
  - identifying, [1076](#)
  - print service, [756–757](#)
  - service fit, [295–296](#)
- EPO (emergency power off) procedures, [624](#)
- Equipment lists, [424–425](#)
- ER (emergency response)
  - operational assessments, [1039–1040](#)
  - operational responsibilities, [1021](#)
- Erasing disks, [132](#)



Escalation of alerts, [682](#)

Escalation process in customer support, [494–495](#)

## Ethics

- code of ethics, [875–876](#)

- copyrights, [878–881](#)

- customer usage guidelines, [875](#)

- informed consent, [873–874](#)

- law enforcement interaction, [881–885](#)

- observing illegal activity, [888–889](#)

- overview, [873](#)

- privacy and monitoring, [885–886](#)

- privileged-access codes of conduct, [877–878](#)

- unethical orders, [887–888](#), [1085](#)

Ethnic diversity in team members, [987–988](#)

## Etsy, Inc.

- launches, [356](#)

- onboarding, [202–203](#)

Evaluation copies of tools, [537](#)

Evans, Paul, [611](#)

Evard, Rémy, [117](#), [118](#)

Events, logging, [97–98](#)

Exceptions, centralization with, [643](#)

Execution stage in maintenance windows, [612](#)

Executioner stereotype, [520](#)

Executive summaries in service launch lessons learned, [344](#)

Exit checklists, [1007](#), [1014–1015](#)

Exit criteria for staging areas, [342](#)

## Expansion

- laptops vs. desktops, [101–102](#)

- servers, [223](#)

- workstations vs. servers, [246](#)

## Expectations

- aligning priorities with, [920–921](#)

- managers, [1088](#)

- privacy and monitoring, [885–886](#)

- system administrators, [1087–1088](#)
- Experimentation
  - DevOps, [17](#)
  - small batches principle, [33](#)
- Expiration dates in DHCP, [717](#)
- Expiring historical monitoring data, [675](#)
- Exterior Border Gateway Protocol (EBGP), [420](#)
- Exterior gateway protocols, [420](#)
- External dependencies, [309–310](#)
- External service access for fired system administrators, [1010](#)
- Extranet connections in centralization, [647–648](#)
- Fabric pods, [411–412](#)
- Facilitator role, [1105](#)
- FAI (fully automatic installation) product, [142](#)
- Fail early and often strategy, [30](#)
- Fail-safe testing, [62](#)
- Failover process, improving, [26–29](#)
- Failure domains, [311–313](#)
- Failures
  - data storage, [773–775](#)
  - disaster recovery, [298](#)
  - email service, [732](#)
  - restores for, [796–797](#)
  - service launch lessons learned, [344](#)
  - service launches, [349–350](#)
- Fall, K. R., [536](#)
- Family time, [967](#)
- Farley, D., [16](#), [356](#)
- FC (fiber channel), [782](#)
- FCC (Federal Communications Commission), [878](#)
- FDE (full disk encryption), [95](#)
- Features
  - adding, [1083](#)
  - customer service requirements, [288–289](#)

- decentralization for, [641](#)
- MVP, [358](#)
- server hardware. *See* [Server hardware features](#)
- Federal Communications Commission (FCC), [878](#)
- Federated identity, [708–709](#)
- Federation setup, [841](#)
- Feedback
  - DevOps, [211](#)
  - launches, [30–31](#)
  - obtaining, [929](#)
  - organizational assessment, [1029](#)
  - town hall meetings, [928](#)
  - workstation standardization, [192–193](#)
- Feeling Good Handbook*, [964](#), [976](#)
- Fiber channel (FC), [782](#)
- File locks for workstations, [92](#)
- File service operations, monitoring, [779](#)
- File systems for RAID disks, [763](#)
- Filtering email service, [733](#)
- Findability of documentation, [559](#)
- Fire drills, [798–799](#)
- Firewalls, [96](#)
- Firing system administrators
  - academic institutions, [1013–1014](#)
  - access removal, [1007–1011](#)
  - bosses, [1012–1013](#)
  - examples, [1011–1014](#)
  - exit checklists, [1007](#)
  - HR cooperation, [1006](#)
  - logistics, [1011](#)
  - overview, [1005–1006](#)
  - supporting infrastructure, [1014–1015](#)
- Firmware, [437](#)
- First impressions
  - making, [914–917](#)

- onboarding, [201–202](#)
- First offers in negotiation, [959](#)
- First support tier for workstations, [167](#)
- First Way in DevOps, [211](#)
- Fisher, M., [322](#)
- Fix it once principle
  - automation, [547–549](#)
  - carpenter principle, [545–547](#)
  - case studies, [544–545](#), [547–549](#)
  - misconfigured servers story, [541–543](#)
  - overview, [541](#)
  - temporary fixes, [543–545](#)
- Fixed scope in service requirements, [287](#)
- Fixes for web content, [867](#)
- Flap detection in real-time monitoring, [676–677](#)
- Flash-cuts, [380–382](#)
- Flash memory, [90](#)
- Flexibility
  - customer support, [513](#)
  - datacenter requirements, [454](#)
  - real-time monitoring, [677](#)
- Flight directors
  - description, [611](#)
  - high-availability sites, [634–635](#)
  - maintenance window tests, [629](#)
  - mentoring, [632](#)
  - selecting, [616–617](#)
- Floor plans, [403](#)
- Florida Election Board web site, [859](#)
- Focus, [936](#)
- FOG (Free and Open Ghost) product, [143](#)
- Follow-the-path debugging, [532](#)
- Follow-through, [937](#)
- Follow-up questions in interview process, [995–997](#)
- Footprints, memory, [271](#)

Forbidden support tier for workstations, [167](#)  
Forecasting and ordering process, [180](#)  
Forklift upgrades, [223](#)  
Forklifts in datacenters, [478](#)  
Form-field corruption, [864](#)  
Formal training as debugging tool, [534](#)  
Format for real-time monitoring messages, [681](#)  
Formatting wiki systems, [558](#)  
Forms for change proposals, [570](#)  
Formulaic names, [695–696](#)  
Forwarding email, [886](#)  
Foundational services, [284](#)  
Fowler, Martin, [39](#)  
Fragmentation, [785–786](#)  
Free and Open Ghost (FOG) product, [143](#)  
Free storage space, monitoring, [778](#)  
Free time, [943–944](#)  
FreeBSD community, [983](#)  
Freeze periods for change, [453](#), [579–580](#)  
Frequent machine crashes, [1072–1073](#)  
Frequently asked questions, [562](#)  
Friendliness at helpdesks, [488](#)  
Friendships, start-day, [215](#)  
Front access to racks, [247](#)  
Frontload installation questions (FAQs), [149](#)  
Full backups, [794](#), [801–804](#)  
Full disk encryption (FDE), [95](#)  
Full-time fleet coordinators, [187](#)  
Fully Automatic Installation (FAI) product, [142](#)  
Fun stuff, [1001](#)  
Functional names, [695–697](#)  
Functional zones, [423](#)  
Functionality skew case study, [689](#)  
Funding models, [894–897](#)  
Fungibility of workstations, [80–82](#)

- Future storage needs, planning for, [770–771](#)
- Fuzzy match algorithms, [593](#)
- Gaps in service requirements, [287](#)
- Gates for staging areas, [342](#)
- Gateways for email service, [734](#)
- Gelernter, David, [37](#)
- General printer architecture policies, [751](#)
- General repositories, [845–846](#)
- Generalists, [899](#)
- Generality in email service, [736](#)
- Generational approach for refresh cycles, [161](#)
- Generators, [463](#)
- Generic processes, [47–51](#)
- GET command in SNMP, [677](#)
- Getting noticed, [1001–1002](#)
- GFS distributed storage systems, [251](#)
- Ghost product, [143](#)
- Gibson, William, [1020](#)
- Git VCS
  - change tracking, [57](#)
  - starting with, [583–584](#)
- Global operations, [1068](#)
- Global optimization in server strategies, [226–227](#)
- Go-to person role, [1092](#)
- Goals
  - setting, [939–940](#)
  - S.M.A.R.T., [338](#)
- Golden hosts and images in cloning, [143–144](#)
- Good first impressions, [914–917](#)
- Goodman, P., [961](#)
- Google
  - approval committees, [356–357](#)
  - backups, [652](#)
  - bug priorities, [338](#)

- Chubby lock service, [29](#)
- grid computing, [235](#)
- hardware maintenance, [467](#)
- hardware types, [41–42](#)
- IT teams, [909](#)
- launch early and often strategy, [30](#)
- Mac OS installation, [121](#)
- mass email, [932](#)
- onboarding process, [202](#), [215–216](#)
- printer maps, [755](#)
- production readiness reviews, [345](#)
- self-service store, [160](#)
- server costs, [278](#)
- server upgrades, [380](#)
- software repositories, [835](#)
- SysOps death squads, [225–226](#)
- Google Cloud Platform
  - autoscale feature, [39](#)
  - in programmable infrastructure, [56](#)
- Google Compute Engine cloud-based service, [238](#)
- Goulah, John, [203](#)
- GPOs (group policy objects), [121–122](#)
- Graceful degradation, [310](#)
- Gradual roll-outs, [379–380](#)
- Granular security, [229](#)
- Granularity of historical monitoring data, [675](#)
- Greatness, measuring. *See* [Assessments](#)
- Green zone servers, [412](#)
- Greeters for incident reports, [508–509](#)
- Grid computing, [235–236](#)
- Group policy objects (GPOs), [121–122](#)
- Groups
  - joining, [1081](#)
  - managers, [1081](#)
  - mapping onto storage infrastructure, [768–769](#)

- Growing small sites, [1068](#)
- Growth, onboarding, [212](#)
- Growth opportunities, [989](#)
- “Guarded Commands, Nondeterminacy and Formal Derivation of Programs,” [67](#)
- Guards in configuration management systems, [66–67](#)
- Guest restrictions, [407–408](#)
- Guth, R., [568](#)
- Gwaltney, W., [362](#)
  
- Habits for happiness, [963–964](#)
- Haggler’s Handbook*, [961](#)
- Hand-trucks, [478](#)
- Handheld radios for maintenance windows, [626](#)
- Handles in namespaces, [703](#)
- Handoffs in customer care, [522](#)
- Happiness, [963–964](#)
- Happy system administrators
  - balancing work and personal life, [966–967](#)
  - case study, [974](#)
  - criticism, [965](#)
  - happiness, [963–964](#)
  - loving jobs, [969](#)
  - managing managers, [972–976](#)
  - motivated, [970–972](#)
  - overview, [963](#)
  - professional development, [967–968](#)
  - self-help books, [976](#)
  - staying technical, [968–969](#)
  - support structure, [965–966](#)
- Hard dependencies, [310](#)
- Hardcoded workstation configuration, [85](#)
- Hardware
  - as cattle, [41–43](#)
  - failure domains, [311](#)
  - life-cycle automation, [467](#)



server. See [Server hardware features](#); [Server hardware specifications](#); [Server hardware strategies](#)

workstations. See [Workstation hardware strategies](#)

Hardware RAID, [275–276](#)

Harris, Trey, [67](#), [539](#)

HBAs (host bus adapters), [761](#)

HDFS distributed storage systems, [251](#)

Heads, disk, [760–761](#)

Heartbleed bug, [294](#)

Heat in datacenters, [477](#)

Hell Month, [24–26](#)

Help, asking for, [966](#)

Helpdesker role, [1106](#)

Helpdesks

- advertising, [500–501](#)

- case studies, [489–492](#), [496–497](#)

- centralization, [654–655](#)

- corporate culture, [488](#)

- emergencies, [495–496](#)

- escalation process, [494–495](#)

- friendliness, [488](#)

- multiple, [501–502](#)

- NOCs, [444](#)

- obtaining help, [493](#)

- organizational structures, [904](#)

- overview, [485–487](#)

- request-tracking software, [496–498](#)

- scope, [490–493](#)

- service planning and engineering, [318](#)

- staffs, [488–489](#)

- statistical improvements from, [498–499](#)

- ticket systems, [6–8](#)

Hermetic builds, [844–845](#)

Hero of the week in Kanban, [10](#)

Hero role, [1092](#)

- Hidden infrastructure, [630](#)
- High-availability
  - databases, [809](#)
  - sites, [634–636](#)
  - workstations vs. servers, [247](#)
- High consistency in namespaces, [704](#)
- High risk change requests, [572](#)
- High-volume list processing, [745–746](#)
- Hijacked sites, [859](#)
- Hire the person strategy, [981](#)
- Hiring date, refresh cycles based on, [161–162](#)
- Hiring system administrators
  - case study, [981](#)
  - employee retention, [1000–1001](#)
  - getting noticed, [1001–1002](#)
  - interview process, [991–994](#)
  - interview team, [990–991](#)
  - job descriptions, [980–982](#)
  - nontechnical interviewing, [998–999](#)
  - overview, [979–980](#)
  - quick process, [1082](#)
  - recruiting, [983–984](#)
  - selling the position, [1000](#)
  - skill levels, [982](#)
  - team considerations, [987–989](#)
  - technical interviewing, [994–998](#)
  - timing, [985–986](#)
- Hiring System Administrators* booklet, [987](#)
- Historical monitoring, [672](#)
  - data gathering, [674–675](#)
  - data storage, [675](#)
  - data viewing, [675–676](#)
  - overview, [674](#)
  - trends, [632–633](#)
- history command for documentation, [555](#)

- History layer in content-management systems, [560](#)
- Hit-and-Run stereotype, [520](#)
- Holistic improvement for customer care, [522](#)
- Horizontal scaling, [856–857](#)
- Host bus adapters (HBAs), [761](#)
- Hostnames, decoupling from service names, [313–315](#)
- Hot-plug components vs. hot-swap, [252–253](#)
- Hot spares
  - masters and slaves pattern, [322](#)
  - RAID disks, [763](#)
- Hot-swap components, [252–253](#)
- How-to (HOWTO) documents, [562](#)
- How to Get Control of Your Time and Your Life*, [945](#)
- HR departments
  - firing system administrators, [1006](#)
  - recruiting by, [983–984](#)

## HTTP

- cookies, [324–325](#)
- redirects, [837](#)
- web servers, [853](#)

## HTTPS

- overview, [854](#)
- web servers, [861](#)
- Hub-and-spoke model, [98](#)
- Hugo blog site generator, [47](#)
- Humble, J., [16–17](#), [356](#)
- Hybrid centralization strategy, [642–643](#)
- Hybrid cloning and automation, [144](#)
- Hybrid datacenters, [451–452](#)
- Hybrid models
  - server upgrades, [596](#)

