



AGILE Application Lifecycle Management

USING DEVOPS TO DRIVE PROCESS IMPROVEMENT



BOB AIELLO
LESLIE SACHS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Agile Application Lifecycle Management

This page intentionally left blank

Agile Application Lifecycle Management

*Using DevOps to Drive
Process Improvement*

Bob Aiello and Leslie Sachs

◆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 U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2016936588

Copyright © 2016 Pearson Education, Inc.

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/.

ISBN-13: 978-0-321-77410-1

ISBN-10: 0-321-77410-8

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, June 2016

In loving memory of:

Bob's mother and grandmother, two exceptional women who encouraged him to embrace all of life's challenges and develop an inner compass for the surest path forward,

and

IT expert and dear friend, Ben Weatherall, a pillar in the CM community who was always eager to share his best practices and tirelessly promoted the value of a modified agile–scrum development methodology. As an editor, I appreciated and chuckled along with our readers at the many zany characters he would weave into his articles for CM Crossroads. He was proud of his involvement with both professional associations, such as the IEEE and ASEE (Association of Software Engineering Excellence—The SEI's Dallas-based SPIN Affiliate), and social and charitable organizations. An enthusiastic resident of his Fort Worth, Texas, community, Ben was an active participant in his local Shriners' "Car-vettes" group and could be counted on to lend a hand whenever their presence was requested at an event. Ben was a man of deep faith and, over the years, we had many engaging discussions about matters much more significant than configuration management.

Each of these three individuals was dedicated to balancing a strong work ethic with an equal commitment to their personal relationships; we cherish their legacy.

This page intentionally left blank

Contents

Preface	xxiii
Acknowledgments	xxxix
About the Authors	xli
PART I DEFINING THE PROCESS	1
Chapter 1 Introducing Application Lifecycle Management Methodology.....	3
1.1 Goals of Application Lifecycle Management.....	4
1.2 Why Is ALM Important?	5
1.3 Where Do I Start?	7
1.4 What Is Application Lifecycle Management?	8
1.4.1 Remember the SDLC?	10
1.4.2 Business Focus.....	11
1.4.3 Agile or Not?	15
1.4.4 Mature Process or Fluid?	16
1.4.5 Rapid Iterative Development.....	17
1.4.6 Core Configuration Management Best Practices	17
1.4.7 Automation	21
1.4.8 Continuous Integration	21
1.4.9 Continuous Deployment	22
1.4.10 Change Management	22
1.4.11 IT Operations.....	22
1.4.12 DevOps	23
1.4.13 Retrospectives	23
1.4.14 IT Governance	23
1.4.15 Audit and Regulatory Compliance	24
1.4.16 ALM and the Cloud.....	24
1.4.17 Mainframe	25
1.4.18 Integration across the Enterprise	25
1.4.19 Quality Assurance and Testing.....	25
1.4.20 Role of Personality.....	26
1.5 Conclusion.....	26
References	26

Chapter 2	Defining the Software Development Process	27
2.1	Goals of Defining the Software Development Process.....	27
2.2	Why Is Defining the Software Development Process Important?	28
2.3	Where Do I Start?	29
2.4	Explaining the Software Development Lifecycle.....	29
2.5	Systems versus Software Development Lifecycle	32
2.6	Defining Requirements.....	32
2.6.1	Managing Complexity and Change.....	33
2.6.2	Validity of Requirements.....	34
2.6.3	Testing Requirements	35
2.6.4	Functional Requirements.....	35
2.6.5	Nonfunctional Requirements	36
2.6.6	Epics and Stories	36
2.6.7	Planning for Changing Requirements.....	36
2.6.8	Workflow for Defining Requirements	37
2.7	Test-Driven Development.....	37
2.8	Designing Systems.....	37
2.9	Software Development	38
2.10	Testing	38
2.10.1	Testing the Application	39
2.10.2	Testing the Process Itself.....	39
2.11	Continuous Integration	40
2.12	Continuous Delivery and Deployment	41
2.13	Defining Phases of the Lifecycle.....	41
2.14	Documentation Required	42
2.15	DevOps.....	43
2.16	Communicating with All Stakeholders.....	44
2.17	Production Support.....	45
2.18	Maintenance and Bugfixes	46
2.19	Lifecycle in the Beginning	46
2.20	Maintenance of the Lifecycle	47
2.21	Creating the Knowledge Base.....	47
2.22	Continuous Improvement	48
2.23	Conclusion.....	48

Chapter 3 Agile Application Lifecycle Management.....	49
3.1 Goals of Agile Application Lifecycle Management.....	49
3.2 Why Is Agile ALM Important?	50
3.3 Where Do I Start?	50
3.4 Understanding the Paradigm Shift.....	51
3.5 Rapid Iterative Development.....	52
3.6 Remember RAD?	53
3.7 Focus on 12 Agile Principles	54
3.8 Agile Manifesto.....	56
3.9 Fixed Timebox Sprints	57
3.10 Customer Collaboration	58
3.11 Requirements	59
3.12 Documentation	60
3.13 Conclusion.....	60
Chapter 4 Agile Process Maturity.....	61
4.1 Goals of Agile Process Maturity.....	62
4.2 Why Is Agile Process Improvement Important?	62
4.3 Where Do I Start?	63
4.4 Understanding Agile Process Maturity	64
4.4.1 Adherence to the Principles	65
4.4.2 Repeatable Process	66
4.4.3 Scalability (Scrum of Scrums).....	66
4.4.4 Comprehensive (Items on the Right)	66
4.4.5 Transparency and Traceability	67
4.4.6 IT Governance	67
4.4.7 Coexistence with Non-agile Projects	68
4.4.8 Harmonization with Standards and Frameworks.....	68
4.4.9 Following a Plan	68
4.4.10 Continuous Process Improvement	69
4.5 Applying the Principles	69
4.6 Recognition by the Agile Community	70
4.7 Consensus within the Agile Community	71
4.8 What Agile Process Maturity Is Not	71
4.9 What Does an Immature Agile Process Look Like?	72
4.10 Problems with Agile.....	72

4.11 Waterfall Pitfalls.....	73
4.11.1 Mired in Process.....	74
4.11.2 Pretending to Follow the Process.....	74
4.12 The Items on the Right	75
4.12.1 Adjusting Ceremony	75
4.13 Agile Coexisting with Non-Agile	75
4.14 IT Governance	75
4.14.1 Providing Transparency.....	76
4.15 ALM and the Agile Principles	76
4.16 Agile as a Repeatable Process.....	76
4.16.1 Scalability.....	77
4.16.2 Delivering on Time and within Budget.....	77
4.16.3 Quality	77
4.17 Deming and Quality Management	77
4.17.1 Testing versus Building Quality In.....	77
4.17.2 Productivity.....	78
4.18 Agile Maturity in the Enterprise.....	78
4.18.1 Consistency across the Enterprise.....	78
4.18.2 Marketing the New Approach	79
4.19 Continuous Process Improvement.....	79
4.19.1 Self-Correcting	79
4.20 Measuring the ALM.....	79
4.20.1 Project Management Office (PMO) Metrics.....	80
4.21 Vendor Management.....	80
4.22 Hardware Development	80
4.22.1 Firmware.....	80
4.23 Conclusion.....	81
Chapter 5 Rapid Iterative Development	83
5.1 Goals of Rapid Iterative Development	83
5.2 Why Is Rapid Iterative Development Important?.....	84
5.3 Where Do I Start?	84
5.4 The Development View.....	85
5.5 Controlled Isolation	85
5.6 Managing Complexity	86
5.7 Continuous Integration.....	86
5.8 It's All About (Technology) Risk.....	87
5.9 Taming Technology.....	87

5.10 Designing Architecture	87
5.11 Conclusion	88
Further Reading	88
PART II AUTOMATING THE PROCESS.....	89
Chapter 6 Build Engineering in the ALM.....	91
6.1 Goals of Build Engineering	91
6.2 Why Is Build Engineering Important?	92
6.3 Where Do I Start?	92
6.4 Understanding the Build.....	93
6.5 Automating the Application Build	94
6.6 Creating the Secure Trusted Base	95
6.7 Baselineing.....	96
6.8 Version Identification	97
6.9 Compile Dependencies	98
6.10 Build in the ALM.....	98
6.11 The Independent Build	99
6.12 Creating a Build Robot	99
6.13 Building Quality In	100
6.14 Implementing Unit Tests	100
6.15 Code Scans.....	100
6.16 Instrumenting the Code.....	101
6.17 Build Tools.....	101
6.18 Conclusion	101
Chapter 7 Automating the Agile ALM	103
7.1 Goals of Automating the Agile ALM	103
7.2 Why Automating the ALM Is Important	103
7.3 Where Do I Start?	104
7.4 Tools	104
7.4.1 Do Tools Matter?.....	105
7.4.2 Process over Tools.....	105
7.4.3 Understanding Tools in the Scope of ALM	105
7.4.4 Staying Tools Agnostic	106
7.4.5 Commercial versus Open Source.....	106
7.5 What Do I Do Today?	107
7.6 Automating the Workflow	108
7.7 Process Modeling Automation	108