- system administration, [904](#)
- Hybrid names, [696](#)
- Hybrid server strategies, [242–243](#)
- Hybrid workstation configuration, [85](#)
- I/O
  - local usage monitoring, [778](#)
  - workstations vs. servers, [246](#)
- I/O operations per second per gigabyte of capacity (OPS/second/GB), [786](#)
- I statements, [950](#)
- IaaS (infrastructure as a service)
  - access management, [1008](#), [1010](#)
  - datacenters, [451](#), [456](#)
- IaC. See [Infrastructure as code \(IaC\)](#)
- IBM, Olympics failure and success, [568](#)
- Idempotency in configuration management systems, [65–66](#)
- Identity
  - users, [705–706](#)
  - workstations, [87](#)
- IDFs (intermediate distribution frames), [402–403](#)
- Illegal activity, observing, [888–889](#)
- Illegal orders, [887–888](#)
- iLO (Integrated Lights-Out), [247](#), [255](#)
- Images, cloning, [143–144](#)
- IMAP4 protocol, [741](#)
- Impact ratings of changes, [572](#)
- Imperative programming languages, [64–65](#)
- Implementation design in corporate standards, [425](#)
- Implementation plans in server upgrades, [598–600](#)
- Improvement
  - levels, [1030–1031](#)
  - onboarding, [209–211](#)
- In-place server upgrades, [595](#)
- In-rack UPS units, [464](#)
- In stock device phase, [434](#)

Inbound Internet connectivity, [421–422](#)

Incident reports

- case study, [516](#)

- closing step, [519](#)

- customer care, [521–525](#)

- greeting phase, [508–509](#)

- overview, [505–506](#)

- planning and execution phase, [515–518](#)

- problem identification phase, [509–515](#)

- process overview, [506–508](#)

- SA stereotypes, [520–521](#)

- solution execution, [517–518](#)

- solution proposals, [515–516](#)

- solution selection, [516–517](#)

- verification phase, [518–519](#)

Incremental backups, [794](#), [801–807](#)

Incremental development, [150](#)

Independence, VMs for, [228](#)

Index machines, [41–42](#)

Ineffective people, [944](#)

Influx

- computers, [1079](#)

- SAs, [1079–1080](#)

- users, [1079](#)

Informal off-site storage, [813](#)

Informed consent, [873–874](#)

Infrastructure

- centralization, [648–656](#)

- for firing system administrators, [1014–1015](#)

- hidden, [630](#)

- teams, [900–902](#)

Infrastructure as code (IaC)

- automation myths, [74](#)

- benefits, [59–62](#)

- configuration management tools, [63–67](#)

- DevOps, [17](#)
- downsides, [73](#)
- enhanced collaboration, [72–73](#)
- example systems, [67–71](#)
- overview, [55–56](#)
- programmable infrastructure, [56](#)
- starting with, [71–72](#)
- tracking changes, [57–59](#)
- Infrastructure builder role, [1093](#)
- Initial assessment level, [1023](#)
- Inner loops in release process, [26](#)
- Input in web services, [865](#)
- Install rooms, [125](#)
- Installation
  - datacenters, [465–466](#)
  - OS. See [Operating system \(OS\) installation](#)
  - packages, [830](#)
  - software, [922–923](#)
  - vs. updating, [123–124](#)
  - workstations, [119](#), [186](#)
- Installed device phase, [434](#)
- Installer role, [1090](#)
- Instant gratification in small batches principle, [33](#)
- Instant roll-back, [384](#)
- Institutional memory, documentation for, [551](#)
- Integrated Lights-Out (iLO), [247](#), [255](#)
- Integrated out-of-band management, [254–255](#)
- Integration phase in SDLC, [15](#)
- Integration testing in infrastructure as code, [60](#)
- Integrity, data, [250–251](#), [392–393](#)
- Intelligent Platform Management Interface (IPMI), [247](#), [255](#)
- Interior routing protocols, [419–420](#)
- Intermediate distribution frames (IDFs), [402–403](#)
- Intermodal shipping, [51](#)
- Internal data, [722](#)

Internal group-specific documents, [562](#)

Internet, recruiting on, [983](#)

Internet access

- inbound connectivity, [421–422](#)

- outbound connectivity, [420–421](#)

- overview, [420](#)

Internet-based backup systems, [815](#)

Interrupt sponge role in Kanban, [10](#)

Interruptions, [935–936](#)

Interviews in hiring system administrators

- interview team, [990–991](#)

- process, [991–994](#)

- technical, [994–998](#)

“Introducing Data Center Fabric, the Next-Generation Facebook Data Center Network,” [412](#)

Intrusiveness in service conversions, [374](#)

Inventory

- data storage, [769–770](#)

- datacenters, [460](#), [464–466](#)

- service checklists, [588–589](#)

- tapes, [811–812](#)

Invested customers, [590](#), [924](#)

Ionescu, A., [535](#)

IP address management (IPAM), [650](#)

IP addresses

- aliases, [314–315](#)

- corporate standards, [423](#), [425](#)

- IPv6, [426–428](#)

IP-KVM switches, [255–256](#)

iPad variations, [40](#)

IPAM (IP address management), [650](#)

IPMI (Intelligent Platform Management Interface), [247](#), [255](#)

IPv6

- deploying, [427–428](#)

- need for, [426](#)

- overview, [426](#)
- Irreproducible configuration states, [44](#)
- iSCSI protocol, [782](#)
- Isolated storage
  - SANs, [764](#)
  - web services, [861](#)
- Isolating states, [44–46](#)
- Isolation, VMs for, [228](#)
- Issue discussions in DevOps, [17](#)
- IT professionals view for cloud-based services, [239](#)
- IT responsibilities for onboarding, [203](#)
- Janitor role, [1108–1109](#)
- Jekyll blog site generator, [47](#)
- Job descriptions
  - documenting, [552–553](#)
  - system administrators, [980–982](#)
- Job Descriptions for System Administrators*, [982](#), [984](#)
- Jobs
  - looking for, [1081–1082](#)
  - protecting, [1085–1086](#)
- Johnson, S., [961](#)
- Jokes, [19](#), [40](#), [46](#), [52](#), [59](#), [64](#), [119](#), [142](#), [214](#), [288](#), [294](#), [300](#), [401](#), [529](#), [587](#), [817](#), [1085](#)
- Journaling file systems, [763](#)
- JSON format, [675](#)
- Jukeboxes, [819](#), [821](#)
- JumpStart product, [142](#)
- “Jupiter Rising,” [412](#)
- Just-in-time manufacturing, [23](#)
- KACE product, [142](#)
- Kanban
  - overview, [8–12](#)
  - ready lists, [339–340](#)
- Kanban: Successful Evolutionary Change for Your Technology Business*, [11](#)



- KDCs (key distribution center), [719](#)
- Kerberos authentication system
  - ActiveDirectory, [300](#)
  - authentication servers, [719](#)
- Kernighan, Brian, [307](#), [593](#)
- Key distribution center (KDCs), [719](#)
- Key repositories for web services, [861](#)
- Keyboard, video screen, and mouse (KVM) switches
  - console, [475–476](#)
  - maintenance windows, [625](#)
  - remote console with, [255–256](#)
- Keys, encryption, [743](#)
- Kick-off meetings, [285–286](#)
- KickStart product, [142](#)
- Kilmartin, Paul, [779–780](#), [785](#)
- Kim, Gene, [16](#), [17](#)
- Klein, Daniel V., [356–357](#)
- Knowledge base for documentation, [556](#)
- Known good state for workstations, [117](#)
- Koren, L., [961](#)
- Kruger, J., [998](#)
- KVM (keyboard, video screen, and mouse) switches
  - console, [475–476](#)
  - maintenance windows, [625](#)
  - remote console with, [255–256](#)
- L0-L3 support, [444](#)
- L1-L4 caches, [271–272](#)
- Lab environments in centralization, [656](#)
- Lab technician role, [1095–1096](#)
- Labeling
  - datacenters, [471–474](#)
  - devices, [439–440](#)
  - printer, [755](#)
- Lakein, A., [945](#)

LAMP (Linux, Apache, MySQL, and Perl) platforms, [853](#)

LAN (local area network). See [Wired office networks](#)

Laptops

- vs. desktops, [101–102](#)

- distribution variation, [52](#)

- irreproducible, [44](#)

- SSDs for, [90](#)

Large companies, [908](#)

Latency

- vs. bandwidth, [330–332](#)

- CPUs, [271–272](#)

- dedicated lines, [417](#)

- example problem, [536–537](#)

- packet size, [332–333](#)

Launch readiness criteria (LRC), [345–349](#)

Launch readiness review (LRR), [345](#)

- criteria, [345–349](#)

- organizational learning, [347](#)

Launches

- CI/CD categorizations, [355–356](#)

- DevOps. See [Service launch DevOps](#)

- generic processes for, [48](#)

- small batches principle, [29–33](#)

Law enforcement, working with, [881–885](#)

Layers vs. pillars approaches in service conversions, [376–377](#)

Lazy path server strategy, [225](#)

LDAP. See [Lightweight Directory Access Protocol \(LDAP\)](#)

Leakage prevention in centralization, [648](#)

Leaky exceptions, centralization with, [643](#)

Lean manufacturing movement, [23](#)

LeanKit system

- Kanban, [11](#)

- ready lists, [339](#)

Learning mode for workstations, [95](#)

Leased workstation model, [169–170](#)

Leases in DHCP, [717–718](#)

Leber, J., [794](#)

LeFebvre, W., [987](#)

Legacy systems, [363](#)

Legal department

- backups, [800](#)

- law enforcement contacts, [882](#)

- repositories for, [828](#)

Legal obligations in disaster recovery, [389–390](#)

Lessons learned in service launch process, [343–345](#)

Level-focused person role, [1109–1110](#)

Levels

- assessment, [1023–1024](#)

- caches, [271–272](#)

- capacity planning, [1043–1044](#)

- change management, [1045–1046](#)

- disaster preparedness, [1060–1061](#)

- emergency response, [1039–1040](#)

- improvement, [1030–1031](#)

- monitoring and metrics, [1041–1042](#)

- new product introduction and removal, [1048](#)

- performance and efficiency, [1051–1053](#)

- print service centralization, [750](#)

- RAID disks, [761–763](#)

- regular task assessments, [1036–1038](#)

- reliability, [250](#)

- service delivery build phase, [1054–1055](#)

- service delivery deployment phase, [1056–1057](#)

- service deployment and decommissioning, [1049–1050](#)

- support, [444](#)

- toil reduction, [1058–1059](#)

Library packages, [831](#)

License management

- simplifying, [880](#)

- software installation, [922–923](#)

## Life cycle

- CMDB for, [184](#)

- datacenters, [465–468](#)

- devices, [433–435](#)

- namespaces, [699–700](#)

- networks, [435](#)

- workstation software. See [Workstation software life cycle](#)

Lights-out management (LOM), [255](#), [625](#)

Lightweight Directory Access Protocol (LDAP) authentication servers, [719](#)

- authorization, [86](#)

- complexity of, [87](#)

- nameservices, [718](#)

Limited availability to maintenance windows, [633–634](#)

Limoncelli, T. A., [182](#), [376](#), [426–427](#), [945](#), [1005](#)

Line-of-sight radio communications, [626–627](#)

Link shorteners, [559](#)

Links, monitoring, [432](#)

Linux, Apache, MySQL, and Perl (LAMP) platforms, [853](#)

Linux operating systems, [83](#)

Lions, J., [534](#)

*Lions' Commentary on UNIX*, [534](#)

List attributes for ready lists, [338–339](#)

List processing

- email service, [733](#)

- high-volume, [745–746](#)

Listening, active, [950–954](#)

Live migration, [230](#)

Load balancers

- email service, [739](#)

- web services, [856–857](#)

Load balancers plus replicas, [323](#)

Load sharing configuration, [323](#)

Load testing

- infrastructure as code, [60–61](#)

- performance, [327](#)

- Local account databases for workstations, [86](#)
- Local engineering design, [308](#)
- Local engineering plans, [305](#)
- Local mirrors, [827](#)
- Local storage for workstations, [89–90](#)
- Locality issues for workstations, [79–80](#)
- Location failure domains, [312–313](#)
- Locking out customers in server upgrades, [604](#)
- Locks for workstations, [92](#)
- Logical design
  - corporate standards, [423–424](#)
  - datacenter networks, [412–413](#)
  - wired office networks, [403–404](#)
  - wireless office networks, [406–408](#)
- Logical networks, [399](#)
- Logical unit numbers (LUNs), [764](#)
- Login name conflicts, [699](#)
- Logistics in firing system administrators, [1011](#)
- Logs and logging
  - CDP, [787](#)
  - centralization, [653–654](#)
  - configuration changes, [436](#)
  - email service, [738](#)
  - helpdesks, [491](#)
  - law enforcement contacts, [882](#)
  - real-time monitoring, [679](#)
  - server upgrades, [589](#)
  - unethical orders, [887–888](#)
  - visibility, [292–293](#)
  - web services, [866](#)
  - workstations, [97–98](#)
- LOM (lights-out management), [255](#), [625](#)
- Long-term changes in service launch lessons learned, [344](#)
- Look-for's in assessment, [1025](#)
- Loops, release process, [26](#)

- Loura, Ralph, [182](#), [376](#), [920](#)
- Low consistency in namespaces, [704](#)
- Low risk change requests, [572–573](#)
- LRC (launch readiness criteria), [345–349](#)
- LRR (launch readiness review), [345](#)
  - criteria, [345–349](#)
  - organizational learning, [347](#)
- Lumeta
  - computer support, [486](#)
  - maintenance windows, [613–614](#)
  - onboarding process, [213–214](#)
- Lunch with customers, [932](#)
- LUNs (logical unit numbers), [764](#)
- MAC addresses for NACs, [405](#)
- Machine crashes, [1072–1073](#)
- MacKenzie, R. A., [945](#)
- Mail delivery agents (MDAs), [733](#)
- Mail eXchanger (MX) records, [739](#)
- Mail relay hosts, [742](#)
- Mail transport agents (MTAs), [733](#)
- Mail user agents (MUAs), [733](#)
- Mailping case study, [687](#)
- Main distribution frames (MDFs), [402–403](#)
- Maintainer role, [1091](#)
- Maintenance
  - datacenters, [466–467](#)
  - LRC, [347–349](#)
  - primary SA duty, [899](#)
  - spare capacity for, [232–233](#)
  - VDI, [106](#)
  - workstations vs. servers, [248](#)
- Maintenance contracts
  - costs, [894–895](#)
  - laptops vs. desktops, [102](#)

servers, [258–259](#)

## Maintenance windows

case studies, [614–615](#), [621](#)

change completion deadlines, [628](#)

change proposals, [617–620](#)

communications, [625–627](#)

comprehensive system testing, [628–630](#)

disabling access, [621–622](#)

flight director mentoring, [632](#)

flight director selection, [616–617](#)

high-availability sites, [634–636](#)

historical data, [632–633](#)

KVM, console service, and LOM, [625](#)

limited availability, [633–634](#)

management buy-in, [613](#)

master plans, [620–621](#)

mechanics and coordination, [622–627](#)

overview, [611–612](#)

post-maintenance communication, [630](#)

postmortem, [631–632](#)

process overview, [612](#)

remote access, [631](#)

scheduling, [613–615](#)

server upgrades, [600–602](#)

shutdown/boot sequence, [622–624](#)

task planning, [615–616](#)

visible presence, [631](#)

Major outage announcements, [522–523](#)

Major update change classifications, [571](#)

Makefiles, [548–549](#)

“Making It Virtually Easy to Deploy on Day One” post, [203](#)

## Malware

email, [735](#)

workstations, [95–96](#)

Managed assessment level, [1024–1025](#)

Managed services, [168–170](#), [641](#)

## Management

network operations, [432–437](#)

operational requirements, [293](#)

pleasing, [1078](#)

workstations vs. servers, [248–249](#)

Management chain influence, [897–898](#)

## Management support

centralizing services, [662](#)

maintenance windows, [613](#)

service requirements, [292](#)

ticket systems, [7](#)

## Managers

expectations of, [1088](#)

groups, [1081](#)

managing, [972–976](#)

Manual installation, [119](#), [145–146](#)

Manual updates, [93](#)

## Maps

groups onto storage infrastructure, [768–769](#)

printer, [755](#)

Marketing-driven customer support, [511](#)

Martyr role, [1110](#)

Master plans for maintenance windows, [620–621](#)

## Masters and slaves

DNS, [716–717](#)

redundancy design patterns, [322–323](#)

## Matrices

change risk, [572](#)

dependency, [309–310](#)

McEntee, Kevin, [240](#)

McKenty, Joshua, [38](#)

McKinley, D., [350](#)

McKusick, M., [534](#)

McLean, Malcom, [51](#)



- MDAs (mail delivery agents), [733](#)
- MDFs (main distribution frames), [402–403](#)
- MDM (mobile device management), [111–113](#)
- MDT (Microsoft Deployment Toolkit), [142](#)
- Mean time to repair (MTTR) in SLAs, [773](#)
- Measurable goals in S.M.A.R.T., [338](#)
- Measurable items in ready lists, [338–339](#)
- Measurable processes in DevOps, [17](#)
- Measure twice, cut once principle, [545–547](#)
- Measuring greatness. *See* [Assessments](#)
- Mechanics and coordination in maintenance windows, [622–627](#)
- Media for backups and restores, [812–814](#)
- Media relations in disaster recovery, [394](#)
- Medium risk change requests, [572](#)
- Medium-size companies, [908](#)
- Meetings
  - kick-off, [285–286](#)
  - management, [926](#)
  - negotiation, [955–956](#)
  - onboarding process, [210](#)
  - town hall, [927–929](#)
- Memory
  - caches, [271–272](#)
  - monitoring systems, [673](#)
  - NUMA, [272–273](#)
  - servers, [270–274](#)
  - swap space, [273–274](#)
- Mentoring flight directors, [632](#)
- Mentors, [989–990](#)
- Mercurial VCS, [57](#)
- Mergers and acquisitions, [1071–1072](#)
- Merging namespaces, [698](#)
- Message format for real-time monitoring, [681](#)
- Meta-monitoring, [690](#)
- Metadata

- in documentation templates, [553–554](#)
- packages, [830](#)

## Metrics

- operational assessments, [1041–1042](#)
- operational responsibilities, [1021](#)
- time to deployment, [361–362](#)

Micro-services, [126](#)

Microsoft Account (MSA), [88](#)

Microsoft Azure, [238](#)

- autoscale feature, [39](#)

- in programmable infrastructure, [56](#)

Microsoft Deployment Toolkit (MDT), [142](#)

Microsoft operating systems for workstations, [83–84](#)

Microsoft registry, [545](#)

## Migration

- launch with, [366–369](#)

- maintenance window example, [619–620](#)

- server upgrades, [595–596](#)

- VMs, [230](#)

MIL-SPEC scam, [262](#)

Miller, Cliff, [554](#)

Minimum viable product (MVP) strategy

- launches, [357–359](#)

- small batches principle, [31–33](#)

## Mirroring

- active listening, [951–952](#)

- backups, [276](#), [776](#)

- boot disks, [306](#)

- RAID disks, [762](#)

- servers, [250–251](#)

- software repositories, [836–837](#), [847–848](#)

- vendor, [827](#)

Mis-delegator stereotype, [520](#)

Misconfigured servers story, [541–543](#)

Misunderstandings in service requirements, [287](#)

Mixed-model BYOD strategies, [110–113](#)

MM (monitoring and metrics)

- operational assessments, [1041–1042](#)
- operational responsibilities, [1021](#)

Mobile device management (MDM), [111–113](#)

mock system, [845](#)

Model-based training, [521–522](#)

Models

- centralized. *See* [Centralized models](#)
- decentralized, [639](#), [895–896](#), [903](#)
- distributed, [639](#)
- funding, [895–896](#)
- hybrid, [596](#), [904](#)
- operational requirements, [296–297](#)
- OSI, [400–401](#)
- servers, [266](#)
- in service planning and engineering, [306](#)

Money for projects, [1076](#)

Monitor role, [1105](#)

Monitoring

- centralized, [653](#)
- data storage, [777–779](#)
- decentralized, [665](#)
- network operations, [431–432](#)
- services. *See* [Service monitoring](#)
- user expectations, [885–886](#)

Monitoring and Metrics (MM)

- operational assessments, [1041–1042](#)
- operational responsibilities, [1021](#)

Monitors, console, [475–476](#)

Monroe, M. G., [356](#)

Montrose, Kevin, [31](#)

Morals, [873](#)

Morgenstern, J., [945](#)

Moskowitz, Adam, [819](#)

Mother role, [1104](#)  
Motivation, [970–972](#)  
Mounting problems in NFS protocol, [535–536](#)  
Moving  
    datacenters, [466](#), [1069–1070](#)  
    to new buildings, [1070](#)  
    offices, [1071](#)  
    packages, [846](#)  
    variations to end, [51–52](#)  
MSA (Microsoft Account), [88](#)  
MTAs (mail transport agents), [733](#)  
MTTR (mean time to repair) in SLAs, [773](#)  
MUAs (mail user agents), [733](#)  
Multi-star topology, [414](#), [416](#)  
Multimedia servers, [852](#)  
Multiple authoritative DNS servers, [715–716](#)  
Multiple CPUs in servers, [267–270](#)  
Multiple OS installation, [142–143](#)  
Multiple web servers on one host, [853–854](#)  
Multitasking, [268–269](#), [272](#)  
Multithreading, [268–269](#)  
Multitiered mirrors and caches, [836–837](#)  
“Must have” feature sets, [337](#)  
Mutual-interruption shield technique, [921](#)  
MVP (minimum viable product) strategy  
    launches, [357–359](#)  
    small batches principle, [31–33](#)  
MX (Mail eXchanger) records, [739](#)  
My problems, [949](#)  
Mysterious deletions anecdote, [538–539](#)

$N + 1$  redundancy

overview, [250](#)

power, [277](#)

servers, [232](#)

service resilience, [326](#)

$N + 2$  redundancy

overview, [250](#)

resilience, [326](#)

NAC (network access control)

wired office networks, [405](#)

workstation fleet logistics, [177](#)

## Names

defining, [694–698](#)

DNS entries, [439](#)

packages, [843](#)

printers, [753–754](#)

service names vs. hostnames, [313–315](#)

## Nameservices

- AAA, [719–720](#)
- access policy, [721–723](#)
- authentication, [719](#)
- authority, [713](#)
- capacity and scaling, [713–714](#)
- case studies, [726–728](#)
- centralized management, [726–728](#)
- change policies, [723–724](#)
- change procedures, [724–726](#)
- consistency, [712–713](#)
- data, [711–712](#)
- databases, [720–721](#)
- DHCP, [717–718](#)
- DNS, [714–717](#)
- LDAP, [718](#)
- overview, [711](#)
- reliability, [714–721](#)

## Namespaces

- authority, [706–708](#)
- case studies, [702–707](#)
- conflicts, [698–699](#)
- consistency, [704–705](#)
- decentralized services, [667](#)
- defining names, [694–698](#)
- description, [693–694](#)
- DNS centralization, [650–651](#)
- email service, [730–731](#)
- federated identity, [708–709](#)
- life-cycle management, [699–700](#)
- merging, [698](#)
- overview, [693](#)
- reuse, [700–701](#)
- rules, [694–695](#)
- scope, [701–704](#)
- usage, [701–707](#)

Narayan, R., [376](#)

NAS. *See* [Network-attached storage \(NAS\)](#)

NASA Apollo program, [336](#)

ND (Neighbor Discovery), [84](#)

NDS (Network Directory Services), [88–89](#)

Needs determination for onboarding, [206](#)



## Negotiation

- meeting formats, [955–956](#)
- planning, [956–957](#)
- situational awareness, [954–955](#)
- tips, [958–960](#)
- win-win outcomes, [956–957](#)

## Neighbor Discovery (ND), [84](#)

## NetAid events, [579](#), [581](#)

## Network access control (NAC)

- wired office networks, [405](#)
- workstation fleet logistics, [177](#)

## Network access controls, hidden, [164–165](#)

## Network-accessible key repositories, [861](#)

## Network architecture

- corporate standards, [422–425](#)
- datacenter networks, [408–413](#)
- Internet access, [420–422](#)
- IPv6, [426–428](#)
- OSI model, [400–401](#)
- overview, [399](#)
- physical vs. logical, [399](#)
- routing, [419–420](#)
- SDNs, [425–426](#)
- WAN strategies, [413–418](#)
- wired office networks, [402–406](#)
- wireless office networks, [406–408](#)

## Network-attached storage (NAS)

- and backups, [777](#)
- performance, [781](#)
- RAID disks, [764](#)
- for reliability, [774](#)

## Network-based backups, [819](#)

## Network Directory Services (NDS), [88–89](#)

## Network drawings with overlays, [438](#)

## Network Information Service (NIS), [702](#)

Network interface cards (NICs), [274](#)

Network layer in OSI model, [400](#)–[401](#)

Network local interface, monitoring, [778](#)

## Network operations

- documentation, [437–440](#)

- management, [432–437](#)

- monitoring, [431–432](#)

- overview, [431](#)

- support, [440–446](#)

Network operation centers (NOCs), [444](#), [650](#)

Network services, [445–446](#)

Network sniffers, [442](#)

Network-synced storage, [91–92](#)

Network team in workstation logistics, [179](#)

## Networks

- bug classes, [533](#)
- centralization, [649–650](#)
- configuration, [84–86](#)
- conversions, [381–382](#)
- datacenters, [465](#)
- decentralized management, [666–667](#)
- design and implementation documentation, [438](#)
- equipment labeling, [473–474](#)
- off-site storage, [814](#)
- path visualization tools, [443](#)

Neville-Neil, G., [534](#)

*New Approach to Scripting* talk, [67](#)

New buildings, opening, [1070](#)

New hires

delivery process, [180](#)

first impressions, [916–917](#)

New items in Kanban, [10](#)

New product introduction and removal (NPI/NPR)

operational assessments, [1047–1048](#)

operational responsibilities, [1022](#)

New state in workstations, [118](#)

Newsletters, [930](#)

- NFS protocol
  - dependencies, [310](#)
  - mounting problems, [535–536](#)
- NICs (network interface cards), [274](#)
- Nine-track tape drives, [820–821](#)
- NIS (Network Information Service), [702](#)
- No update strategy, [125–126](#)
- NOCs (network operation centers), [444](#), [650](#)
- Non-integrated out-of-band management, [255–257](#)
- Non-persistent VDIs, [106–108](#)
- Non-Uniform Memory Architecture (NUMA) RAM systems, [272–273](#)
- Non-verifier stereotype, [520](#)
- Nonprofit organizations, [909–910](#)
- Nonstandard protocols for email, [736–737](#)
- Nonstandardized datacenters, [649](#)
- Nontechnical interviewing, [998–999](#)
- NPI/NPR (new product introduction and removal)
  - operational assessments, [1047–1048](#)
  - operational responsibilities, [1022](#)
- NTP client configuration, [72](#)
- NUMA (Non-Uniform Memory Architecture) RAM systems, [272–273](#)
- OAuth, [709](#)
- Observability in operational requirements, [292–293](#)
- Observing illegal activity, [888–889](#)
- OCI (Open Container Initiative), [51](#)
- O’Dell, Mike, [855](#)
- Off-site formal training, [534](#)
- Off-site links for documentation, [562–563](#)
- Off-site shredding services, [757–758](#)
- Off-site storage for backups and restores, [812–814](#)
- Off state in workstations, [118](#)
- Office location and visibility, [927](#)
- Office networks, diagnostics for, [441](#)
- Office Space*, [969](#)

## Offices

- in datacenters, [477](#)

- moving, [1071](#)

Offline spreadsheets for ready lists, [340](#)

Ogre stereotype, [520](#)

On-site shredding services, [757–758](#)

On-site spare parts, [260](#)

## Onboarding

- cadence changes, [212](#)

- case studies, [212–216](#)

- checklists, [205–206](#)

- communication, [208–209](#)

- first impressions, [201–202](#)

- improving, [209–211](#)

- IT responsibilities, [203](#)

- keys for success, [203](#)

- needs determination, [206](#)

- overview, [201](#)

- performing, [207–208](#)

- processes, [48](#)

- timelines, [204–206](#)

Oncall expert role, [1098](#)

*One Minute Manager*, [960](#)

*One Minute Sales Person*, [960](#)

One-shot tasks, [907](#)

One, some, many patch technique, [129](#)

One-time projects, [182](#)

Online spreadsheets for ready lists, [339](#)

Ontology, [287–288](#)

OOB management. *See* [Out-of-band \(OOB\) management](#)

Open architectures, [298–302](#)

Open Container Initiative (OCI), [51](#)

Open source software, [880](#)

Open standards for web services, [851–852](#)

Open Systems Interconnection (OSI) reference model, [400–401](#)