7.8 Managing the Lifecycle with ALM.....	109
7.9 Broad Scope of ALM Tools.....	109
7.10 Achieving Seamless Integration	109
7.11 Managing Requirements of the ALM.....	110
7.12 Creating Epics and Stories	111
7.13 Systems and Application Design	111
7.14 Code Quality Instrumentation	111
7.15 Testing the Lifecycle.....	112
7.16 Test Case Management	112
7.17 Test-Driven Development.....	113
7.18 Environment Management.....	114
7.18.1 Gold Copies	114
7.19 Supporting the CMDB	115
7.20 Driving DevOps	115
7.21 Supporting Operations.....	116
7.22 Help Desk	116
7.23 Service Desk.....	117
7.24 Incident Management	117
7.25 Problem Escalation	117
7.26 Project Management	118
7.27 Planning the PMO	118
7.28 Planning for Implementation.....	119
7.29 Evaluating and Selecting the Right Tools	119
7.30 Defining the Use Case	119
7.31 Training Is Essential.....	120
7.32 Vendor Relationships	120
7.33 Keeping Tools Current	120
7.34 Conclusion	120
Chapter 8 Continuous Integration.....	121
8.1 Goals of Continuous Integration.....	121
8.2 Why Is Continuous Integration Important?	122
8.3 Where Do I Start?	123
8.4 Principles in Continuous Integration.....	123
8.5 Challenges of Integration	123
8.6 Commit Frequently	124
8.7 Rebase and Build Before Commit	125
8.8 Merge Nightmares	125

8.9 Smaller Units of Integration	126
8.10 Frequent Integration Is Better	126
8.10.1 Easier to Find Issues.....	126
8.10.2 Easier to Fix Problems	126
8.10.3 Fix Broken Builds.....	127
8.11 Code Reviews	127
8.12 Establishing a Build Farm	127
8.12.1 Virtualization and Cloud Computing.....	128
8.13 Preflight Builds.....	129
8.14 Establishing the Build and Deploy Framework	129
8.15 Establishing Traceability	130
8.16 Better Communication	131
8.17 Finger and Blame	133
8.18 Is the Nightly Build Enough?	133
8.19 Selecting the Right Tools.....	134
8.19.1 Selecting the Right CI Server.....	134
8.19.2 Selecting the Shared Repository	135
8.20 Enterprise Continuous Integration	135
8.21 Training and Support.....	136
8.22 Deploy and Test.....	136
8.23 Tuning the Process	137
8.23.1 Getting Lean	137
8.23.2 Interesting Builds.....	138
8.24 CI Leads to Continuous Deployment.....	138
8.25 Conclusion.....	138
Chapter 9 Continuous Delivery and Deployment	139
9.1 Goals of Continuous Deployment.....	139
9.2 Why Is Continuous Deployment Important?.....	140
9.3 Where Do I Start?	141
9.4 Establishing the Deployment Pipeline	141
9.5 Rapid Incremental Deployment.....	143
9.6 Minimize Risk.....	144
9.7 Many Small Deployments Better than a Big Bang.....	145
9.8 Practice the Deploy	146
9.9 Repeatable and Traceable	147
9.10 Workflow Automation.....	148
9.10.1 Kanban—Push versus Pull.....	148

9.11 Ergonomics of Deployments	150
9.12 Verification and Validation of the Deployment.....	150
9.13 Deployment and the Trusted Base.....	151
9.14 Deploy to Environments that Mirror Production.....	152
9.15 Assess and Manage Risk	153
9.16 Dress Rehearsal and Walkthroughs.....	154
9.17 Imperfect Deployments	155
9.18 Always Have a Plan B.....	155
9.19 Smoke Test.....	156
9.20 Conclusion.....	157
PART III ESTABLISHING CONTROLS	159
Chapter 10 Change Management.....	161
10.1 Goals of Change Management.....	161
10.2 Why Is Change Management Important?.....	162
10.3 Where Do I Start?	163
10.4 Traceability for Compliance.....	164
10.5 Assess and Manage Risk	164
10.6 Communication	165
10.7 Change in Application Lifecycle Management	166
10.8 The Change Ecosystem	167
10.9 QA and Testing.....	167
10.10 Monitoring Events	168
10.11 Establishing the Command Center.....	169
10.12 When Incidents Occur.....	170
10.13 Problems and Escalation	172
10.14 The Change Management Process.....	173
10.14.1 Entry/Exit Criteria.....	174
10.14.2 Post-Implementation	175
10.15 Preapproved Changes.....	175
10.16 Establishing the Change Management Function.....	176
10.16.1 Change Control Board	176
10.16.2 Change Advisory Board	176
10.17 Change Control Topology.....	176
10.17.1 A Priori	177
10.17.2 Gatekeeping	177
10.17.3 Configuration Control	178

10.17.4 Emergency Change Control.....	179
10.17.5 Process Change Control	179
10.17.6 E-change Control	179
10.17.7 Preapproved	180
10.18 Coordinating across the Platform	180
10.19 Coordinating across the Enterprise	180
10.20 Beware of Fiefdoms.....	181
10.21 Specialized Change Control	182
10.22 Vendor Change Control	182
10.23 SaaS Change Control	182
10.24 Continuous Process Improvement	183
10.25 Conclusion.....	184
Chapter 11 IT Operations	185
11.1 Goals of IT Operations	185
11.2 Why Is IT Operations Important?	186
11.3 Where Do I Start?	186
11.4 Monitoring the Environment	188
11.4.1 Events.....	188
11.4.2 Incidents.....	189
11.4.3 Problems	190
11.5 Production Support.....	191
11.6 Help Desk	192
11.6.1 Virtual Help Desks.....	193
11.6.2 Remote Work	194
11.6.3 Virtual World Help Desk	194
11.6.4 Developers on the Help Desk.....	195
11.7 IT Process Automation.....	195
11.7.1 Knowledge Management.....	195
11.8 Workflow Automation	196
11.9 Communication Planning.....	197
11.9.1 Silos within the Organization.....	197
11.10 Escalation	198
11.10.1 Level 1	198
11.10.2 Level 2	199
11.10.3 Level 3	199
11.11 DevOps.....	200

11.12	Continuous Process Improvement	200
11.13	Utilizing Standards and Frameworks	201
11.13.1	ITIL v3.....	201
11.13.2	Knowledge Management.....	204
11.13.3	ISACA Cobit	205
11.14	Business and Product Management.....	205
11.15	Technical Management	206
11.16	IT Operations Management	206
11.17	IT Operations Controls.....	206
11.17.1	Facilities Management.....	207
11.18	Application Management.....	208
11.18.1	Middleware Support	208
11.18.2	Shared Services.....	208
11.19	Security Operations.....	208
11.19.1	Center for Internet Security	209
11.19.2	Outsourcing	209
11.20	Cloud-Based Operations	209
11.20.1	Interfacing with Vendor Operations.....	209
11.21	Service Desk.....	210
11.21.1	Centralized.....	210
11.21.2	Virtual.....	211
11.21.3	Specialized.....	211
11.21.4	Vendor Escalation	211
11.22	Staffing the Service Desk.....	211
11.23	Incidents and Problems	212
11.24	Knowledge Management	212
11.25	Conclusion.....	212
Chapter 12	DevOps.....	213
12.1	Goals of DevOps.....	213
12.2	Why Is DevOps Important?	214
12.3	Where Do I Start?	214
12.4	How Do I Implement DevOps?.....	215
12.5	Developers and Operations Conflict	216
12.6	Developers and Operations Collaboration	216
12.7	Need for Rapid Change	218
12.8	Knowledge Management.....	219
12.9	The Cross-Functional Team	220

12.10 Is DevOps Agile?.....	221
12.11 The DevOps Ecosystem.....	222
12.12 Moving the Process Upstream	223
12.12.1 Left-Shift	223
12.12.2 Right-Shift.....	224
12.13 DevOps in Dev.....	224
12.14 DevOps as Development.....	225
12.14.1 Deployment Pipeline	226
12.15 Dependency Control	227
12.16 Configuration Control	228
12.17 Configuration Audits	228
12.18 QA and DevOps	229
12.19 Information Security	229
12.20 Infrastructure as Code.....	229
12.21 Taming Complexity.....	230
12.22 Automate Everything	230
12.23 Disaster Recovery and Business Continuity.....	230
12.24 Continuous Process Improvement	231
12.25 Conclusion.....	231
Chapter 13 Retrospectives in the ALM	233
13.1 Goals of Retrospectives.....	234
13.2 Why Are Retrospectives Important?	234
13.3 Where Do I Start?	234
13.4 Retrospectives as Process Improvement	235
13.4.1 Start with Assessing Success	235
13.4.2 Incidents and Problems	236
13.4.3 Mistakes Are Good	237
13.4.4 Personality and Disposition.....	237
13.4.5 Don't Just Tell Me What I Want to Hear	238
13.5 Which Mode Should You Use?	238
13.5.1 In Person Is Best.....	238
13.5.2 Online and Video Conferencing	239
13.5.3 Teleconference	239
13.5.4 Virtual Worlds.....	239
13.6 Perspective Is Essential.....	240
13.6.1 Developers.....	240
13.6.2 Customers	240

13.6.3 Tester	240
13.6.4 Operations	241
13.7 DevOps: The Cross-Functional View	241
13.8 Understanding the Use Case	241
13.8.1 Epics and Stories	241
13.9 Retrospectives as Leadership.....	241
13.9.1 Removing Barriers.....	241
13.10 Running the Meeting	242
13.10.1 Probing and Questioning.....	242
13.11 Retrospectives Supporting ITIL.....	242
13.11.1 Incidents.....	242
13.11.2 Problems	243
13.12 Retrospectives and Defect Triage	243
13.13 Retrospectives as Crisis Management	243
13.14 Supporting IT Governance	244
13.15 Audit and Regulatory Compliance.....	244
13.16 Retrospectives as Risk Management	244
13.17 Vendor Management.....	244
13.18 Too Much Process.....	245
13.19 Corporate Politics	245
13.20 Metrics and Measurement.....	245
13.21 Conclusion.....	246
PART IV SCALING THE PROCESS	247
Chapter 14 Agile in a Non-Agile World	249
14.1 Goals of Hybrid Agile.....	249
14.2 Why Is Hybrid Agile Important?.....	250
14.3 Where Do I Start?	250
14.4 Pragmatic Choices.....	251
14.5 The Best of Both Worlds	251
14.6 Keeping It Agile	252
14.7 Establishing the Agile Pilot	253
14.8 Transitioning to Agile	253
14.9 Having a Baby	254
14.10 The Elephant in the Room.....	254
14.11 Are We There Yet?	255
14.12 Agile Disasters	255