- OpenDirectory, [86](#)
- “OpenFlow: A Radical New Idea in Networking,” [426](#)
- OpenID technology, [709](#)
- Opening new buildings, [1070](#)
- OpenSSL cryptography library, [294](#)
- Operating system (OS) installation
  - automating in steps, [148–152](#)
  - automation, [142–143](#)
  - checklists, [146](#), [214](#)
  - cloning, [143–145](#)
  - consistency, [138–141](#)
  - manual, [145–146](#)
  - overview, [137](#)
  - test-driven configuration development, [146–148](#)
  - as time sinkhole, [13–14](#)
  - vendor support, [152–154](#)
  - workstations, [120](#)
- Operating systems (OS)
  - repositories, [827](#), [847–848](#)
  - workstation configuration, [120–123](#)
  - workstation updates, [93–94](#)
  - workstations, [82–83](#)
  - workstations vs. servers, [248–249](#)
- Operational assessments, [1035–1036](#)
  - capacity planning, [1043–1044](#)
  - case study, [1053](#)
  - change management, [1045–1046](#)
  - disaster preparedness, [1060–1061](#)
  - emergency response, [1039–1040](#)
  - monitoring and metrics, [1041–1042](#)
  - new product introduction and removal, [1047–1048](#)
  - performance and efficiency, [1051–1053](#)
  - regular tasks, [1036–1038](#)
  - service delivery build phase, [1054–1055](#)
  - service delivery deployment phase, [1056–1057](#)



service deployment and decommissioning, [1049–1050](#)

toil reduction, [1058–1059](#)

Operational device phase, [434](#)

Operational excellence

measuring greatness, [1020–1021](#)

overview, [1019–1020](#)

Operational level agreements (OLAs), [306](#), [317](#)

Operational models in service planning and engineering, [306](#)

Operational requirements, [292](#)

disaster recovery, [298](#)

environment fit, [295–296](#)

remote and central management, [293](#)

scaling, [293–294](#)

service requests, [297](#)

software upgrades, [294–295](#)

support models, [296–297](#)

system observability, [292–293](#)

Operational responsibilities in assessment, [1021–1022](#)

Operational service requirements, [286](#)

Opportunity costs in cloud-based services, [240](#)

Optimization

data storage, [780–784](#)

server hardware strategies, [226–227](#)

Optimizing assessment level, [1024](#)

Orchestration, application, [68](#)

Orders, illegal and unethical, [887–888](#)

Organizational assessment, [1029–1030](#)

Organizational learning in LRR, [347](#)

## Organizational structures

- case studies, [893–894](#), [901–902](#), [910](#)
- consultants and contractors, [906–907](#)
- customer support, [902–904](#)
- funding models, [894–897](#)
- helpdesk, [904](#)
- infrastructure teams, [900–902](#)
- management chain influence, [897–898](#)
- outsourcing, [904–906](#)
- overview, [891–892](#)
- sample, [907–910](#)
- sizing, [892–894](#)
- skill selection, [898–900](#)
- support, [443–445](#)

## Organizers, [937](#)

*Organizing from the Inside Out*, [945](#)

Orientation sessions, [917–918](#)

OS. See [Operating systems \(OS\)](#)

OSI (Open Systems Interconnection) reference model, [400–401](#)

Other people's problems, [950](#)

Our problems, [949](#)

## Out-of-band (OOB) management

- datacenter networks, [412](#)
- integrated, [254–255](#)
- non-integrated, [255–257](#)
- remote management, [247](#)

## Outages

- announcements, [522–523](#)
- data storage, [773–774](#)
- dealing with, [1073–1074](#)
- from decentralized DNS, [651](#)
- monitoring, [778](#)
- ticket systems, [7](#)

Outbound Internet connectivity, [420–421](#)

Outer loops in release process, [26](#)

Outsider role, [1109](#)

## Outsourcing

- datacenters, [451](#)

- organizational structures, [904–906](#)

- printing, [754](#)

Outward appearances, [914–915](#)

Overlap in workstation standardization, [193–194](#)

Overlays, network drawings with, [438](#)

Oversubscription, [270](#)

## PaaS (platform as a service)

- access management, [1008](#), [1010](#)

- datacenters, [450](#), [456](#)

- web servers, [860](#)

Packaged software, rapid release with, [359–362](#)

## Packages

anatomy, [829–833](#)

management systems, [827](#), [829](#)

software repository, [825–826](#)

staging, [843](#)

workstations, [123](#)

Packer system for images, [144](#)

Packet size, [332–333](#)

Packing, VM, [230–231](#)

Pages, memory, [271](#)

Paper for printer, [756–757](#)

Paper plates analogy, [39](#)

Paper trails for unethical orders, [887–888](#)

Parallel requests, [332](#)

Parameterized code, [59](#)

Parity in RAID disks, [762–763](#)

Parking spaces at datacenters, [480](#)

Part-time fleet management, [186–187](#)

## Passwords

fired system administrators, [1009–1010](#)

workstations, [86–88](#)

## Patch cables

buying vs. making, [468](#)

labeling, [471–472](#)

overview, [468–471](#)



Patch panels

central switches with, [409–410](#)

labeling, [439–440](#)

Patch rooms, [449](#)

## Patches

software, [436](#), [826](#)

source packages, [832](#)

workstations, [93](#), [117](#), [129](#)

workstations vs. servers, [249](#)

Paterson, R., [961](#)

Path of least resistance for standardization, [196](#)

Path visualization tools, [443](#)

PATs (production assurance tests), [295](#)

PC deployment bottlenecks, [18–19](#)

PDU (power distribution units), [255](#)

PE (performance and efficiency)

operational assessments, [1051–1053](#)

operational responsibilities, [1022](#)

## Perception

aligning priorities with customer expectations, [920–921](#)

attitude, [918–920](#)

first impressions, [914–917](#)

fixing, [1069](#)

overview, [913–914](#)

system advocates, [921–925](#)

## Performance

- antivirus software, [96](#)

- backups, [799–800](#), [807–808](#)

- capacity planning, [327–328](#)

- CPUs, [271–273](#)

- data storage, [780–784](#)

- vs. resilience, [325–326](#)

- and scaling, [326–333](#)

- service patterns. See [Service resiliency and performance patterns](#)

- solving problems, [923–924](#)

- systems, [1078](#)

- workstations, [105](#)

- workstations vs. servers, [246](#)

## Performance and efficiency (PE)

- operational assessments, [1051–1053](#)

- operational responsibilities, [1022](#)

## Periodic launch category, [355–356](#)

## Permissions

license management, [880](#)

web services, [866](#)

Persistent VDIs, [106–109](#)

*Personal Kanban: Mapping Work | Navigating Life*, [11](#)

Personal life and work, balancing, [966–967](#)

Pervasive monitoring, [686–687](#)

## Pets and cattle analogy

automation, [53](#)

desktops as cattle, [40](#)

generic processes, [47–51](#)

introduction, [37–39](#)

isolating states, [44–46](#)

pets store state, [43–44](#)

scaling, [39](#)

server hardware as cattle, [41–43](#)

variations, [51–52](#)

Phillips, G., [987](#)

*Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*, [17](#)

Phone menus for customer support, [511](#)

Phone number conversions, [383](#)

Physical access for fired system administrators, [1008](#)

Physical design in corporate standards, [424–425](#)

Physical disposal of workstations, [132–133](#)

Physical infrastructure

datacenter networks, [409–412](#)

wired office networks, [402–403](#)

wireless office networks, [406](#)

Physical layer in OSI model, [400–401](#)

Physical networks, [399](#)

Physical visibility, [927](#)

Physical workstations

laptops vs. desktops, [101–102](#)

product line selection, [103–105](#)

vendor selection, [102–103](#)

Pike, R., [593](#)

Pillars vs. layers approaches in service conversions, [376–377](#)

ping tool, [535–536](#)

Pipeline optimization, [783–784](#)

Pirated software, [859](#), [879](#)

Pivoting, [30](#)

Planning and execution phase in incident reports, [515–518](#)

Plans

backout, [383–385](#), [573–574](#), [600](#)

capacity. See [Capacity planning \(CP\)](#)

centralizing services, [661–662](#)

change management, [573–574](#)

contingency, [393](#)

future storage needs, [770–771](#)

maintenance tasks, [615–616](#)

maintenance windows, [620–621](#)

negotiations, [956–957](#)

for problems, [335–336](#)

and risk, [388](#)

server upgrade backout, [600](#)

service. See [Service planning and engineering](#)

Platform as a service (PaaS), [860](#)

Platform team in workstation fleet logistics, [178](#)

Platforms, [158–159](#)

Platters, disk, [760–761](#)

Plonka, D., [436](#)

PMO (program office), [181–182](#)



## Policies

- alerts, [680–681](#)

- backups and restores, [795](#), [800–801](#)

- centralization security, [646](#)

- copyrights, [878–881](#)

- documentation, [1075–1076](#)

- email retention, [743–744](#)

- email service, [730](#)

- labeling, [473](#)

- nameservice access, [721–723](#)

- nameservice changes, [723–724](#)

- print service architecture, [751–754](#)

- refresh cycles, [163](#)

- storage spares, [769–770](#)

- web services, [867–868](#)

Policy enforcer role, [1099–1100](#)

Policy navigator role, [1106](#)

Policy writer role, [1093–1094](#)

Polling systems for historical monitoring, [674–675](#)

POP3 protocol, [741](#)

Porcelain plates analogy, [39](#)

Post-maintenance communication

- high-availability sites, [635](#)

- maintenance windows, [630](#)

Postmaster address, monitoring, [738](#)

Postmortem for maintenance windows, [631–632](#)

## Power

computer rooms, [253](#)

datacenters, [462–464](#)

monitoring, [788](#)

power cable separation, [470–471](#)

power cord labeling, [439](#)

servers, [277](#)

Power distribution units (PDUs), [255](#)

Power dynamics in negotiations, [955](#)

*Practical Guide to Risk Management*, [387](#)

*Practice of Cloud System Administration*, [855](#)

Pre-labeled cables, [473](#)

Precompiling decisions, [942–943](#)

Prep rooms, [174–177](#)

## Preparation

disaster recovery, [391–392](#)  
maintenance windows, [612](#)  
negotiations, [956–957](#)  
web services, [865–866](#)

Presentation layer

content-management systems, [560](#)

OSI model, [400–401](#)

Preston, W. C., [794](#)

Prevent mode in workstations, [95](#)

Primary applications, [284](#)

Primary dependencies, [309](#)

## Print service

- access policies, [752–753](#)
- architecture policy, [751–754](#)
- case studies, [752](#), [755](#), [756](#)
- centralization, [750](#)
- documentation, [754–755](#)
- environmental issues, [756–757](#)
- monitoring, [755–756](#)
- outsourcing, [754](#)
- overview, [749](#)
- printer equipment standards, [751](#)
- printer maps, [755](#)
- printer naming policies, [753–754](#)
- shredding, [757](#)
- standards, [751–752](#)

## Priorities

aligning with customer expectations, [920–921](#)

service monitoring, [684](#)

setting, [1086](#)

## Privacy

- datacenter requirements, [453–454](#)

- email policy, [730](#)

- and law enforcement, [881](#)

- user expectations, [885–886](#)

Privileged-access codes of conduct, [877–878](#)

Privileges for web services, [866](#)

Problem identification phase in incident reports, [509–515](#)

Problem preventer role, [1091](#)

## Problems

- categories, [949–950](#)

- classifying, [511](#)

- debugging. See [Debugging](#)

- planning for, [335–336](#)

- service launches, [349–351](#)

- statements, [511–513](#)

- verification, [513–514](#)

Procedural Test-Oriented Scripting (PTOS), [67](#)

Procedures documentation, [562](#), [1075](#)

Process of elimination in debugging, [532](#)



## Processes

generic, [47–51](#)

incident reports, [506–508](#)

Product finder role, [1096](#)

Product lines

servers, [266](#)

workstations, [103–105](#)

Production assurance tests (PATs), [295](#)

Production environments, cloning, [362–363](#)

Production services, launching, [343](#)

Production staging areas, [342](#)

Productivity issues for workstations, [79–80](#)

Products vs. protocols, [299](#)

Professional development, [967–968](#), [1086](#)

Program office (PMO), [181–182](#)

Programmable infrastructure, [56](#)

Programming languages, declarative vs. imperative, [64–65](#)

Project-focused hours, [7](#)

Project management

dashboards, [180](#)

workstation fleet logistics, [180–181](#)

## Projects

completing, [1076–1077](#), [1087](#)

money for, [1076](#)

planning cycles, [1028](#)

Promotions, asking for, [973](#)

Proposals in change management, [570](#)

Proprietary email, [300–302](#)

Protected files in web services, [861](#)

Protective triangles, [1009](#)

## Protocols

email, [736–737](#)

interior routing, [419–420](#)

open, [298–302](#)

vs. products, [299](#)

*Providence: Failure Is Always an Option*, [31](#), [350](#)

Provisioning helpdesks, [502](#)

PTOS (Procedural Test-Oriented Scripting), [67](#)

Public data, [722](#)

Public packages, repackaging, [838](#)

Pull systems

DevOps, [17](#)

Kanban, [11](#)

software repositories, [826](#)

Punyon, Jason, [31](#), [350](#)

Puppet modules, [63](#)–[64](#)

Purchasing

as bottleneck, [19](#)

centralization, [655–656](#)

workstation fleet logistics, [174–175](#)

Push-on-green launch category, [355–356](#)

Push systems

software repositories, [826](#)

ticket systems, [11](#)

PUT command in SNMP, [678](#)

PXE booting, [142](#), [148](#)

QIC tapes, [821](#)

QRCs (Quick Request Cards) in Kanban, [10](#)



Quality

automation, [74](#)

decentralization for, [641](#)

Quality Assurance staging areas, [342](#)

Queries in web services, [863](#)

Query lists for latency, [332](#)

## Questions

- capacity planning, [1043](#)
  - centralization, [660–661](#)
  - change management, [1045](#)
  - customer service requirements, [289](#)
  - disaster preparedness, [1060](#)
  - emergency response, [1039](#)
  - interview process, [992](#)
  - monitoring and metrics, [1041](#)
  - nontechnical interviewing, [998–999](#)
  - OS installation, [149](#)
  - performance and efficiency, [1051](#)
  - product introduction and removal, [1047](#)
  - regular task assessments, [1036](#)
  - service delivery build phase, [1054](#)
  - service delivery deployment phase, [1056](#)
  - service deployment and decommissioning, [1049](#)
  - storage-needs assessments, [766–768](#)
  - technical interviewing, [995–997](#)
  - toil reduction, [1058](#)
- Quick guides, [553–554](#)
- Quick Request Cards (QRCs) in Kanban, [10](#)
- Quotas, [769](#)