14.13 Developer View.....	256
14.14 No Information Radiators Allowed	256
14.15 Waterfall Is Iterative, Too	256
14.16 Document Requirements as Much as Possible	257
14.17 Last Responsible Moment.....	257
14.18 Technology Risk.....	257
14.19 Understanding the Ecosystem.....	257
14.20 Mature Agile.....	258
14.21 Meeting IT Governance Requirements.....	258
14.22 Conclusion.....	259
Chapter 15 IT Governance.....	261
15.1 Goals of IT Governance.....	261
15.2 Why Is IT Governance Important?.....	262
15.3 Where Do I Start?	262
15.4 Senior Management Makes Decisions.....	263
15.5 Communicating Up.....	264
15.6 How Much Work Is Going On?.....	265
15.7 Identify and Manage Risk.....	266
15.8 Time and Resources	267
15.9 Scalability with More Resources	268
15.10 Delays Happen.....	268
15.11 The Helicopter Mom	269
15.12 I Told You That Already	269
15.13 Learning from Mistakes.....	270
15.14 Governance Ecosystem.....	270
15.15 Continuous Process Improvement	270
15.16 Governance and Compliance	271
15.17 Conclusion.....	271
Chapter 16 Audit and Regulatory Compliance.....	273
16.1 Goals of Audit and Regulatory Compliance	273
16.2 Why Are Audit and Regulatory Compliance Important?	274
16.3 Where Do I Start?	274
16.4 Compliance with What?.....	275
16.5 Establishing IT Controls	275
16.6 Internal Audit	276

16.7 External Audit	277
16.8 Federally Mandated Guidelines.....	278
16.8.1 Section 404 of the Sarbanes-Oxley Act of 2002.....	278
16.8.2 Financial Industry Regulatory Authority	280
16.8.3 Health Insurance Portability and Accountability Act of 1996.....	280
16.8.4 ISACA Cobit	281
16.8.5 Government Accountability Office.....	281
16.8.6 Office of the Comptroller of the Currency (OCC) ...	282
16.9 Essential Compliance Requirements.....	283
16.10 Improving Quality and Productivity through Compliance	283
16.11 Conducting an Assessment.....	283
16.12 Conclusion.....	284
Chapter 17 Agile ALM in the Cloud	285
17.1 Goals of ALM in the Cloud	285
17.2 Why Is ALM in the Cloud Important?.....	286
17.3 Where Do I Start?	286
17.4 Understanding the Cloud	287
17.5 Developing in the Cloud	288
17.5.1 Source Code Management in the Cloud.....	288
17.5.2 Build Automation in the Cloud.....	289
17.5.3 Release Engineering in the Cloud	289
17.5.4 Deployment in the Cloud	290
17.6 Change Management in the Cloud.....	290
17.6.1 Service Provider Notification.....	291
17.7 Managing the Lifecycle with ALM.....	292
17.8 Cloud-based ALM Tools.....	292
17.9 Achieving Seamless Integrations	292
17.10 Iterative Development in the Cloud	293
17.10.1 Development Models in SaaS.....	293
17.11 Interfacing with Your Customers	293
17.11.1 Fronting Service Providers.....	294
17.12 Managing with SLAs.....	294
17.12.1 Reliance upon Service Providers.....	294
17.13 Managing Cloud Risk.....	294

17.14 Development and Test Environments for All.....	295
17.14.1 Starting Small.....	295
17.15 Environment Management.....	295
17.15.1 Gold Copies	295
17.15.2 CMDB in the Cloud.....	296
17.16 DevOps in the Cloud	296
17.17 Controlling Costs and Planning	296
17.18 Conclusion.....	297
Chapter 18 Agile ALM on the Mainframe	299
18.1 Goals of Agile ALM on the Mainframe	299
18.2 Why Is Agile ALM on the Mainframe Important?.....	299
18.3 Where Do I Start?.....	300
18.4 DevOps on the Mainframe	302
18.5 Conclusion.....	303
Chapter 19 Integration across the Enterprise.....	305
19.1 Goals of Integration across the Enterprise	305
19.2 Why Is Integration across the Enterprise Important?	305
19.3 Where Do I Start?.....	306
19.4 Multiplatform	307
19.5 Coordinating across Systems.....	307
19.6 Understanding the Interfaces.....	307
19.7 The Enterprise Ecosystem	308
19.8 Release Coordination.....	308
19.9 Conclusion.....	308
Chapter 20 QA and Testing in the ALM.....	309
20.1 Goals of QA and Testing	309
20.2 Why Are QA and Testing Important?	309
20.3 Where Do I Start?.....	310
20.4 Planning the Testing Process	311
20.5 Creating the Test Cases.....	313
20.6 Ensuring Quality.....	313
20.7 Ensuring Quality from the Beginning.....	313
20.8 Conclusion.....	314

Chapter 21 Personality and Agile ALM.....	315
21.1 Goals of Personality and the Agile ALM.....	315
21.2 Why Are Personality and Agile ALM Important?	315
21.3 Where Do I Start?	316
21.3.1 Understanding the Culture	316
21.3.2 Probing Deeper into the Organization’s Psyche.....	318
21.4 Group Dynamics.....	320
21.4.1 Using DevOps to Drive Out Silos.....	320
21.4.2 Managing Power and Influence in DevOps	321
21.5 Intergroup Conflict	323
21.5.1 Overly Agreeable People and Other Challenges	323
21.5.2 Learned Helplessness.....	325
21.5.3 Introspection and the Postmortem	327
21.6 Managing Stress and Dysfunctional Behavior	329
21.6.1 The Danger of Learned Complacency	329
21.6.2 Dealing with Aggressive Team Members.....	331
21.6.3 Extremism in the Workplace	333
21.7 Taking a Positive Approach	335
21.7.1 How Positive Psychology Can Help Your Organization	335
21.7.2 Three Pillars of Positive Psychology	337
21.7.3 Using Positive Psychology to Motivate Your Team ...	339
21.7.4 Learning from Mistakes	340
21.7.5 Positive Psychology in DevOps	342
21.8 Conclusion.....	344
References	344
Further Reading.....	345
Chapter 22 The Future of ALM.....	347
22.1 Real-World ALM	347
22.2 ALM in Focus.....	348
22.3 Conclusion.....	349
Index.....	351

Preface

This is an amazing, and perhaps chaotic, time to be involved with the technology industry. The demand for talent, skills, and commitment to excellence has never been higher. Developing software and systems has become a remarkably complex task, with many factors affecting the success of the development effort. Learning new development frameworks and adapting legacy systems to meet the need for continued growth and flexibility require the modern IT professional to be able to press forward, while understanding the limitations imposed by earlier conditions. Teams may be located in one specific “war” room or distributed across the globe and frequently working at different hours of the day, with varying languages, cultures, and expectations for how they will operate on a daily basis. The project itself might involve writing complex application software or customizing a vendor package as part of a systems (versus software) development effort. The competition for specialized technical resources motivates many organizations to allow flexible work arrangements, including telecommuting along with choosing office locations convenient to attract local candidates. Technology professionals must often choose between the demands of high-paying (and often stressful) opportunities and trying to maintain a comfortable work-life balance. The Internet has clearly become the backbone of commerce, and companies are expected to continuously align themselves with growing Web capabilities in order to achieve and maintain success.

Pragmatic Focus

This book focuses on the real world of creating and implementing processes and procedures to guide your software and systems delivery effort. The views expressed in these pages may make you feel uncomfortable, especially if you view yourself as an agile purist. We are going to challenge assumptions regarding the way things are being done today, and we are going to encourage you to participate in a discussion on how we can do a better job of developing software and systems. We are going to stipulate up front that our views may not always be applicable in every

possible situation, but all that we write is based upon our real-world experiences or that which we have heard about from reliable sources. This is not a “feel-good” book about agile. This is a book about creating processes and procedures to guide you in overcoming the day-to-day challenges of developing complex software and systems. We look forward to hearing from you as you read through these chapters!

Successful organizations need to support complex technologies, most often with a significant Web presence. Even companies going out of business are expected to have a functioning Web presence capable of handling the peak transaction requirements of customers and other users. In practice, these complex development efforts necessitate effective processes and methodologies to meet both the demands of today and those that will surface in the future. This book will describe best practices for designing the appropriate application lifecycle management (ALM) processes necessary to successfully develop and implement complex software systems, whether your team is writing the code or customizing a system that you have purchased from a vendor. We will discuss both agile and non-agile methodologies to empower the reader to choose the best approach for your organization and the project that you are trying to complete. Our goal is to increase and enhance the reader’s understanding and ability to apply these principles and practices in your own environment. We often work in the imperfect world of having to support lifecycle methodologies that are not always optimal. In fact, we are usually called in when things get really bad and an organization needs to figure out how to incrementally improve processes to improve quality and productivity. In our opinion, the most effective methodology to emerge in the last decade has been agile.