## Racks

datacenters, [410–412](#), [461–462](#)

dependency alignment, [312](#)

labeling, [471–472](#)

workstations vs. servers, [247](#)

Radios for maintenance windows, [626–627](#)

RADIUS (Remote Authentication Dial-In User Service)

AAA, [719](#)

VPN servers, [702](#)

Radosevich, L., [568](#)

RAID. *See* [Redundant Array of Independent Disks \(RAID\)](#)

RAM. *See* [Memory](#)

RANCID (Really Awesome New Cisco Config Differ) product, [436](#)

Rapid Release Initiative, [353–354](#)

Rapid release with packaged software, [359–362](#)

Ratcheting for workstation standardization, [194–195](#)

Rate of storage change, monitoring, [778](#)

Raw disk capacity, [766](#)

RDP (Remote Desktop Protocol), [93](#)

Read-only slaves, [323](#)

Ready lists

- defining, [337–340](#)

- launches, [336](#)

- list attributes, [338–339](#)

- storing, [339–340](#)

- working, [340–341](#)

Real-time monitoring, [672](#)

- alerts, [679–682](#)

- case studies, [682–683](#)

- log processing, [679](#)

- overview, [676–677](#)

- SNMP, [677–679](#)

Really Awesome New Cisco Config Differ (RANCID) product, [436](#)

Rear access to racks, [247](#)

Reboot test anecdote, [582–583](#)

Rebuild process for workstations, [118](#)

Recent Changes areas in wiki systems, [560](#)

Recorders for customer support, [511–513](#)

Recruiting, [983–984](#)

Recycling toner cartridges, [756–757](#)

Redirectors, URL, [559](#)

Redirects, HTTP, [837](#)

## Redundancy

email, [738–739](#)

servers, [232](#), [250–251](#)

Redundancy design patterns

- load balancers plus replicas, [323](#)

- masters and slaves, [322–323](#)

- performance vs. resilience, [325–326](#)

- replicas and shared state, [324–325](#)

Redundant Array of Independent Disks (RAID) all eggs in one basket strategy, [223](#)

- and backups, [276](#), [775–776](#)

- directly attached storage, [764](#)

- hardware, [275–276](#)

- levels, [761–763](#)

- network-attached storage, [764](#)

- performance, [780–781](#)

- for reliability, [774](#)

- servers, [250–251](#)

- software, [275](#)

- storage-area networks, [764](#)

- volumes and file systems, [763](#)

Redundant links, monitoring, [432](#)

Redundant sites, [393–394](#)

Reference lists, [562](#)

## Reflection

- active listening, [953–954](#)

- onboarding process, [209–211](#)

Refresh cycles in workstation service definition, [161–164](#)

Refresh process in workstation fleet logistics, [180](#)

Regional operations centers (ROCs), [444](#), [650](#)

Registry, [545](#)

Regression testing, [593](#)

Regressions in OS installation, [148](#)

## Regular Tasks (RTs)

- operational assessments, [1036–1038](#)

- operational responsibilities, [1021](#)

Regulatory constraints requirements for datacenters, [453](#)

Rehearsals for server upgrades, [599–600](#)

Reingold, Tommy, [182](#), [376](#), [932](#)

Reinstallation of workstations, [119](#)

Relationships in customer support, [903](#)

Release distributions, [826](#)

Release process

loops, [26](#)

software repositories, [836](#)

Relevant goals in S.M.A.R.T., [338](#)



## Reliability

- computer rooms, [253–254](#)
- data integrity, [250–251](#)
- data storage, [773–775](#)
- email service, [731–732](#)
- hot-swap components, [252–253](#)
- increasing, [1082](#)
- levels, [250](#)
- nameservices, [714–721](#)
- servers, [249–254](#)
- web services, [860](#)
- workstations, [79–80](#)

## Remote access

- datacenter requirements, [455](#)

- fired system administrators, [1008–1009](#)

- maintenance windows, [631](#)

## Remote Authentication Dial-In User Service (RADIUS)

- AAA, [719](#)

- VPN servers, [702](#)

Remote console with serial consoles, [256–257](#)

Remote Desktop Protocol (RDP), [93](#)

Remote insight board (RIB), [255](#)

Remote Insight Light-Out Edition (RILOE), [255](#)

Remote logistics in workstation fleet logistics, [180](#)

Remote management

- operational requirements, [293](#)

- servers, [254–257](#)

- workstations vs. servers, [247](#)

Remote power cycle, [255](#)

Remote storage for workstations, [91](#)

Remote-wipe feature, [95](#)

Removable media, [886](#)

Removing

    fired system administrators access, [1007–1008](#)

    packages, [830](#)

Renewal requests in DHCP, [717–718](#)

Renting datacenters, [450–451](#)

Reorganizing, rationale for, [640](#)

Repackaging

public packages, [838](#)

third-party software, [839](#)

Repair, workstation, [104](#)

Repair person role, [1090–1091](#)

Repeat step in service launch process, [345](#)

Repeatable assessment level, [1023](#)

Repeaters, [626–627](#)

Replacing services, [1069](#)

## Replicas

- load balancers plus replicas, [323](#)

- scaling, [39](#), [46](#), [294](#)

- shared state, [324–325](#)

Reporting in antivirus software, [96](#)

Repositories

in content-management systems, [560](#)

software. *See* [Software repositories](#)

tests, [604](#)

Reproducibility of containers, [234–235](#)

Request-tracking software for helpdesks, [496–498](#)

Requests for Comment (RFCs), [717](#)

## Resilience

vs. performance, [325–326](#)

service. See [Service resiliency and performance patterns](#)

Resistance to workstation standardization, [192](#)

Resolution stage in maintenance windows, [612](#)

Resolvers, DNS, [714](#)

Resource contention in software repositories, [840](#)

Resource isolation, VMs for, [228](#)

## Resources

availability in service launches, [349–350](#)

datacenters, [460](#)

service requirements, [291–292](#)

usage monitoring, [779](#)

Response time monitoring for applications, [688–689](#)

Responsibility in helpdesks, [491–492](#)

Responsiveness datacenter requirements, [456](#)

Restores. See [Backups and restores](#)

Restoring access after server upgrades, [606](#)

Retention policy for email service, [743–744](#)

Retrospective for launches, [337](#)

Reuse of namespaces, [700–701](#)

Revision histories in change management, [583–585](#)

RFCs (Requests for Comment), [717](#)

RIB (remote insight board), [255](#)

Richardson, J., [362](#)

Ries, Eric, [31](#)

RILOE (Remote Insight Light-Out Edition), [255](#)

Ring topology, [415–416](#)

Ringel, M. F., [1005](#)

Rioting mob technique, [375–376](#)



## Risks

- change management, [572–573](#)

- disaster recovery, [388–389](#)

- failover process, [26–28](#)

- infrastructure as code, [59](#)

- MVP, [358](#)

- server upgrades, [597](#)

rm command, [546](#)

Roaming profiles, [141](#)

ROCs (regional operations centers), [444](#), [650](#)

## Roles, system administrator

- ad hoc solution finder, [1097](#)

- bleeding edger, [1107](#)

- budget administrator, [1102](#)

- capacity planner, [1101](#)

- Captain Break Time, [1111](#)

- careful planner, [1101](#)

- case study, [1094–1095](#)

- cowboy, [1107–1108](#)

- crying wolf, [1107](#)

- customer/SA, [1105–1106](#)

- customer's advocate, [1102](#)

- disaster worrier, [1100](#)

- doer of repetitive tasks, [1111](#)

- educator, [1098–1099](#)

- end-to-end expert, [1109](#)

- facilitator, [1105](#)

- go-to person, [1092](#)

- helpdesker, [1106](#)

- hero, [1092](#)

- infrastructure builder, [1093](#)

- installer, [1090](#)

- lab technician, [1095–1096](#)

- level-focused person, [1109–1110](#)

- maintainer, [1091](#)

martyr, [1110](#)  
monitor, [1105](#)  
mother, [1104](#)  
oncall expert, [1098](#)  
overview, [1089–1090](#)  
outsider, [1109](#)  
policy enforcer, [1099–1100](#)  
policy navigator, [1106](#)  
policy writer, [1093–1094](#)  
problem preventer, [1091](#)  
product finder, [1096](#)  
repair person, [1090–1091](#)  
salesperson, [1103](#)  
slaves, scapegoats, and janitors, [1108–1109](#)  
social director, [1111](#)  
solution designer, [1096–1097](#)  
system clerk, [1094](#)  
technocrat, [1102–1103](#)  
technology staller, [1107](#)  
unrequested solution person, [1097–1098](#)  
vendor liaison, [1103–1104](#)  
visionary, [1104](#)

## Rollouts

- documentation, [560](#)

- gradual, [379–380](#)

- IPv6, [428](#)

## Rolling upgrades, [323](#)

## Root CAs, [862](#)

## Root DNS servers, [715–716](#)

## Routine update change classifications, [571](#)

## Routing

- exterior gateway protocols, [420](#)

- interior routing protocols, [419–420](#)

- overview, [419](#)

- static, [419](#)

`rpcinfo` tool, [535–536](#)

## RTs (regular tasks)

- operational assessments, [1036–1038](#)

- operational responsibilities, [1021](#)

## Rules

- firewalls, [164](#), [315](#)
- namespaces, [694–695](#)
- unsupported hardware, [165–167](#)

Runbooks, [318](#)

Russinovich, M., [534](#)

Rutgers University, [726](#)

## SaaS (Software as a Service)

- datacenters, [451–456](#)
- example, [443](#)
- overview, [241](#)

Safe workplaces, [971](#)

Salary negotiations, [959–960](#)

Salesperson role, [1103](#)

SAM (Security Accounts Manager), [86](#)

SAML (Security Assertion Markup Language), [709](#)

SANs. *See* [Storage-area networks \(SANs\)](#)

## Sarbanes-Oxley Act

- dual teams, [909](#)
- email, [744](#)

Saturation, data storage, [789](#)

Scale-out, [294](#)

Scale-up, [294](#)

## Scaling

dataflow analysis for, [328–330](#)

decoupling state, [46](#)

email service, [739–742](#)

importance, [859](#)

nameservices, [713–714](#)

operational requirements, [293–294](#)

and performance, [326–333](#)

pets and cattle analogy, [39](#)

service monitoring, [684–685](#)

updates, [93](#)

web servers, [853–854](#)

web services, [855–859](#)

Scapegoat role, [1108–1109](#)

## Schedules

assessments, [1032](#)

backups and restores, [795](#), [801–807](#)

change management, [574–575](#)

grid computing, [235–236](#)

maintenance windows, [613–615](#)

service requirements, [291–292](#)

ticket systems, [8](#)

Schneier, Bruce, [97](#)

SCIM (Secure Cross-Domain Identity Management), [709](#)

## Scope

helpdesks, [490–493](#)

namespaces, [701–704](#)

service requirements, [287](#), [291–292](#)

Screenshots for documentation, [554](#)

script command, [555](#)



## Scripts

automation, [924](#)

helpdesk, [494](#)

packages, [830](#)

server upgrades, [594](#)

*Scrooged*, [970](#)

SDD (service deployment and decommissioning)

operational assessments, [1049–1050](#)

operational responsibilities, [1022](#)

SDLC. *See* [software delivery life cycle](#)

SDNs (software-defined networks), [425–426](#)

Searches in wiki systems, [559](#)

Second support tier for workstations, [167](#)

Second Way in DevOps, [211](#)

Sectors, disk, [760–761](#)

Secure Cross-Domain Identity Management (SCIM), [709](#)

Secure Shell (SSH), [93](#)

Secure Sockets Layer (SSL), [860–861](#)

SecurID authentication server upgrade, [618–619](#)

## Security

- backups and restores, [813–816](#)
  - BYOD, [111–112](#)
  - centralization, [645–648](#)
  - CMDB for, [184](#)
  - datacenter networks, [412–413](#)
  - datacenter requirements, [455](#)
  - decentralized services, [667](#)
  - disasters, [394](#)
  - DNS/DHCP infrastructure software, [364](#)
  - email service, [742](#)
  - inbound connectivity, [421–422](#)
  - monitoring systems, [673](#)
  - namespaces, [697–698](#)
  - outbound connectivity, [421](#)
  - outsourcing, [905](#)
  - print service, [757–758](#)
  - remote management, [254](#)
  - SNMP, [678](#)
  - software repositories, [834–835](#), [842](#)
  - VDI, [109](#)
  - virtualization for, [229](#)
  - web services, [859–866](#)
  - wired office networks, [403](#)
  - workstation disposal, [132](#)
  - workstations, [94–97](#)
  - workstations vs. servers, [248–249](#)
- Security Accounts Manager (SAM), [86](#)
- Security Assertion Markup Language (SAML), [709](#)
- “Security Market for Lemons,” [97](#)
- Seek time, disk, [760–761](#)
- Segal’s law, [140](#)
- Self-confidence of teams, [1084](#)
- Self-contained packages, [842](#)
- Self-help books, [976](#)

Self-help desk documentation, [562](#)  
Self-help systems, [487](#)  
Self-service automation for nameservice changes, [725–726](#)  
Self-service container administration, [234](#)  
Self-signed certificates, [862](#)  
Self-updating software, [369](#)  
Selling the position, [1000](#)  
Selling workstations, [133](#)  
Seminars for professional development, [968](#)  
Sensitive update change classifications, [571](#)  
Sequential names, [697](#)

## Serial consoles

- datacenters, [475–476](#)
- maintenance windows, [625](#)
- remote console with, [256–257](#)

## Server hardware as cattle, [41–43](#)

## Server hardware features

- administrative networks, [257–258](#)
- cross-shipping, [261](#)
- maintenance contracts, [258–259](#)
- overview, [245](#)
- reliability, [249–254](#)
- remote management, [254–257](#)
- service contracts, [260–261](#)
- spare parts, [259–260](#)
- vendor selection, [261–262](#)
- workstations vs. servers, [246–249](#)

## Server hardware specifications

- CPUs, [267–270](#)
- details, [266](#)
- memory, [270–274](#)
- models and product lines, [266](#)
- network interfaces, [274](#)
- overview, [265–266](#)
- power supplies, [277](#)
- RAID disks, [275–276](#)
- unnecessary items, [278](#)

## Server hardware strategies

- all eggs in one basket, [222–224](#)
- appliances, [241–242](#)
- beautiful snowflakes, [224–228](#)
- blade servers, [237–238](#)
- buy in bulk and allocate fractions, [228–235](#)
- case study, [236–237](#)
- cloud-based compute services, [238–241](#)
- containers, [234–235](#)

grid computing, [235–236](#)

hybrid, [242–243](#)

overview, [221–222](#)