Agile configuration management and, by extension, agile application lifecycle management have become two of the most popular software development methodologies in use today. Agile has resulted in indisputable successes boasting improved productivity and quality. My 25-year (and counting) career has always involved software process improvement with a particular focus on configuration management. As a practitioner, I am completely tools and process agnostic. I have seen projects that successfully employed agile methods and other efforts that thrived using an iterative waterfall approach. Still, *all* organizations need a reliable and repeatable way to manage work, allowing full traceability and clear, complete communication. Years ago, the IT community looked to the software development lifecycle (SDLC) to guide us in understanding what needed to be done by each member of the team on a daily basis, although the SDLC process

documentation often sat on the shelf along with the outdated requirements specification from the latest software or systems development effort. When purchasing commercial off-the-shelf (COTS) products became popular, we began to use the term systems development lifecycle to refer to the work, at times spanning months or even years, to customize and configure COTS systems. We will discuss the differences between software and systems lifecycles further in Chapter 1. Whether applied to software development or systems customization and configuration, the SDLC, in practice, generally only referred to the required activities to create and maintain the system. Some vendors marketed efforts to customize and configure their solution as production lifecycle management (PLM) solutions. Many companies struggled with improving programmer productivity, and some tried to use the Software Engineering Institute's (SEI) Capability Maturity Model (CMM). These efforts often had limited success, and even those that succeeded had limited return on their investment due to the excessive cost and effort involved. The SEI chartered a series of Software Process Improvement Networks (SPINs) throughout the United States, which provided speakers and opportunities to meet with other professionals involved with software process improvement. I had the pleasure of serving for many years on the steering committee of one of the SPINs located in a major city. Today, most of the SPIN presentations focus on agile practices, and most of the attendees are interested in establishing scrums, iterative development, and agile testing. Agile has certainly had a major impact on software process improvement, although not without its own inherent challenges and limitations. Application lifecycle management has emerged as an essential methodology to help clarify exactly how software development is conducted, particularly in large-scale distributed environments. ALM typically has a broader focus than was considered in scope for an SDLC and helped to resolve many of the most common challenges, such as providing a comprehensive software development methodology helping each member of the team understand what needed to be done on a daily basis. At its core, the ALM enhances collaboration and communication. DevOps is a closely related approach that is particularly effective at driving the entire ALM.

DevOps and the ALM

DevOps is a set of principles and practices that improve communication between the development and operations teams. Many DevOps thought leaders acknowledge that DevOps is also effective at helping development interact with other groups, including quality assurance (QA) and information security

(InfoSec). In this book, we will broaden that definition to show that DevOps is essential to enhancing communication between every other group that participates in the ALM. We will be discussing DevOps throughout this book and in detail in Chapter 12. DevOps principles and practices are applicable to the interactions between any two or more groups within the organization and are essential in driving the ALM.

The initial goal of any ALM is to provide the transparency required to enable decision makers to understand what needs to be done and, most importantly, approve the project, including its budget and initial set of goals and objectives. Providing this transparency is precisely where IT governance plays an essential role in helping to get the project approved and started.

IT Governance

Effective software methodology today must have a strong focus on IT governance, which is essentially the control of the organizational structures through effective leadership and the hands-on management of organizational policies, processes, and structures that affect information, information-related assets, and technology. Fundamentally, IT governance provides the guidance necessary to ensure that the information technology organization is performing successfully and that policies, processes, and other organizational structures are in place so that essential organizational strategies and objectives are achieved. Organizations with excellent IT governance enjoy improved coordination, communication, and alignment of goals throughout the entire enterprise. IT governance is closely related to, and must align with, corporate governance in order to ensure that information technology can help drive the business to success and profitability. The initial goals of IT governance are to define policies, clarify the objectives of corporate governance, and ensure that the information technology organization aligns with the business to provide essential services that enable the business to achieve its goals. From an IT service management perspective, IT governance helps drive the development and deployment of services that help achieve value; these include fitness for purpose (utility) and fitness for use (warranty). IT governance is also concerned with establishing the most efficient organizational structure that will allow technology to be delivered successfully as a valued corporate asset. In this context, management is also responsible for providing adequate resources while maintaining necessary budget and financial controls.

IT governance cannot exist in a vacuum. Management requires accurate and up-to-date information in order to make the best possible decisions. Department

managers and teams must provide valid and relevant information so that management understands the risks, challenges, and resources required for success. IT governance enables the business by ensuring that informed decisions are made, that essential resources are available, and that barriers to success are removed or identified as risks. Risk management is essential to effective IT governance. Risk is not always bad, and many organizations thrive on well-defined risk. IT governance provides the essential information that is needed to enable senior management to identify and mitigate risk so that the organization can successfully operate within the global business environment.

IT governance has the unique view of seeing the organization as part of an ecosystem, with the focus on competitors and outside forces, including regulatory requirements that affect the business and business objectives. Information security and business continuity are special areas of focus for IT governance, as it is essential to ensure that the business can operate and thrive regardless of challenges, such as competitive forces and other external pressures. Other considerations of IT governance include data privacy, business process engineering, and project governance.

Closely related to IT governance, and often mentioned in the same sentence, is compliance with regulatory requirements, industry standards, and internal audit requirements. IT governance helps all relevant stakeholders within the entire organization understand what they need to do in order to meet and comply with all regulatory requirements. Effective IT governance enables businesses to implement organizations with organizational structures that operate successfully, while providing the necessary information to help senior management make the decisions, which then propel the organization to achieve improved quality, productivity, and profitability. With this guidance from senior management, the next step is to ensure that all of the stakeholders understand their roles and what needs to be done on a day-to-day basis. This is exactly where application lifecycle management comes into the picture.

Application Lifecycle Management

Application lifecycle management (ALM) evolved from the early days of process improvement to provide a comprehensive software development methodology that provides guidance from requirements gathering, to design development, all the way through to application deployment. In practice, ALM takes a wide focus, with many organizations establishing an ALM to manage their entire software and systems delivery effort. Even nondevelopment functions such as operations and the help desk can benefit from a well-defined ALM. Some

organizations implement ALM in a way that would not be considered agile, using a waterfall model that has a heavy focus on completing the tasks in each phase before moving on to the next. Configuration management, consisting of source code management, build engineering, environment configuration, change control, release management, and deployment, has been a key focus of ALM for some time now. Another central theme has been applying agile principles to support and improve configuration management functions.

Agile CM in an ALM World

Agile configuration management (CM) provides support for effective iterative development, including fast builds, continuous integration, and test-driven development (TDD), that is essential for successful agile development. In a comprehensive lifecycle methodology, agile CM can make the difference between success and failure.

The Definition of Agile ALM

Agile ALM is a comprehensive software development lifecycle that embodies the essential agile principles and provides guidance on all activities needed to successfully implement the software and systems development lifecycle. Agile ALM embodies agile CM and much more. Agile ALM starts with tracking requirements with “just-enough process” to get the job done without any extra steps, or what agile enthusiasts often call “ceremony.” This is often accomplished by creating user stories, which need to be under version control just like any other artifact. Testing throughout the lifecycle also plays a significant role in agile ALM and may even be used to supplement requirements documents that are often intentionally kept brief in an agile world. Agile ALM focuses on iterative development that requires a minimum amount of process, with an emphasis on proven practices that include iterative development, strong communication, and customer collaboration. Understanding agility is much easier when we examine the process methodologies that have come before.

Understanding Where We Have Come From

Understanding where we have come from should always start with reviewing the essential principles of process improvement. For example, most practitioners will confirm that process improvement needs to be iterative, pragmatic, and continuous. One excellent source of valid principles for process improvement may

be found in the work of W. Edwards Deming. Many of Deming's teachings¹ provide principles that are practical and form the basis of quality management.

Principles of Process Improvement

Process engineering focuses on defining the roles, responsibilities, and essential tasks that need to be accomplished in order for the process to be completed successfully. Processes themselves need to be understood, clearly defined, and communicated to all stakeholders. Complex processes are most often created in a collaborative way and usually take several iterations before they are comprehensive or complete. Processes may need to change over time and may be loosely defined early in the lifecycle, but usually require greater clarity and discipline as the target delivery date approaches. Too much process is just as bad as not enough. Therefore, the best processes are Lean with few, if any, extra unnecessary steps. Quality must be built into the process from the very beginning, and it is essential to maintain an honest and open culture to achieve effective processes and process improvement.

Mired in Process

Bob worked in an international financial services firm that was deeply mired in process. The CEO of the company once commented in a town hall meeting that they realized they had too much process, and their solution was, unfortunately, to add more process. The organization had a deeply held belief in process, which also had a high degree of ceremony. The dark unspoken secret, however, was that many people simply chose to work around the burdensome processes, which required far too many documents. Most people in the organization become quite clever at gaming the system to deal with the burdensome requirements of the organizational processes. But because the culture was so focused on process, it was considered disloyal to complain or attempt to push back on this. The organization wanted to grow, but just about every effort took far too long to complete.

1. Deming, W. Edwards. (1982). *Out of the Crisis*. Cambridge, MA: Massachusetts Institute of Technology, Center for Advanced Engineering Study.

Right-sizing processes is essential for organizational success, as is effective communication. Application lifecycle management has its own terminology, which needs to be understood for effective communication among all stakeholders.

Terminology

Every effort has been made to use terms that are consistent with industry standards and frameworks. Please contact us via social media with any questions or via the website for this book as noted on the next page.

Use of “I” versus “We”

Although we do everything as a team, there were quite a few places where it was much easier to write in the first-person singular. Bob is also much more technical and “hands-on” than Leslie, so when you see first-person singular “I” or “my” you can safely assume that this is a first-person account from Bob.

Why I Write About Agile CM, DevOps, and Agile ALM

Agile configuration management and agile application lifecycle management provide the basis for essential best practices that help a software or system development team improve their productivity and quality in many significant ways. DevOps and the agile ALM help ensure that teams can produce systems that are reliable and secure while maintaining high levels of productivity and quality. As is often the case, early life experiences have greatly shaped my view of the world.

Blindness and Process Improvement

Much of how I have approached my life and career has been influenced by the fact that I had a significant visual handicap growing up that could not be safely corrected until I was in my late teens. Consequently, I used Braille, a white cane, and lots of taped recordings (“talking books”). Even when I gained useable vision, at first it was only for short amounts of time because my eyes would fatigue quickly, and then for all practical purposes I would be temporarily blind again (or what we blind guys like to refer to as “blinking” out). My beloved ophthalmologist, Dr. Helen Grady Cole, once noted that my handicap *made* me successful because I learned to achieve against all odds and “move mountains” when necessary. No doubt, you will hear some of that fierce determination in these

pages. I am very comfortable when approaching the seemingly impossible and viewing it as quite doable. You will get to hear about some of my experiences in the motorcycle gang of para- and quadriplegics with whom I proudly associated during my most formative years.

Classroom Materials

University professors who would like to use our book for a class in software engineering or software methodology are encouraged to contact us directly. We are glad to review classroom materials and would guest lecture (via Skype where travel is impractical) if appropriate and feasible. Obviously, we are glad to answer any and all questions related to the material in the book.

Website for this Book

Please register on our website at <http://agilealmdevops.com> and connect with us on social media to engage in discussions on Agile ALM and DevOps!

Who Should Read This Book

This book will be relevant for a wide variety of stakeholders involved in application lifecycle management.

Development managers will find guidance regarding best practices that they need to implement in order to be successful. We also discuss the many people issues involved with managing the software development process.

How This Book Is Organized

This book is organized into 22 chapters divided into four parts. Part I consists of five chapters defining the software development process, agile ALM and agile process maturity, and rapid iterative development. Part II covers automation, including build engineering, automating the ALM, continuous integration, delivery, and deployment. Part III covers establishing essential IT controls, including change management, operations, DevOps, retrospectives, agile in non-agile environments, IT governance, and audit and regulatory compliance. Part IV covers scalability, including integration across the enterprise, agile ALM in the cloud, ALM on the mainframe, QA and testing, personality, and the future of ALM.

Part I: Defining the Process

Chapter 1: Introducing Application Lifecycle Methodology

This chapter introduces application lifecycle management by explaining what you need to know in order to define an ALM that will help you implement a comprehensive and effective software or systems lifecycle. We discuss how to implement the ALM using agile principles in a real-world, pragmatic way that will help guide the activities of each member of your team, whether you are creating new software or customizing a commercial package. Systems lifecycles are a little different than a software development lifecycle and are usually associated with obtaining (and customizing) a project from a solution vendor. Commercial off-the-shelf (COTS) software is commonly used today to deliver robust technology solutions, but they often require some effort to customize and implement. In this chapter, we introduce the core concepts and then build upon them throughout the rest of the book.

Chapter 2: Defining the Software Development Process

This chapter helps you understand the basic skills of how to define the software development process. Defining the software development process always sounds straightforward until you actually start trying to do the work. Many tasks are involved with any software or systems lifecycle, and a well-defined process must provide guidance on exactly what needs to get done and who is responsible for completing each task.

Chapter 3: Agile Application Lifecycle Management

In this chapter we discuss the core strategies that will help you create a flexible and robust software and systems development lifecycle while ensuring that the values and principles of agility are understood and maintained.

Chapter 4: Agile Process Maturity

In this chapter, we will examine the factors that affect agile process maturity from a number of different perspectives. Many technology professionals find that they must implement agile processes in a large organizational context, including managing teams that are composed of many scrums, totaling scores or even hundreds of developers working from a variety of locations. Scalability is certainly an essential aspect of agile process maturity. Mature agile processes must be repeatable for each project in the organization and have sufficient support for project planning. We also need to understand how process maturity affects non-agile development methodologies, including waterfall and other process models.

Chapter 5: Rapid Iterative Development

In this chapter, we discuss rapid iterative development and its impact on software methodology that came long before agile development reached its level of popularity common today. We consider what we learned from rapid iterative development and how it may be applied in practical situations today.

Part II: Automating the Process

Chapter 6: Build Engineering in the ALM

This chapter helps you understand the build within the context of application lifecycle management. We discuss essential aspects of automating the application build, with particular attention on techniques for creating the trusted application base. We also discuss baselining, compile dependencies, and embedding version IDs as required for version identification. We discuss the independent build and creating a fully automated build process. Building quality into the build through automated unit tests, code scans, and instrumenting the code is an important part of this effort. Finally, we will discuss the ever-challenging task of selecting and implementing the right build tools.

Chapter 7: Automating the Agile ALM

In this chapter we discuss how application lifecycle management touches every aspect of the software and systems lifecycle. This includes requirements gathering, design, development, testing, application deployment, and operations support. Automation plays a major role in the agile ALM, which sets it apart in many ways from other software development methodologies. We also explain why it is essential to appreciate the big picture at all times so that the functions that you implement are in alignment with the overall goals and structure of your ALM.

Chapter 8: Continuous Integration

In this chapter we explain that continuous integration (CI) is an essential practice that involves putting together code that has been created by different developers and ascertaining if the code components can compile and run together successfully. CI requires a robust automated testing framework, which we will discuss further in Chapter 20 and provides the basis for ensuring code quality through both static and instrumented code scanning. Continuous integration often involves merging together code that has been written by different developers and is essential for code quality. The fundamental value of this practice is that integrating a small amount of code as early as possible can avoid much

bigger issues later. It would be difficult to imagine an effective ALM that does not embrace integrating code frequently, although I will also discuss a couple of situations where it is difficult or even impossible to achieve. When code cannot be integrated early and often, there is increased risk, which must be identified and addressed. It is also important to understand that continuous integration relies upon many other practices, including continuous testing, effective build engineering, static and instrumented code analysis, and continuous deployment, discussed in Chapter 9.

Chapter 9: Continuous Delivery and Deployment

In this chapter we explain how continuous deployment (CD) is a methodology for updating production systems as often as necessary and generally in very small increments on a continuous basis. It would be difficult to understand continuous deployment without discussing continuous integration and delivery. The terminology for CD has been confusing at best, with many thought leaders using the terms *continuous delivery* and *continuous deployment* interchangeably. We discussed continuous integration in Chapter 8. Continuous delivery focuses on ensuring that the code baseline is always in a state of readiness to be deployed at any time. With continuous delivery, we may choose to perform a *technical deployment* of code without actually exposing it to the end user, using a technique that has become known as *feature toggle*. Continuous deployment is different from continuous delivery in that the focus is on immediate promotion to a production environment, which may be disruptive and a poor choice from a business perspective. We will help you find the right balance so that you can support your business by promoting changes to production as often as desired.

Part III: Establishing Controls

Chapter 10: Change Management

In this chapter we examine how change management is a broad function that helps us plan, review, and communicate many different types of planned and emergency (unplanned) system modifications. Changes may be bugfixes or new features and can range from a trivial configuration modification to a huge infrastructure migration. The goal of change control is to manage all changes to the production (and usually QA) environments. Part of this effort is just coordination, and that is very important. But part of this effort is also managing changes to the environment that could potentially affect all of the systems in the environment. It is also essential to control which releases are promoted to QA and

production. Change control can act as the stimulus to all other configuration management–related functions as well. Throughout this chapter we will discuss how to apply change management in the ALM.

Chapter 11: IT Operations

In this chapter, we will discuss how to create an effective IT operation group that is aligned with your agile ALM. The IT operations organization is responsible for maintaining a secure and reliable production environment. In large organizations, operations often resembles a small army with too many divisions to navigate that is also often held responsible when things go wrong. Developers, working on the bleeding edge of technology, often regard their colleagues in operations as lacking technical skills and ability, which is true in so far as operations resources tend to focus more on the day-to-day running of the systems. Understanding these different perspectives is a key aspect of our DevOps approach to the agile ALM.

Chapter 12: DevOps

In this chapter, we discuss DevOps as a set of principles and practices intended to help development and operations collaborate and communicate more effectively. DevOps is truly taking the industry by storm and, in some circles, reaching almost mythical proportions. I hear folks suggesting that DevOps can help solve almost any issue, which given the versatility of its cross-functional approach, is a view that has some merit, but some groups are losing sight of what this methodology is all about and how it can really help us implement the ALM.

Chapter 13: Retrospectives in the ALM

This chapter discusses the practical application of retrospectives to support application lifecycle management. The first section of this chapter will examine the main function of retrospectives, namely, to evaluate what went well and what needs to be improved. But that’s just the beginning. Getting accurate information from all stakeholders in a retrospective can be very challenging. If you are successful, the retrospective can help drive the entire ALM process. Retrospectives require leadership, and this chapter will provide guidance on how to succeed if you are responsible for implementing this function. We will discuss how to employ retrospectives to support ITIL incidents and problem management, along with other industry standards and frameworks. Crisis and risk management are also key considerations along with IT governance and compliance. Retrospectives take on a different tone when used as vendor management tool. We will complete this chapter by considering how much process is necessary,

how to deal with politics (or, more accurately, *relationships*), and the use of effective metrics to drive the process improvement journey.

Part IV: Scaling the Process

Chapter 14: Agile in a Non-Agile World

In this chapter we discuss that being agile in a non-agile world can be very difficult, and at times even seem impossible to accomplish. We have often found ourselves in organizations that insisted on a waterfall approach. What is most difficult is trying to predict things that are just not possible to ascertain up-front. Many are unaware that waterfall was originally envisioned as an iterative process because today it seems that some organizations expect their employees to be able to predict the future to a degree that is simply not reasonable. The real problem is that these are the same organizations that expect you to make the project actually conform to the plan once it has been developed and approved. Any deviations may be perceived as a lack of planning and proper management. Being agile in a non-agile world can be very challenging and is fraught with its own set of risks and pitfalls.

Chapter 15: IT Governance

In this chapter we discuss how IT governance provides transparency to senior management so that they can make the best decisions based upon the most accurate and up-to-date information. The ALM provides unique capabilities for ensuring that managers have the essential information necessary for evaluating their options. From the CEO to the board of directors, information must often be compartmentalized due to the practical constraints of just how much information can be consumed at any point in time. Achieving this balance empowers your leadership to make informed decisions that help steer your organization to success.