VMs, [228–235](#)

Server name indication (SNI), [854](#)

Server upgrades, [223–224](#)

announcing, [602–603](#)

backing out, [605–606](#)

backout plans, [600](#)

case studies, [602](#), [607–608](#)

communication, [606–607](#)

implementation plans, [598–600](#)

locking out customers, [604](#)

maintenance windows, [600–602](#)

overview, [587](#)

performing, [605](#)

process, [587–588](#)

rehearsals, [599–600](#)

restoring access, [606](#)

service checklists, [588–591](#)

software compatibility, [591–592](#)

strategies, [595–598](#)

testing, [603–605](#)

verification tests, [592–595](#)

version mixing, [599–600](#)

## Servers

- blade, [237–238](#)

- console, [475–476](#)

- datacenter networks, [412](#)

- name conflicts, [699](#)

Service checklists for server upgrades, [588–591](#)

Service contracts for servers, [260–261](#)

## Service conversions

- backout plans, [383–385](#)

- communication, [378–379](#)

- flash-cuts, [380–382](#)

- gradual roll-outs, [379–380](#)

- intrusiveness, [374](#)

- layers vs. pillars, [376–377](#)

- overview, [373–374](#)

- training, [379](#)

- vendor support, [377–378](#)

Service definition for workstations. See [Workstation service definition](#)

## Service delivery

- build phase, [1054–1055](#)

- deployment phase, [1056–1057](#)

## Service deployment and decommissioning (SDD)

- operational assessments, [1049–1050](#)

- operational responsibilities, [1022](#)

## Service launch DevOps

- case study, [361](#)

- cloning production environments, [362–363](#)

- continuous integration and deployment, [354–357](#)

- data migration, [366–369](#)

- DNS/DHCP infrastructure software, [363–366](#)

- MVP, [357–359](#)

- overview, [353–354](#)

- rapid release with packaged software, [359–362](#)

- self-updating software, [369](#)

## Service launch fundamentals

- calendars, [348](#)
- launch readiness reviews, [345–348](#)
- overview, [335](#)
- planning for problems, [335–336](#)
- problems, [349–351](#)
- six-step process, [336–345](#)

## Service level agreements (SLAs)

- backups and restores, [795](#)
- customer service requirements, [290](#)
- data-recovery, [800–801](#)
- data storage, [773](#)
- maintenance windows, [601](#)
- servers, [258–259](#)
- service planning and engineering, [317](#)
- ticket systems, [6](#)
- web services, [854–855](#)

## Service monitoring, [316–317](#)

- application response time, [688–689](#)
- building monitoring systems, [673](#)
- case studies, [687](#), [689](#)
- centralization and accessibility, [685–686](#)
- compliance monitoring, [689–690](#)
- email, [738](#)
- end-to-end tests, [687–688](#)
- historical, [674–676](#)
- meta-monitoring, [690](#)
- overview, [671–672](#)
- pervasive, [686–687](#)
- print, [755–756](#)
- real-time, [676–683](#)
- scaling, [684–685](#)
- types, [672](#)
- web, [855](#)

## Service planning and engineering



- case studies, [307–308](#), [314](#)
- decoupling hostnames from service names, [313–315](#)
- dependency engineering, [309–313](#)
- engineering basics, [306–307](#)
- overview, [305–306](#)
- simplicity, [307–308](#)
- support models, [315–318](#)
- vendor-certified designs, [308–309](#)
- Service request model (SRM), [317–318](#)

## Service requirements

- customer, [288–291](#)

- importance, [284–285](#)

- kick-off meetings, [285–286](#)

- open architecture, [298–302](#)

- operational, [292–298](#)

- overview, [283–284](#)

- scope, schedule, and resources, [291–292](#)

- written, [286–287](#)

## Service resiliency and performance patterns bandwidth vs. latency, [330–332](#)

- case study, [332–333](#)

- overview, [321–322](#)

- performance and scaling, [326–333](#)

- redundancy design patterns, [322–326](#)

## Service set identifiers (SSIDs), [407](#)

## Services

- adding and removing in server upgrades, [598–599](#)
  - architecture, [305](#)
  - assessment, [1025–1028](#)
  - centralizing. *See* [Centralizing services](#)
  - datacenter requirements, [454](#)
  - decoupling names from hostnames, [313–315](#)
  - failure domains, [311–312](#)
  - infrastructure as code, [63](#)
  - outages. *See* [Outages](#)
  - replacing, [1069](#)
- Servicing device phase, [434](#)
- Session layer in OSI model, [400–401](#)
- Settings for workstations, [121](#)
- Shame, [970–971](#)
- Shared directories, [557](#)
- Shared libraries, [831](#)
- Shared passwords, [1009–1010](#)
- Shared spreadsheets for ready lists, [339](#)
- Shared state in redundancy design patterns, [324–325](#)
- Shared storage for VMs, [230–231](#)
- Shift down approach in refresh cycles, [162–163](#)
- Ship It! A Practical Guide to Successful Software Projects*, [362](#)
- Shipping containers, [51](#)
- Shoe-shining effect, [807](#)
- Short-sighted approach for refresh cycles, [162–163](#)
- Short-term changes in service launch lessons learned, [344](#)
- Shredding printer output, [757](#)
- Shutdown sequence for maintenance windows, [622–624](#)
- Silence as negotiating tool, [960](#)
- Silent updating in antivirus software, [96](#)
- Siloing, [182](#)
- Simple Network Monitoring Protocol (SNMP), [677–679](#)

Simplicity

email service, [733–735](#)

service planning and engineering, [307–308](#)

Simultaneous workstation access, [92](#)

Singh, A., [412](#)

Single sign-on (SSO), [708](#)

## Sites

- building from scratch, [1067](#)

- small, [1068](#)

Situational awareness, [292–293](#)

Six-step service launch process

- beta launch, [342–343](#)

- lessons learned, [343–345](#)

- production launch, [343](#)

- ready lists, defining, [337–340](#)

- ready lists, working, [340–342](#)

- repeat step, [345](#)

- service launch fundamentals, [336–337](#)

Sizing organizational structures, [892–894](#)

Skill levels factor in hiring system administrators, [982](#)

Skill selection in organizational structures, [898–900](#)

SLAs. See [Service level agreements \(SLAs\)](#)

Slashdot effect, [855](#)

Slave role, [1108–1109](#)

Slaves

DNS, [716–717](#)

masters and slaves pattern, [322–323](#)

Slow bureaucrats, [944–945](#)

Small batches principle, [23](#)

benefits, [25–26](#)

carpenter analogy, [23–24](#)

emergency failovers, [26–29](#)

Hell Month, [24–26](#)

infrastructure as code, [63](#)

launches, [29–33](#)

Small business workstation products, [104](#)

Small changes, importance of, [20–21](#)

Small companies, [908](#)

Small-scale fleet logistics, [186–188](#)

Small sites

growing, [1068](#)

OS installation, [147](#)

*Smart and Gets Things Done*, [980](#)

S.M.A.R.T., [338](#)

Smart network sniffers, [442](#)

Smart pipelining algorithms, [783](#)

Smith, M. J., [961](#), [1009](#)

Smoke tests, [60](#)

SMTP, [736](#)

Snapshots, [743](#)

benefits, [796–797](#)

file, [775](#)

SNI (server name indication), [854](#)

Sniffers, [442](#)

SNMP (Simple Network Monitoring Protocol), [677–679](#)

SNMP trap-based system, [673](#)

Snowflake metaphor, [38–39](#), [224–227](#)

Social director role, [1111](#)

Social engineering, [883–885](#)

Soft dependencies, [310](#)

## Software

- compatibility in server upgrades, [591–592](#)
- deployment as time sinkhole, [15–16](#)
- deployment process, [437](#)
- installing, [922–923](#)
- for labeling, [474](#)
- life cycle. *See* [Workstation software life cycle](#)
- in onboarding, [209](#)
- packages, [825–826](#)
- self-updating, [369](#)
- trouble-tracking, [508](#)
- updates, [123–128](#)
- upgrades, [294–295](#)
- versions, [436](#)

## Software as a service (SaaS)

- datacenters, [451](#)
- overview, [241](#)

## Software-defined networks (SDNs), [425–426](#)

## Software delivery life cycle (SDLC)

- with IaC, [61](#)
- phases, [15–16](#)
- time sinkhole, [13](#)

## Software packages, [825–826](#)

## Software RAID, [275](#)

## Software repositories

- anatomy, [833–837](#)
- benefits, [827–828](#)
- build environments, [843–845](#)
- case studies, [835](#), [846](#)
- clients, [841–843](#)
- conflicts, [843](#)
- examples, [845–848](#)
- managing, [837–841](#)
- multitiered mirrors and caches, [836–837](#)
- OS mirror, [847–848](#)



- overview, [825–826](#)
- package anatomy, [829–833](#)
- package management systems, [827](#)
- release process, [836](#)
- repackaging public packages, [838](#)
- security, [834–835](#)
- as service, [840–841](#)
- staged, [845–846](#)
- types, [826–827](#)
- universal access, [835](#)
- version management, [841–842](#)

Solid-state drives (SSDs)

- characteristics, [90](#)
- description, [761](#)
- performance, [782](#)
- workstations, [90](#)

Solomon, D., [535](#)

Solution designer role, [1096–1097](#)

Solutions database for documentation, [555–556](#)

Solutions for incident reports

    executing, [517–518](#)

    proposals, [515–516](#)

    selecting, [516–517](#)

Source packages, [832–834](#)

Sources for documentation, [554–556](#)

Space program, [336](#)

Spam, [735](#)

SPARC architecture, [270](#)

Spare capacity for VMs, [232–233](#)

Spare parts for servers, [259–260](#)

Spare time, [943–944](#)

## Spares

- data storage policy, [769–770](#)

- datacenters, [478](#)

- masters and slaves pattern, [322–323](#)

Specific goals in S.M.A.R.T., [338](#)

## Speed

backups, [807–808](#)

disk drive, [785](#)

infrastructure as code, [59](#)

restores, [808–809](#)

server upgrades, [596](#)

Spindles, disk, [760–761](#)

Spinny disks, [761](#)

Split brain situation, [323](#)

Spolsky, J., [936](#), [980](#)

Spool service

email service, [741](#)

print service, [755](#)

Spreadsheets for ready lists, [339–340](#)

SQL Injection, [864](#)

SRM (service request model), [317–318](#)

SSDs. *See* [Solid-state drives \(SSDs\)](#)

SSH (Secure Shell), [93](#)

SSIDs (service set identifiers), [407](#)

SSL (Secure Sockets Layer), [860–861](#)

SSO (single sign-on), [708](#)

## Stack Overflow

datacenters, [413](#)

DNS updates, [73](#)

fail-safe tests, [62](#)

swap space, [274](#)

web page delivery, [1053](#)

workstation standardization case study, [197–198](#)

Stack ranking, [1029](#)

Staff for helpdesks, [488–489](#), [493–494](#)

Staged software repositories, [845–846](#)

Staging areas

- overview, [342](#)

- packages, [843](#)

- ready lists, [342–343](#)

- workstations, [125](#)

Stakeholders in kick-off meetings, [285–286](#)

Stand-ups for ready lists, [341](#)

Standardized templates for configuration management, [435–436](#)

## Standards

corporate, [422–425](#)

data storage, [771–772](#)

email addresses, [731](#)

labeling, [473](#)

open, [298–302](#)

printers, [751–752](#)

web services, [851–852](#)

workstation. *See* [Workstation standardization](#)

Star topology, [414–415](#)

Start-day friendships, [215](#)

Starting out to improve, [20–21](#)

STARTTLS protocol extension, [743](#)

Stateless storage, [89](#)



## Statements

configuration management systems, [66–67](#)

problem, [511–513](#)

## States

blogs, [47](#)

isolating, [44–46](#)

pets and cattle analogy, [43–44](#)

redundancy design patterns, [324–325](#)

workstations, [118–120](#)

Static routing, [419](#)

Static web servers, [852](#)

Static workstation configuration, [85](#)

Statistical data from network sniffers, [442](#)

Statistical improvements from helpdesks, [498–499](#)

## Status

- print service, [755–756](#)

- ticket systems for, [6](#)

- web pages, [925–926](#)

Status meetings for ready lists, [341](#)

Staying calm, [915–916](#)

Staying technical, [968–969](#)

Steps in documentation templates, [553–554](#)

Stevens, W. R., [536](#)

Stoll, Cliff, [539](#)

Storage-area networks (SANs)

- and backups, [777](#)

- caveats, [779–780](#)

- performance, [782](#)

- RAID disks, [764](#)

Storage-needs assessments, [766–768](#)

Storing

data. See [Data storage](#)

ready lists, [339–340](#)

Stranded capacity, [228](#)

Stranded storage, [764](#)

Stress reduction

small batches principle, [33](#)

tips, [1087](#)

Strictly centralized service strategy, [642–643](#)

Strictly confidential data, [722](#)

Striping RAID disks, [762–763](#)

Strong consistency in namespaces, [705–707](#)

Structure for documentation, [561–562](#)

StudyCaps, [558](#)

Subnets datacenter networks, [412](#) vs. VLANs, [400](#) wired office networks, [403](#)

SubVersion VCS, [57](#)

Success criteria in change management plans, [573](#)

Successes in service launch lessons learned, [344](#)

“Successful Strategies for IPv6 Rollouts: Really!” [427](#)

Successive refinement in debugging, [532](#)

Summarization of historical monitoring data, [675](#)

Summary statements, [952–953](#)

Super-packages, [831–832](#)

Supervisors for helpdesks, [501](#)

Supplies for datacenters, [478–479](#)

## Support

- centralization, [654–655](#)

- customer. *See* [Customer care and support](#)

- for firing system administrators, [1014–1015](#)

- for happy system administrators, [965](#)

- network operations, [440–446](#)

- network services, [445–446](#)

- organizational structure, [443–445](#)

- servers, [262](#)

- service planning and engineering, [315–318](#)

- third-party packages, [839–840](#)

- tools, [440–443](#)

- workstation levels, [165–168](#)

Support models for operational requirements, [296–297](#)

Swap space, [273–274](#)

Swimlanes in Kanban, [10](#)

Switch boxes in console, [475–476](#)

Switched fabric networks, [436](#)

Switches

with patch panels, [409–412](#)

VLANs, [404–405](#)

Symptoms vs. causes in debugging, [531](#)

sync command, [784](#)