Chapter 16: Audit and Regulatory Compliance

This chapter explains that audit and regulatory compliance require that you establish IT controls to guide the way in which the team works. Your auditors may be internal employees or external consultants engaged by your firm. The internal audit team usually focuses on internal policy, whereas external auditors are often engaged to ensure compliance with federal regulatory guidelines. Although many technology professionals look at audit and regulatory compliance as just something that you have to do, others view it as an obligatory yet unfortunate waste of time and effort. Our focus is on establishing effective IT controls that help avoid both defects and risk. This chapter will help you

understand how to use audit and regulatory compliance to ensure that you prevent the sorts of major systems glitches and outages that we read about all too often.

Chapter 17: Agile ALM in the Cloud

This chapter explains how cloud-based computing promises, and often delivers, capabilities such as scalable, virtualized enterprise solutions; elastic infrastructures; robust services; and mature platforms. Cloud-based architecture presents the potential of limitless scalability, but it also presents many challenges and risks. The scope of cloud-based computing ranges from development tools to elastic infrastructures that make it possible for developers to use full-size test environments that are both inexpensive and easy to construct and tear down, as required. The first step to harnessing its potential is to understand how application lifecycle management functions within the cloud.

Chapter 18: Agile ALM on the Mainframe

This chapter explains how to apply the agile ALM in a mainframe environment. Application lifecycle management on the mainframe typically enjoys a specific workflow. Despite a culture that lends itself well to step-by-step defined procedures, ALM on the mainframe often falls short of its potential. Sure, we can specify steps of a process, and everyone accepts that process toll-gates are necessary on the mainframe. But that does not mean that our mainframe processes help to improve productivity and quality. It is essential that ALM on the mainframe be agile and help the team reach their goals and the business achieve success.

Chapter 19: Integration across the Enterprise

This chapter explains that understanding the ALM across the entire organization requires an understanding of the organization at a very broad level. It also requires that you understand how each structure within the company interfaces with the others. In DevOps, we call this *systems thinking* when we are examining an application from its inception to implementation, operation, and even its deprecation. DevOps principles and practices are essential in integrating the ALM across the organization.

Chapter 20: QA and Testing in the ALM

In this chapter we discuss how quality assurance (QA) and testing are essential to any software or systems lifecycle. Most technology professionals view the QA and testing process as simply executing test cases to verify and validate that requirements have been met and that the system functions as expected. But there

is a lot more to QA and testing, and this chapter will help you understand how to establish effective processes that help ensure your system functions as needed. DevOps helps us build, package, and deploy software much more quickly. Too often, the QA and testing process cannot keep up with the accelerated deployment pipeline. DevOps cannot succeed without excellent QA and testing.

Chapter 21: Personality and Agile ALM

In this chapter we examine key aspects of human personality in the context of the agile ALM. Top technology professionals often have remarkable analytical and technical skills. However, even the most skilled professionals often have great difficulty dealing with some of the interesting behaviors and personalities of their colleagues. Implementing an agile ALM requires that you are able to work with all of the stakeholders and navigate the frequently thorny people issues inherent in dealing with diverse groups of very intelligent, albeit somewhat idiosyncratic, and often equally opinionated, people

Chapter 22: The Future of ALM

In this chapter we discuss what lies ahead for the agile ALM.

Register your copy of *Agile Application Lifecycle Management* 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 (9780321774101) and click Submit. Once the process is complete, you will find any available bonus content under “Registered Products.”

Acknowledgments

Many people assisted me during the writing and publishing of this book, beginning with a family who tolerates my obsession with writing and broadcasting about configuration management and application lifecycle management best practices. I also need to acknowledge the amazing folks at Addison-Wesley, especially Chris Guzikowski who graciously supported my insatiable requests for books and online materials to read during my writing and gently nudged me to get this work completed. I also have to thank the many thousands of colleagues who collaborate with me on the ground implementing these best practices and all those who shared their thoughts and ideas online. I especially want to thank the thousands of people who have written to me commenting on the books and articles that I have written over what has been more than a decade. Writing, for me, is a team sport, and I am grateful for everyone who connected with me via the various social networks and especially everyone who dropped me a note commenting on what I had written. I have learned from each one of these exchanges, and I am grateful for your collaboration and collegiality.

This page intentionally left blank

About the Authors

Bob Aiello has more than twenty-five years of prior experience as a technical manager at leading financial services firms, with company-wide responsibility for CM and DevOps. He often provides hands-on technical support for enterprise source code management tools, SOX/Cobit compliance, build engineering, continuous integration, and automated application deployment. He serves on the IEEE Software and Systems Engineering Standards Committee (S2ESC) management board and served as the technical editor for CM Crossroads for more than 15 years. Bob is also editor of the *Agile ALM DevOps journal* and coauthor of *Configuration Management Best Practices* (Addison-Wesley, 2011).

Leslie Sachs is a New York State certified school psychologist and COO of CM Best Practices Consulting. She has more than twenty years of experience in psychology, intervening in diverse clinical and business settings to improve individual and group functioning. Leslie is assistant editor of the *Agile ALM DevOps journal* and coauthor of *Configuration Management Best Practices*.

This page intentionally left blank

PART I

Defining the Process

This page intentionally left blank

Chapter 4

Agile Process Maturity

Agile process maturity is a very important consideration when implementing an agile ALM. But what exactly does process maturity really mean in an agile context? We know that agile is defined by specific values and principles,¹ so obviously the agile ALM must be—well—*agile*. To begin with, we know from the agile manifesto that agile ALM values individuals and interactions over processes and tools.² But this does not mean that we don't need to focus on processes and tools. Similarly, the agile ALM focuses on creating working software over comprehensive documentation and customer collaboration over contract negotiation. Still, documentation is often absolutely necessary, and signed contracts are rarely optional in the business world. It is equally true that successful professionals do not hide behind a contract and make every effort to delight their customers with excellent value and service.

The agile ALM also emphasizes responding to change over following a plan, although many of the places where we work will not fund any effort without a comprehensive plan. Those who provide the funds for a development project want to know exactly what they are getting into and when they will see results.

In this chapter, we will examine the factors that affect agile process maturity from a number of different perspectives. Many technology professionals find that they must implement agile processes in a large organizational context, including managing teams that are composed of many scrums, totaling scores or even hundreds of developers working from a variety of locations. Scalability is certainly an essential aspect of agile process maturity. Mature agile processes must be repeatable for each project in the organization and have sufficient support for project planning. We also need to understand how process maturity affects non-agile development methodologies, including waterfall and other

1. <http://agilemanifesto.org/principles.html>

2. <http://agilemanifesto.org>

process models. There are other important considerations as well and any discussion of ALM should start with a clear understanding of the goals involved.

4.1 Goals of Agile Process Maturity

This chapter focuses on helping you establish an agile development process that is light but effective and, most importantly, repeatable. This is not an easy goal to accomplish. In many ways, agile shifts the focus away from implementing processes that contain comprehensive controls, or as agile enthusiasts describe as being high in *ceremony*. Ceremony, in this context, really means bureaucracy or, more specifically, laden with excess controls and “red tape.” The goal of this chapter is to help you implement agile processes that are Lean,³ repeatable, clearly defined, measureable, and adhere to the principles defined in the agile manifesto.⁴ We will also discuss how to coexist (or perhaps survive) in non-agile environments. The first step is to understand process maturity in an agile development environment.

4.2 Why Is Agile Process Improvement Important?

Any software or systems development process must continually evolve to meet the ever-changing challenges and requirements of the real world. Agile is no different in this respect. Agile practitioners also know that agile is not perfect and many agile projects have failed for a variety of reasons. Agile processes need to evolve and improve using the very same values and principles that are expected in any agile development effort.

Getting Started with Agile Process Maturity

- Assess your existing practices.
- What works well?
- What needs to be improved?
- Process improvement must be inclusive.
- Prioritize based upon risk.
- Process improvement is a marathon—not a sprint.
- Process improvement must be pragmatic, agile, and Lean.

3. www.poppendieck.com

4. <http://agilemanifesto.org/principles.html>

In some ways agile process maturity could be understood almost in terms of a purity measure. Agile processes that adhere closely to agile principles would, in these terms, be considered a more *agile* process and, obviously, processes that just embrace some agile processes would be more of a hybrid waterfall-agile process.

In order for this measure to be valid, we need to operationalize these principles by considering the extent to which processes embrace agile principles and practices. So how agile are you?

Many organizations want to embrace agile practices and may even recognize the value of agility. They also may find themselves unable to immediately shed their existing processes, especially in terms of corporate governance. This does not mean that they don't want to start the journey, and they may actually reach a very high level of agile process maturity eventually. So how do you start to adopt agile practices and begin the journey?

4.3 Where Do I Start?

The toughest part of implementing mature agile processes is figuring out where to start. I usually start by assessing existing practices and fully understand what works well and what needs to be improved. It is common for me to find that some practices work just fine in one organization that I would have expected were the source of problems. I find that sometimes less-than-perfect processes and procedures may not really be the pain point that one would expect—usually because of the organizational culture. Obviously, trying to fix something that isn't broken will not be very successful, and you will likely find that you do not have much credibility with the key stakeholders if they just don't feel that you are focused on solving the most important problems. In these situations, I communicate my concerns and then focus on what they want me to work on, although I know that they will come back to me and ask for help when things go wrong.

Cludgy Version Control

I recall working with a small software development team supporting an equities trading system. The developers used ClearCase and wanted my help with implementing some branching methods. While I was working with them, I discovered that they had integrated ClearCase with bugzilla in a very unreliable way. They had written the scripts (e.g., ClearCase

triggers) themselves and were very proud of their achievement. I looked at the scripts and realized that these would not work if they had more than one or two developers on the project. I communicated my concerns to the development manager, who assured me that “his” scripts worked just fine. There was no point in trying to fix something that my colleague did not view as broken. The manager approached me a year later, right after he added two more developers to his team and he ran into the problems that I had explained could occur. This time he was more than willing to work with me and accept my help.

Getting started with agile process maturity is certainly an essential first step. Being successful with agile ALM requires that you understand what agile process maturity is all about.