SysOps death squads, [225–226](#)

## System administrators

attrition, [1080](#)

expectations, [1087–1088](#)

firing. See [Firing system administrators](#)

happy. See [Happy system administrators](#)

hiring. See [Hiring system administrators](#)

influx of, [1079–1080](#)

pleasing, [1078](#)

roles. See [Roles, system administrator](#)

System advocates, [921–925](#)

System clerk role, [1094](#)

System observability requirements, [292–293](#)

System software and applications updating, [123–128](#)

System status for web pages, [925–926](#)

System testing for maintenance windows, [628–630](#)

Systematic debugging, [532–533](#)



## Systems

end-to-end understanding of, [538–539](#)

performance, [1078](#)

Tables of contents in wiki systems, [559](#)

TACACS+ (Terminal Access Controller Access-Control System Plus) protocol, [719](#)

Tack, A. J., [436](#)

Tagging systems for ready lists, [339–340](#)

Tape

backups, [802–808](#), [820–821](#)

costs, [810–811](#)

inventory, [811–812](#)

## Tasks

- maintenance windows, [615–616](#)

- ready lists, [341](#)

Taxonomy for documentation, [561–562](#)

TCO (total cost of ownership)

- grid computing, [236](#)

- workstations, [104–105](#)

*TCP/IP Illustrated*, [536](#)

TCP protocols, debugging, [536](#)

TDD (test-driven development)

- centralizing services, [663](#)

- OS configuration, [146–148](#)

- server upgrades, [594](#)

Team considerations in hiring system administrators, [987–989](#)

## Teams

change management, [581–583](#)

follow-through by, [1084](#)

maintenance window tests, [629](#)

in onboarding, [208–209](#)

roles, [1109–1111](#)

self-confidence, [1084](#)

Technical interviewing, [994–998](#)

“Technical Interviews” article, [980](#)

Technical knowledge in debugging, [534–535](#)

Technical libraries, [562](#)

Technical planning in change management, [573–574](#)

Technical requirements for datacenters, [454–456](#)

Technocrat role, [1102–1103](#)

## Technology

- backups and restores, [820–821](#)

- WANs, [417–418](#)

Technology failures in service launches, [350](#)

Technology staller role, [1107](#)

Technology widows/widowers, [967](#)

Telecom closets, [449](#)

telnet tool, [536](#)

Temperature in datacenters, [477](#)

## Templates

- configuration management, [435–436](#)

- documentation, [553–554](#)

- maintenance window boot sequence, [623](#)

- maintenance window master plans, [621](#)

Temporary fixes, avoiding, [543–545](#)

Tenants, [840–841](#)

Terminal Access Controller Access-Control System Plus (TACACS+) protocol, [719](#)

Terminology for service requirements, [287](#)

Test-driven development (TDD)

- centralizing services, [663](#)

- OS configuration, [146–148](#)

- server upgrades, [594](#)

## Testing

- centralizing services, [663](#)
  - change management plans, [573](#)
  - data migration, [368–369](#)
  - data storage, [784–785](#)
  - DNS/DHCP infrastructure software, [363–366](#)
  - infrastructure as code, [60–62](#)
  - IPv6, [428](#)
  - load, [327](#)
  - maintenance windows, [628–630](#)
  - ordering in CI/CD, [355](#)
  - print service, [756](#)
  - rapid release, [359–360](#)
  - server upgrades, [592–595](#), [603–605](#)
  - service monitoring, [687–688](#)
  - small batches principle, [32](#)
- The slashdot effect, [855](#)
- Theft of workstations, [94–95](#)
- Thematic names, [695](#), [697](#)
- “There’s a S.M.A.R.T. Way to Write Management’s Goals and Objectives,” [338](#)
- Thick clients in VDI, [109](#)
- Thickness in namespaces, [703–704](#)
- Third-party software, repackaging, [839](#)
- Third Way in DevOps, [211](#)
- 30-minute rule for unsupported hardware, [165–167](#)
- Threads, [268–269](#)
- Three Ways in DevOps, [211](#)

Ticket systems

- alerts, [680](#)

- for documentation, [555–556](#)

- overview, [5–8](#)

- ready lists, [339](#)

Tickets in Kanban, [12](#)

Tie-wraps for patch cables, [468–470](#)

Tiered change review boards, [578–579](#)

Tiered support levels, [165–168](#)

Tiered ticket systems, [7](#)



## Time

backups and restores, [807–809](#)

helpdesks, [490](#)

Time-bound goals in S.M.A.R.T., [338](#)

Time coverage for customer support, [499–500](#)

## Time management

courses for, [945](#)

email, [940–941](#)

follow-through, [937](#)

free time, [943–944](#)

goals, [939–940](#)

ineffective people, [944](#)

interruptions, [935–936](#)

precompiling decisions, [942–943](#)

slow bureaucrats, [944–945](#)

to-do lists, [938–939](#)

*Time Management for System Administrators*, [945](#)

Time sinkholes

bottlenecks, [18–20](#)

description, [3–4](#)

eliminating, [12–13](#)

OS installation and configuration, [13–14](#)

software deployment, [15–16](#)

Time-to-call completion, [489](#)

Time to deployment metrics, [361–362](#)

*Time Trap*, [945](#)

Timelines

onboarding, [204–206](#)

service launch lessons learned, [344](#)

Timeouts, [788](#)

Timestamps, [58](#)

## Timing

hiring system administrators, [985–986](#)

maintenance windows, [601–602](#)

## Titles

documentation templates, [553–554](#)

wiki systems, [559](#)

TLS (Transport Layer Security), [860](#)

To-do list management, [938–939](#)

Toigo, J., [387](#)

Toil reduction, [1058–1059](#)

Toner cartridges, [756–757](#)

## Tools

datacenters, [477–479](#)

debugging, [533–537](#)

diagnostics for office networks, [441](#)

evaluating, [537–538](#)

limitations, [535](#)

network path visualization, [443](#)

requirements, [1074–1075](#)

return of, [1075](#)

selecting, [535–536](#)

smart network sniffers, [442](#)

support, [440–443](#)

Tools team in workstation fleet logistics, [180](#)

Top of rack (TOR) switches, [410–412](#)

Top requesters in helpdesks, [498–499](#)

## Topology

- corporate standards, [424](#)

- WANs, [414–416](#)

TOR (top of rack) switches, [410–412](#)

Total cost of ownership (TCO)

- grid computing, [236](#)

- workstations, [104–105](#)

Town hall meetings, [927–929](#)

traceroute tool, [535–536](#)

tracert tool, [536](#)

Track and control WIP, [4](#)



## Tracking

changes in infrastructure as code, [57–59](#)

service contracts, [260–261](#)

Tracks, disk, [760–761](#)

## Training

- customer, [524–525](#)
- customer care, [521–522](#)
- as debugging tool, [534](#)
- flight directors, [632](#)
- professional development, [968](#), [1086](#)
- program office, [182](#)
- service conversions, [379](#)
- service launches, [350](#)
- workstations, [104](#)

## Transitions

- easing, [487](#)
- in workstation standardization, [193–194](#)

Transparency

Kanban, [11](#)

service requirements, [287](#)

Transport layer in OSI model, [400–401](#)

Transport Layer Security (TLS), [860](#)

Transport security vs. end-to-end, [835](#)

Traps in SNMP, [677–679](#)

Trello system

Kanban, [11](#)

ready lists, [339](#)

Trend analysis

- customer care, [523–524](#)

- customer training, [524–525](#)

- maintenance windows, [632–633](#)

Triple-A authentication servers, [719–720](#)

Triple-mirror configuration, [776](#)

Trivia questions in interview process, [995](#)

Trouble-tracking software, [508](#)

Troubleshooting

bugs. See [Debugging](#)

DNS, [439](#)

Trucks in datacenters, [478](#)

True incremental backups, [806](#)

UATs (user acceptance tests)

data migration, [368](#)

software upgrades, [295](#)

staging areas, [342](#)

UCCs (user codes of conduct), [875](#)

UCE (unsolicited commercial email), [735](#)

Ultra-quick requests, [7–8](#)

Unethical orders, [887–888](#), [1085](#)

Unexpected access methods for service launches, [349](#)

Unexplained states, [44](#)

Unified onboarding processes, [48](#)

Unit testing in infrastructure as code, [60](#)

Universal access for software repositories, [835](#)

Universal clients for web browsers, [851](#)

Universities, [909–910](#)

*Unix Backup and Recovery*, [794](#)

Unknown workstation state, [118](#)

Unpleasant processes, documenting, [552](#)

Unrequested solution person role, [1097–1098](#)

Unsolicited commercial email (UCE), [735](#)

Unstable repositories, [845](#)

Untrusted software in VDI, [110](#)

## Updating

- antivirus software, [96](#)
- datacenters, [454–455](#)
- generic processes for, [48](#)
- vs. installations, [123–124](#)
- methods, [125–128](#)
- scheduling, [575](#)
- self-updating software, [369](#)
- system software and applications, [123–128](#)
- web service content, [867](#)
- wiki systems, [562](#)
- workstations, [93–94](#)

## Upgrades

- CMDB for, [184–186](#)

- DNS/DHCP infrastructure software, [363–366](#)
- forklift, [223](#)

- Google servers, [380](#)

- load balancers plus replicas, [323](#)

- operational requirements, [294–295](#)

- servers. *See* [Server upgrades](#)

- software, [436](#), [591–592](#)

- workstations vs. servers, [247](#)

- UPSs for datacenters, [462–464](#)

- Upstream repositories, [827](#)

- Uptime in customer service requirements, [291](#)

- Upward delegation, [974–975](#)

- URL redirectors, [559](#)

- Usability in BYOD, [112–113](#)

- Usable disk capacity, [766](#)

- Used storage space, monitoring, [778](#)

- USENIX LISA conferences, [968](#)

- User acceptance tests (UATs)

  - data migration, [368](#)

  - software upgrades, [295](#)

  - staging areas, [342](#)

- User agency issues, [79–80](#)

- User codes of conduct (UCCs), [875](#)

- User IDs for namespaces, [705–706](#)

- User-initiated updates, [126–127](#)

- User input in web services, [865](#)

- User-prompted updates, [127–128](#)

## Users

- attrition, [1080](#)

- influx of, [1079](#)

- informed consent, [873–874](#)

- service launch training, [350](#)



Vacations, [967](#)

Vagrant system, [144](#)

Validating web input, [865](#)

Value line servers, [266](#)

## Variations

- moving, [51–52](#)

- server hardware strategies, [225](#)

vCPUs (virtual CPUs), [270](#)

VCSs (version control systems), [57–58](#)

- configuration changes, [436](#)

- infrastructure as code, [62](#)

- merge requests, [72–73](#)

VDI. See [Virtual desktop infrastructure \(VDI\)](#)

Vendor-certified designs, [308–309](#)

Vendor liaison role, [1103–1104](#)

Vendor selection

servers, [261](#)–[262](#)

workstations, [102](#)–[103](#)

## Vendors

- engineering recommendations, [306](#)

- mirrors, [827](#)

- OS installation support, [152–154](#)

- service conversion support, [377–378](#)

- SLAs, [258–259](#)

- workstation fleet logistics, [186](#)

Venting about customers, [919](#)

## Verification

- problems, [513–514](#)

- server upgrades, [592–595](#)

Verification phase for incident reports, [518–519](#)

Version control systems (VCSs), [57–58](#)

- configuration changes, [436](#)

- infrastructure as code, [62](#)

- merge requests, [72–73](#)

## Versions

- mixing in server upgrades, [599](#)

- software, [436](#)

- software repositories, [841–842](#)

Vertical integration for datacenters, [450](#)

Vertical power distribution units, [470](#)

Vertical scaling, [857–858](#)

virtual CPUs (vCPUs), [270](#)

Virtual desktop infrastructure (VDI)

- case study, [110](#)

- costs, [106](#)

- maintenance, [106](#)

- overview, [105–106](#)

- persistent vs. non-persistent, [106–109](#)

- untrusted software, [110](#)

Virtual hosting, [853](#)

Virtual LANS (VLANs)

- myths, [404–405](#)

- vs. subnets, [400](#)

Virtual machine managers (VMMs), [269–270](#)

Virtual machines (VMs), [105–106](#)

- containers, [234–235](#)

- CPUs, [269–270](#)

- fixed-size, [42](#)

- live migration, [230](#)

- maintenance, [232–233](#)

- managing, [229](#), [234](#)

- packing, [230–231](#)

- server strategies, [228–235](#)

- spare capacity, [232–233](#)

Virtual private networks (VPNs), [417](#)

Virtual Router Redundancy Protocol (VRRP), [739](#)

Virtualization in programmable infrastructure, [56](#)

## Viruses

email service, [735](#), [742](#)

pirated software, [879](#)

web services, [860](#)

workstations, [95](#)–[96](#)

## Visibility

email, [930–932](#)

lunch, [932](#)

and maintenance windows, [631](#)

management meetings, [926](#)

newsletters, [930](#)

overview, [913](#), [925](#)

physical, [927](#)

system status web pages, [925–926](#)

town hall meetings, [927–929](#)

Visibility paradox, [925](#)

Visionary role, [1104](#)

VLANs (virtual LANs)

myths, [404–405](#)

vs. subnets, [400](#)

VMMs (virtual machine managers), [269–270](#)

VMs. *See* [Virtual machines \(VMs\)](#)

Voice over IP (VoIP), [445–446](#)

Volumes in RAID disks, [763](#)

VPNs (virtual private networks), [417](#)

VRRP (Virtual Router Redundancy Protocol), [739](#)

Vulnerabilities in web services, [863](#)

WANs. *See* [Wide area networks \(WANs\)](#)

Watchlists, [339](#)

Watson, R., [534](#)

Wattage monitors, [788](#)

Watters, J., [387](#)

Weak consistency in namespaces, [705](#)

Web application infrastructure as code example, [68–71](#)

Web pages for system status, [925–926](#)



## Web servers

infrastructure as code, [67–68](#)

multiple, [853–854](#)

simple, [852–853](#)

## Web services

- automation, [867–868](#)
- content management, [866–867](#)
- monitoring, [855](#)
- multiple web servers, [853–854](#)
- overview, [851–852](#)
- scaling, [855–859](#)
- security, [859–866](#)
- simple web servers, [852–853](#)
- SLAs, [854–855](#)

Weekends, changes before, [581–582](#)

Welcome letters, [186](#)

Well-defined exceptions, centralization with, [643](#)

What can get done by Friday? development, [31](#)

*When I Say No, I Feel Guilty*, [961](#), [1099](#)

Whistleblowing, [888–889](#)