4.4 Understanding Agile Process Maturity

Agile process maturity can be understood in many different ways. The most obvious measure of agile process maturity could be in terms of the degree to which the practices adhere to the agile manifesto and the agile principles.⁵ I usually refer to this as a *purity* measure to indicate the degree to which the process follows authentic agile principles. As a consultant, I am usually called in to help with situations that are less than perfect. This pragmatic reality does change the fact that we want to approach implementing the agile ALM in a manner that adheres to and benefits from agile values and principles.

Agile Process Maturity

Agile process maturity may be understood in terms of

- Adherence to agile principles
- Repeatable process
- Scalability (scrum of scrums)
- Comprehensive (the items on the right)

5. Ibid.

- Transparency and traceability
- IT governance
- Coexistence with non-agile
- Harmonization with standards and frameworks
- Planning
- Continuous process improvement

These need to occur without compromising individuals and interactions, working software, customer collaboration, and responding to change.

In order for this measure to be valid, we need to operationalize these principles. So let's consider what following agile values and principles really means in practice and how we can strive for the most effective agile practices possible.

4.4.1 Adherence to the Principles

Mature agile requires that we adhere to the agile principles that we reviewed in Section 3.7. In this book we seek to educate you on software methodology in a way that empowers you to apply these principles and create a software lifecycle that is best for your project and your organization. One of the ironies that we often see is that some agile practitioners are the least “agile” people in the world, insisting on there being only one right way to become *truly* agile. I disagree, and we hope to share the best practices in creating an agile ALM that you can tailor to your own special requirements.

Dysfunctional Agile

In our consulting practice, we often see groups adopting agile practices and actually getting lost along the way. Becoming agile does not happen overnight and, in practice, maybe it shouldn't. Many groups have legacy processes in place that cannot be abandoned without affecting projects already underway. We view organically transitioning to agile as being more practical. In order to be successful, your team needs to understand agile principles and how to create a mature agile application lifecycle. Above all, right-sizing your processes is your most critical success factor.

4.4.2 Repeatable Process

Agile processes, just like any other process, must be repeatable. It does not help to have an agile ALM unless it can be used repeatedly to achieve the desired results. We have seen many agile teams struggle with repeatability because they depended upon the guidance of individual players rather than understanding that agile is still a process that should yield the same results, regardless of who is performing the task—assuming the proper level of skills and training.

Agile and Law Enforcement

Bob has long had a passion for serving as a volunteer in both law enforcement and emergency medical services. From responding to fires, to medical emergencies, and especially to crimes in progress, police and emergency personnel must provide a predictable consistent response while still maintaining the flexibility to deal with the situation at hand. When you call because a bad guy is breaking into your car in front of your house, you expect the same results regardless of which police officer happens to respond. You also realize that the situation can be dynamic, and police must be the very model of agility. Law enforcement, emergency medical, and fire response must provide repeatable processes while maintaining the flexibility to respond to the situation at hand.

Agile process maturity should be understood in terms of repeatability. Another important issue is scalability.

4.4.3 Scalability (Scrum of Scrums)

Organizations often pilot agile methodologies in one particular group with spectacular results. The truth is that the participants in the agile pilot are often hand-picked and among the best resources in the organization. But agile processes must be scalable so that other teams within the organization can also be successful. We discuss the critical success factors for scalability throughout this book and then tie them together in Chapter 19, “Integration across the Enterprise.” If you want a scalable process, then you need to start by ensuring that your approach is comprehensive.

4.4.4 Comprehensive (Items on the Right)

Agile processes must be comprehensive so that everyone understands what needs to be accomplished, including interdependencies and deadlines. The agile manifesto aptly notes the following:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.⁶

Mature agile processes value the items on the right so that we can ensure our processes are comprehensive, including processes and tools, comprehensive documentation, contract negotiation, and following a plan.

Comprehensive processes are essential, as are transparency and traceability.

4.4.5 Transparency and Traceability

Mature agile processes are transparent and traceable. Transparency is fundamental because you want everyone to understand what is being done and how their work impacts (and is impacted by) the work of others. You also want to be able to verify that steps have been completed. Processes that are transparent are easier to understand and follow, ensuring that everyone understands the rules of the road. Being able to go back and verify that each step was completed successfully is also essential, particularly when regulatory compliance is required. In addition, you want to be able to provide transparency to senior management through effective IT governance.

4.4.6 IT Governance

IT governance provides visibility into the organizational processes and existing operations so that senior management can make accurate decisions based upon the information that is available. I always explain to my colleagues that IT governance is essential because this function enables senior management to make the right decisions based upon accurate and up-to-date information. You can even look at IT governance as managing the right information “up” to those who are in the position of making decisions. In some large organizations, agile projects may be in progress at the same time as non-agile projects.

6. <http://agilemanifesto.org/>

Mature agile processes must be able to successfully coexist in these real-world hybrid environments.

4.4.7 Coexistence with Non-agile Projects

The elephant in the room for agile is the number of non-agile projects that exist within an organization that is working to implement agile. We have seen many organizations where existing non-agile projects were already underway, or perhaps existing team members were just not comfortable with taking an agile approach. Mature agile application lifecycle management often requires coexistence with non-agile projects. Coexistence is a sign of maturity, as is aligning with industry standards and frameworks.

4.4.8 Harmonization with Standards and Frameworks

Many organizations must follow the guidance provided in industry standards, including ISO 9000 or frameworks such as ISACA COBIT or the ITIL v3 framework. Mature agile processes can easily align and harmonize with the guidelines provided by these well-respected industry standards and frameworks. This includes meeting the requirements of Section 404 of the Sarbanes-Oxley Act of 2002 or safety standards such as those commonly required by the automotive, medical engineering, or defense industries. Mature agile helps improve quality and can align well with IT controls that are reasonable and appropriate.

The agile manifesto notes that it is more important to be able to respond to change than to simply follow a plan. However, mature agile processes must still be able to create adequate plans that will help guide the development effort.

4.4.9 Following a Plan

Planning is essential for the success of any significant endeavor. Too many agile enthusiasts erroneously think that they don't need to create comprehensive plans to guide the software and systems development effort. The dark side of planning is that sometimes those creating the plan refuse to admit what they do not know. Mature agile processes plan as much as possible and communicate those plans effectively. Unknowns should be identified as risks, which are then mitigated as part of the risk management process. Many years ago W. Edwards Deming noted the importance of "driving out fear." Agility admits when it does not have enough information to specify the details of a plan. Decisions are made at the "last responsible moment." Mature agile processes

embrace comprehensive plans but also do not attempt to plan out details that cannot yet be specified.

4.4.10 Continuous Process Improvement

Process improvement is a journey that must be continuously harnessed throughout the software and systems lifecycle. The mature agile process embraces continuous process improvement at both a deep technical level and at a pragmatic business level. Improving your technical processes is mostly focused on avoiding human error while maintaining a high level of productivity and quality. Improving your business processes can be a bit more complicated.

Do you make satisfying the customer through early and continuous delivery of valuable software your highest priority? Does your agile ALM process harness change for the customer's competitive advantage and welcome changing requirements, even late in development? Your delivery cycle should favor shorter iterations, with delivering working software frequently, from every couple of weeks to every couple of months. Developers and businesspeople should be working together daily throughout the project. Projects are built around motivated individuals, and they are provided the environment and support they need and are trusted to get the job done. Information is best conveyed face to face, and working software is the primary measure of progress.

The agile ALM should help all the stakeholders maintain a constant pace indefinitely in what is known as sustainable development. There is also continuous attention to technical excellence and good design, including a focus on simplicity—the art of maximizing the amount of work not done. Self-organizing teams produce the best architectures, requirements, and designs. At regular intervals, the team reflects on how to become more effective and then tunes and adjusts its behavior accordingly. These principles have formed the basis of agile development for many years now. In order to understand them, you need to consider how to operationalize and implement these principles in practice. Then we will show how they fit into and, of course, facilitate the agile ALM.

4.5 Applying the Principles

Implementing the agile ALM requires that you understand the agile values and principles and, more importantly, how to utilize them in practical terms. Technology projects require a deep understanding of exactly what the system should do and how it should work. These are important details that are typically expressed in terms of requirements. There are many different types of

requirements, from system-level response time to functional usage, including navigation. Many professionals use epics⁷ and stories⁸ to describe requirements in high-level terms. Writing and maintaining a requirements document is often less than fruitful, with most requirements documents out of date even before they have been approved by the user. Agile takes a pragmatic approach to requirements management that focuses on working software instead of writing requirements documents that are often of limited value.

One very effective way to manage requirements is to supplement them with well-written test cases and test scripts. Test cases often contain exactly the same information that you might expect in a requirements document.

Test Cases for Trading Systems

Many years ago I requested that the testers work with me to write test cases for a major trading system in use on the floor of the New York Stock Exchange. The user representative was hesitant at first to focus on testing early in the software development process. I managed to persuade one of the senior representatives to give me one hour. During that session I simply asked him to say what he would test and what he expected the results to be. Within that first hour, this business expert actually picked up the phone and told the head of development that “what he had asked for was not what he needed.” I had caused the business expert to start actively thinking about how he was really going to use the system. The requirements phase had been long and thorough, yet the real breakthrough occurred when we started writing test cases. Well-written tests can be very effective at supplementing, and even completing, requirements descriptions that are often incomplete because all of the usage details may not be initially understood.

4.6 Recognition by the Agile Community

Agile development is part of a large ecosystem with an active and involved community. Mature agile processes are aligned with agile principles and are recognized by the agile community. Much of my work involves taking innovative

7. Epics are a description of a group of features (e.g., stories) that help document requirements in agile development.

8. Stories are descriptions of features from an end-user perspective, which serve to document requirements in agile development.

and even risky approaches when I customize software methodology to meet the unique needs of often complex organizations. Although I always maintain confidentiality, I find it effective to write and publish articles that describe my approach to DevOps and agile process maturity. Sometimes, my views are well accepted by the agile community, and other times the reaction can be quite significant. I actually use my esteemed colleagues in the agile community as a feedback loop to continuously improve my own process methodologies.