Whitelisting workstations, [95](#)

“Why MongoDB Never Worked Out at Etsy,” [350](#)

## Wide area networks (WANs)

- case study, [418–419](#)
- demarcation points, [414](#)
- NOCs, [444](#)
- overview, [413](#)
- technology, [417–418](#)
- topology, [414–416](#)

Wifi. See [Wireless office networks](#)

## Wiki systems

- checklists, [1068](#)
- content-management systems, [560–561](#)
- culture of respect, [561](#)
- documentation, [557–559](#)
- findability, [559](#)
- roll-out issues, [560](#)
- taxonomy and structure, [561–562](#)

## WikiWords, [558](#)

## Willis, J., [17](#)

## Win-win outcomes, [956–957](#)

## Windowed requests, [332](#)

## *Windows Internals*, [534](#)

## *Windows NT Backup and Restore*, [794](#)

## Windows Server Update Services (WSUS), [15](#)

## WIP. See [Work in progress \(WIP\)](#)

## Wired office networks, [402](#)

- emergency services, [405–406](#)
- logical design, [403–404](#)
- NAC, [405](#)
- physical infrastructure, [402–403](#)

## Wireless office networks

- guest restrictions, [407–408](#)
- logical design, [406–408](#)
- physical infrastructure, [406](#)

## Wiring in datacenters, [464–465](#)

## Witthoff, Jim, [687](#)

## Work and personal life, balancing, [966–967](#)

## Work in progress (WIP), [3–4](#)

- Kanban, [8–12](#)
- organizing, [5](#)
- ticket systems, [5–8](#)
- variation handling in, [52](#)

## Work stoppages, [1073–1074](#)

## Workbenches, [476–477](#)

Working sets, memory, [271](#)

Workshops for professional development, [968](#)

Workstation architecture

- accounts and authorization, [86–88](#)

- data storage, [89–93](#)

- elements, [82](#)

- fungibility, [80–82](#)

- hardware, [82](#)

- issues, [79–80](#)

- logging, [97–98](#)

- network configuration, [84–86](#)

- operating system updates, [93–94](#)

- operating systems, [82–84](#)

- security, [94–97](#)

Workstation fleet logistics, [173](#)

- case study, [187](#)

- configuration management database, [183–186](#)

- delivery team, [177–178](#)

- employee view, [173–174](#)

- network team, [179](#)

- platform team, [178](#)

- prep team, [174–177](#)

- program office, [181–182](#)

- project management, [180–181](#)

- purchasing team, [174](#)

- small-scale fleet logistics, [186–188](#)

- tools team, [180](#)

Workstation hardware strategies, [101](#)

- BYOD, [110–113](#)

- laptops vs. desktops, [101–102](#)

- product line selection, [103–105](#)

- VDI. *See* [Virtual desktop infrastructure \(VDI\)](#)

- vendor selection, [102–103](#)

Workstation service definition

- application selection, [159](#)

- basic, [157–160](#)
- case study, [163–164](#)
- CMDB, [160](#)
- overview, [157](#)
- refresh cycles, [161–164](#)
- tiered support levels, [165–168](#)
- workstations as managed services, [168–170](#)

Workstation software life cycle

- changes, [128–129](#)
- disposal, [130–134](#)
- machine life cycle, [117–120](#)
- OS configuration, [120–123](#)
- OS installation, [120](#)
- overview, [117](#)
- updating system software and applications, [123–128](#)

Workstation standardization

- case study, [196–198](#)
- corporate culture, [195–196](#)
- customer involvement, [192–193](#)
- cut-off dates, [195](#)
- feedback, [193](#)
- overview, [191–192](#)
- path of least resistance, [196](#)
- ratcheting, [194–195](#)
- resistance, [192](#)
- transition intervals, [193–194](#)

Workstations vs. servers, [246–249](#)

Worldviews, [988–989](#)

“Would be nice” feature sets, [337](#)

Written service requirements, [286–287](#)

Wrong Fixer stereotype, [520](#)

WSUS (Windows Server Update Services), [15](#)

Your problems, [949](#)

Yum repository systems, [829](#)

## Zones

datacenter networks, [412–413](#)

DNS, [716–717](#)

logical design, [423](#)

Zookeeper lock service, [29](#)

**HAVE WE GOT A CONFERENCE FOR YOU!**



For more than 30 years, LISA has been the premier conference for IT operations, where systems engineers, operations professionals, and academic researchers share real-world knowledge about designing, building, and maintaining the critical systems of our interconnected world.

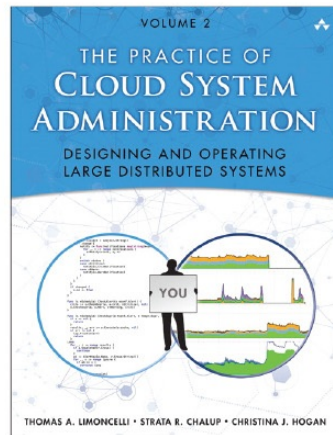
---

***SREcon***<sup>®</sup>

SREcon is a gathering of engineers who care deeply about site reliability, systems engineering, and working with complex distributed systems at scale. The conference has a culture of critical thought, deep technical insights, continuous improvement, and innovation.

To see our complete list of upcoming conferences, go to  
**[www.usenix.org/conferences](http://www.usenix.org/conferences)**





*The Practice of Cloud System Administration* focuses on “distributed” or “cloud” computing and brings a DevOps/SRE sensibility to the practice of system administration. Unsatisfied with books that cover either design or operations in isolation, the authors created this authoritative reference centered on a comprehensive approach.

*The Practice of Cloud System Administration* brings together knowledge and best practices for administering systems in the age of cloud computing, and for architecting, scaling, and operating services that perform reliably and well.

Case studies and examples from Google, Etsy, Twitter, Facebook, Netflix, Amazon, and other industry giants are explained in practical ways that are useful to all enterprises.





## REGISTER YOUR PRODUCT at [informit.com/register](http://informit.com/register) Access Additional Benefits and SAVE 35% on Your Next Purchase

- Download available product updates.
- Access bonus material when applicable.
- Receive exclusive offers on new editions and related products.  
(Just check the box to hear from us when setting up your account.)
- Get a coupon for 35% for your next purchase, valid for 30 days. Your code will be available in your InformIT cart. (You will also find it in the Manage Codes section of your account page.)

Registration benefits vary by product. Benefits will be listed on your account page under Registered Products.

---

### InformIT.com—The Trusted Technology Learning Source

InformIT is the online home of information technology brands at Pearson, the world's foremost education company. At InformIT.com you can

- Shop our books, eBooks, software, and video training.
- Take advantage of our special offers and promotions ([informit.com/promotions](http://informit.com/promotions)).
- Sign up for special offers and content newsletters ([informit.com/newsletters](http://informit.com/newsletters)).
- Read free articles and blogs by information technology experts.
- Access thousands of free chapters and video lessons.

### Connect with InformIT—Visit [informit.com/community](http://informit.com/community)

Learn about InformIT community events and programs.



**informit.com**  
the trusted technology learning source

Addison-Wesley • Cisco Press • IBM Press • Microsoft Press • Pearson IT Certification • Prentice Hall • Que • Sams • VMware Press

## Code Snippets

```
FROM ubuntu:12.04
```

```
RUN apt-get update && apt-get install -y apache2 \
 && apt-get clean && rm -rf /var/lib/apt/lists/*
```

```
ENV APACHE_RUN_USER www-data
```

```
ENV APACHE_RUN_GROUP www-data
```

```
ENV APACHE_LOG_DIR /var/log/apache2
```

```
EXPOSE 80
```

```
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

services:

web:

git\_url: git@github.com:example/node-js-sample.git

git\_branch: test

command: rackup -p 3000

build\_command: rake db:migrate

deploy\_command: rake db:migrate

log\_folder: /usr/src/app/log

ports: ["3000:80:443", "4000"]

volumes: ["/tmp:/tmp/mnt\_folder"]

health: default

api:

image: quay.io/example/node

command: node test.js

ports: ["1337:8080"]

requires: ["web"]

databases:

- "mysql"
- "redis"

| ChangeId: | (Author: | Date:      | Line:) | Code:                                   |
|-----------|----------|------------|--------|-----------------------------------------|
| eae564f2  | (craigp  | 2015-08-06 | 172)   | isRelayed := r.Header.Get(relayHeader)  |
| 67b67bd0  | (mjibson | 2015-04-20 | 173)   | reader := &passthru{ReadCloser: r.Body} |
| 67b67bd0  | (mjibson | 2015-04-20 | 174)   | r.Body = reader                         |
| 67b67bd0  | (mjibson | 2015-04-20 | 175)   | w := &relayWriter{                      |
| 67b67bd0  | (mjibson | 2015-04-20 | 176)   | ResponseWriter: responseWriter}         |
| ba83ae2e  | (craigp  | 2015-09-16 | 177)   | rp.TSDBProxy.ServeHTTP(w, r)            |
| ade02709  | (ipeters | 2016-02-23 | 178)   | if w.code/100 != 2 {                    |
| ade02709  | (ipeters | 2016-02-23 | 179)   | verbose("got status %d", w.code)        |
| 67b67bd0  | (mjibson | 2015-04-20 | 180)   | return                                  |
| 67b67bd0  | (mjibson | 2015-04-20 | 181)   | }                                       |
| 67b67bd0  | (mjibson | 2015-04-20 | 182)   | verbose("relayed to tsdb")              |
| a6d3769b  | (craigp  | 2015-11-13 | 183)   | collect.Add("puts.relayed",             |
| a6d3769b  | (craigp  | 2015-11-13 | 184)   | opentsdb.TagSet{}, 1)                   |

commit ade02709

Author: ipeters <ipeters@example.com>

Date: Tue Feb 23 16:12:39 2016 -0500

cmd/tsdbrelay: accept responses with 2xx code

opentsdb may return 204 or 200 on successful PUTs to /api/put. 204 indicates success with no response body, but when the details parameter is specified, a 200 is returned along with a simple response document. tsdbrelay should relay all successful PUTs to bosun.

I am explicitly not changing relayMetadata, since bosun itself appears to only return a 204.

```
for package_name in ["apache", "openssh", "httpunit", "bosun"] do
 if packageIsInstalled(package_name) then
 print "Package " package " is installed. Skipping."
 else
 if operating_system == "centos" then
 result = yum_install(packagename)
 else if operating_system == "debian" then
 result = apt_install(packagename)
 else
 print "ERROR: Unknown operating system."
 endif

 if error then
 print "ERROR: Package " package " did not install!"
 else
 print "Package " package " installed successfully."
 endfi
 endif
endfor
```

```
required_package("apache")
required_package("openssh")
required_package("wine")
required_package("christmas")
```

```
1 class {'ntpclient':
2 servers => ['ntp1.example.com', 'ntp2.example.com'],
3 }
```



```
1 class { 'apache': }
2
3 apache::vhost { 'first.example.com':
4 port => '80',
5 docroot => '/home/webdata/first.example.com/www/',
6 }
7
8 apache::vhost { 'second.example.com':
9 port => '443',
10 docroot => '/home/webdata/second.example.com/www/',
11 ssl => true,
12 ssl_cert => '/etc/ssl/second.example.com.cert',
13 ssl_key => '/etc/ssl/second.example.com.key',
14 }
```

```
1 site {
2
3 blog { 'beta':
4 db_username => 'betauser',
5 db_password => 'thepassword',
6 nodes => {
7 Node['beta1.example.com'] =>
8 [Blog::Web['beta'], Blog::App['beta']],
9 Node['beta2.example.com'] => Blog::Db['beta'],
10 }
11 }
12
13 blog { 'prod':
14 db_username => 'produser',
15 db_password => 'anotherpassword',
16 nodes => {
17 Node['prod1.example.com'] => Blog::App['prod'],
18 Node['prod2.example.com'] => Blog::Web['prod'],
19 Node['prod3.example.com'] => Blog::Db['prod'],
20 }
21 }
22
23 }
```

```
1 application blog (
2 db_username ,
3 db_password ,
4 docroot = '/var/www/html' ,
5) {
6
7 blog::db { $name :
8 db_username => $db_username ,
9 db_password => $db_password ,
10 export => Sql["blog-$name"] ,
11 }
12
13 blog::web { $name :
14 docroot => $docroot ,
15 export => Http["blog-$name"] ,
16 }
17
18 blog::app { $name :
19 consume => Sql["blog-$name"] ,
20 export => Http["blog-$name"] ,
21 }
22
23 }
```

From: IT

To: Everyone in the company

All servers in the Burlington office are up and running. Should you have any issues accessing servers, please open a helpweb ticket.

From: A Developer

To: IT

Devwin8 is down.

From: IT

To: Everyone in the company

Whoever has devwin8 under their desk, turn it on, please.

We trust you have received the usual lecture from the local System Administrator. It usually boils down to these two things:

- 1) Respect the privacy of others.
- 2) Think before you type.

Subject: No printing in Bld 1 and 2 SATURDAY MORNING

Printing in Buildings 1 and 2 will not work on Saturday, June 24, between 8AM and 11AM because of essential system maintenance.

If this is a problem, please let John Smith know at extension 54321 as soon as possible.

The printing servers in those buildings are approaching the end of their useful lives, and we anticipate that they will become less reliable in a few months. We are replacing them with new hardware now, so that we don't suffer from reliability problems in the future. If you have further questions, please contact us by email at [print-team@company.com](mailto:print-team@company.com).

Dear all,

To serve you better, the system administration team monitors all components of the system. We endeavor to anticipate problems and to fix them before they arise. To do so, we need to occasionally schedule outages of some component of the system to perform maintenance. We do our best to schedule the maintenance at a time that will not adversely affect any urgent work in other parts of the company.

We have identified a potential problem with the print servers in Buildings 1 and 2. We anticipate that those print servers will become less reliable in a few months' time and will then cause problems printing in those buildings. Because of this, we have scheduled some time on Saturday, June 24, between 8AM and 11AM to replace those servers with new, reliable machines. Because of that, you will be unable to print in Buildings 1 and 2 during those hours.

If the timing of this maintenance window will interfere with some urgent work, please let us know as soon as possible and we will reschedule it for another time. John Smith, at extension 54321, is the person to contact about scheduling issues.

As always, we are happy to answer any questions that you might have about this maintenance work. Questions can be addressed directly to the people working on this project by sending email to [print-team@company.com](mailto:print-team@company.com). All other questions should be addressed to [helpdesk@company.com](mailto:helpdesk@company.com) as usual.

Thank you for your cooperation with our ongoing maintenance program.

The SA Team