Recognition within the agile community is a worthy goal. However, gaining consensus may be much more difficult to achieve.

4.7 Consensus within the Agile Community

The agile community can be both opinionated and very vocal. It can also be difficult to gain consensus. You need to expect that there will always be a diversity of views and opinions expressed in the agile community. Sometimes, views are expressed in rather emphatic terms. In fact, it is the great irony that some agile practitioners are the least agile people I have ever worked with, insisting that agility can be practiced in only one particular way. My view is to enjoy the plurality of opinions, looking for consensus when I can find it and also understand that sometimes experienced practitioners will have differing points of view. This is especially true when confronting some of the more thorny issues in understanding agility. One of these considerations is what agile process maturity is not.

4.8 What Agile Process Maturity Is Not

The agile process is not an excuse to skip documenting and tracking requirements, so agile process maturity is also not an excuse for failing to implement enough “ceremony” to get the job done. Although agile has boasted many fabulous successes, it is also not without its failures. One of the biggest problems with agile today is folks just going along with what they are told without questioning and reflecting upon the effectiveness of the agile process.

Emperor’s Clothes

Hans Christian Andersen tells the age-old story of the Emperor’s New Clothes in which a team of conmen come into a town and convince everyone there that they can create a set of clothes that can only be seen

by those subjects who are truly loyal to the emperor and are invisible to those unfit for their positions, are stupid, or are incompetent. As the story goes, the emperor is sitting on his throne in his underwear while these two men pretend to be tailoring him a fine suit. Predictably, everyone is silent because they are afraid to speak up and have the emperor think that they are not loyal to him. Finally, a young child blurts out that the emperor is not wearing any clothes and the townspeople realize that they are being fooled.

Too often folks involved with the agile transformation are silent even though they may have well-founded misgivings. The young child in this fable innocently speaks up. We should all have the courage to express our own misgivings. Remember Deming teaches us to drive out fear.

Immature agile processes can create many challenges for the development team.

4.9 What Does an Immature Agile Process Look Like?

Immature agile processes can resemble software development in the Wild, Wild West. If your team delivers in an inconsistent way and lacks transparency and predictability, then you are likely dealing with a lack of process maturity. You might even be successful from time to time—but maturity involves a lot more than occasional heroics. There are many other potential problems with agile.

4.10 Problems with Agile

Too often agile has become an excuse to work in a very loose and ineffective way. We have seen agile teams use the agile manifesto as an excuse to not plan out their projects and also to not communicate their plans with others. Sometimes teams also fail to document and track requirements, which can lead to many problems, including a high incidence of defects. We have also seen teams that used agile as an excuse to not document their work. Mature agile processes provide the right balance of planning, requirements management, and documentation to avoid mistakes and get the job done. We recall one major incident in a large bank where a vendor claimed to be employing agile and shipped a release that was not really ready to be seen by the customer.

One CIO's View of Agile

During a configuration management (CM) assessment, which I conducted as a consultant, I had the opportunity to speak with the CIO of a large international bank, who described his recent experience with a software vendor who had represented their development practices as being agile. The vendor did a lot of development offshore with teams of only four or five developers using scrum and sprints that lasted only two or three weeks. Because the team adhered to fixed iterations, they were delivering code that was incomplete or, as the CIO described it, “half-baked.” I spoke with the vendor’s development manager, who essentially admitted that they did adhere strictly to fixed timebox iterations and, as a result, occasionally some features were not completely implemented. The vendor saw no problem with this and viewed their development methodology as quite excellent, completely ignoring the viewpoint of the customer.

Although agile has its challenges, let’s not lose sight of the challenges often seen in waterfall.

4.11 Waterfall Pitfalls

Agile enthusiasts have long described the many pitfalls and problems inherent in following the waterfall software methodology. In Chapter 14, “Agile in a Non-Agile World,” we will discuss these and other challenges as well, but also acknowledge that there are times when waterfall is the only choice. From the perspective of agile process maturity, we need to understand exactly where waterfall is problematic so that we do not make the same mistakes in our agile or hybrid agile processes.

Waterfall, as envisioned by Winston Royce,⁹ was iterative in nature. But waterfall fails when you try to define requirements that are not yet understood. Many organizations go through exhaustive planning exercises that are essentially an effort to plan what is not yet known and understood. When creating a plan, you need to identify the things that are not yet well understood as project risks. Risk itself is not inherently bad. Many organizations, including trading

9. Royce, Winston W. (1970). “Managing the Development of Large Software Systems.” In: Technical Papers of Western Electronic Show and Convention (WesCon) August 25–28, 1970, Los Angeles, USA.

firms, actually thrive on risk. It is also essential to create adequate documentation and to keep it updated as necessary. Many organizations spend so much time trying to track requirements and create exhaustive project plans that they leave no time to actually get to software development and have to rush to make project deadlines. This dysfunctional approach can result in defects and significant technical debt.

Mature agile processes take a pragmatic approach to requirements definition and tracking while also establishing enough of a project plan to communicate dates and deliverables to all stakeholders. There are times when documentation is not negotiable, whether your project is using agile or waterfall.

Essentials

- Planning the unknown
- Failing to manage risk
- Documentation outdated
- No time for coding since we spent our time planning

4.11.1 Mired in Process

We often see organizations that are simply mired in their waterfall processes. These groups typically take a very rigid approach to planning and requirements gathering. Although sometimes waterfall makes sense, it is essential to always be pragmatic and avoid getting mired in your own processes. When this happens, we have seen teams where it actually became part of the culture to pretend to be following the process.

4.11.2 Pretending to Follow the Process

One of the most dysfunctional behaviors we often see is organizations that require complete adherence to waterfall processes, which results in team members being forced to pretend to be following these rigid waterfall processes. In these circumstances we find people who feel pressured into creating and following plans even when they really do not have all of the necessary details, or creating requirements specifications that document features that are not yet well understood. If management forces employees to follow waterfall in a rigid and dysfunctional way, then they really have no choice but to smile and pretend to follow the process. The better way is to create mature agile processes that include both the items on the left of the agile manifesto and the items on the right.

4.12 The Items on the Right

The agile manifesto teaches us to value individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan. But mature agile processes must have robust processes and tools, adequate documentation, and plans. You also don't want to try to engage with customers without well-written contracts and clear agreements. The items on the right side of the agile manifesto are actually very important. It is also important to adjust your ceremony for the environment and culture in which your organization is operating.

4.12.1 Adjusting Ceremony

Agile processes are said to be “light” in terms of ceremony, which means that they are not overly burdensome with rigid verbose rules and required procedures, which are inherent in creating IT controls. Mature agile processes are able to adjust the amount of ceremony required to avoid mistakes and still get the job done. Although right-sizing the amount of process is a must-have, so is coexisting with non-agile processes when necessary.

4.13 Agile Coexisting with Non-Agile

There are many times when agile simply *must* exist with non-agile processes. This is the real world that many agile practitioners find so difficult to accept. We work with many large banks and financial services firms where agile must coexist with non-agile processes. This is often the case when large organizations must have IT governance in place to ensure that senior management can make decisions based upon adequate information.

4.14 IT Governance

IT governance is all about providing information to senior management so that the right decisions can be made. Many agile processes suffer from failing to provide adequate information to senior managers. Mature agile processes provide enough information so that senior management can make the right decisions in support of the development effort. IT governance is closely aligned with providing transparency.

4.14.1 Providing Transparency

Mature agile processes provide the transparency that is essential to help all stakeholders understand the tasks that they have to complete and especially how their work affects the work of other members of the team. Processes, and especially workflows, help the entire team understand what needs to be done on a day-to-day basis. This is exactly where having just enough process can help you get the job done and avoid costly mistakes. Above all, you want to have an ALM that follows the agile principles.

4.15 ALM and the Agile Principles

Mature agile processes should obviously adhere to agile principles. The agile ALM is customer-centric and facilitates the early and continuous delivery of valuable software. We welcome changing requirements, even late in development, and harness change for the customer's competitive advantage. The agile ALM should help deliver working software by frequently facilitating daily collaboration between all stakeholders, including businesspeople and developers. Projects should be built around motivated individuals with the environment and support they need while encouraging face-to-face interactions. Working software is the primary measure of progress.

The agile ALM should promote sustainable development, including a constant pace throughout the duration of the project. There also should be continuous attention to technical excellence and good design enhancing agility, along with valuing simplicity instead of overly complex design and processes. The agile ALM empowers the cross-functional self-organizing team, resulting in the best architectures, requirements, and designs. The mature agile ALM also includes regular opportunities to reflect on how the process can become more effective, tuning and adjusting its processes and behavior. The mature agile process adheres to these agile principles on a constant and reliable basis. This is why you need to start off with processes that are repeatable and predictable.

4.16 Agile as a Repeatable Process

Mature agile processes must be repeatable above all else. Even the best process will be of little value if it cannot be used reliably across all of the projects and groups involved with completing the work. Closely related is the need for scalability.

4.16.1 Scalability

Scalability means that the mature agile process can be used reliably across the enterprise. We often find that this is exactly where organizations struggle the most. We will review some tactics to help ensure that your processes can scale to the enterprise in Chapter 19, “Integration across the Enterprise.” Another key aspect of agile process maturity is ensuring that you deliver on time and within budget.

4.16.2 Delivering on Time and within Budget

We see many agile teams struggling with the reality that no one is going to give them a blank check and tell them to take their time on delivering results. Mature agile processes should provide enough planning and structure to help ensure that the software can be delivered on time and within budget. Unless your senior management team is clairvoyant and just anticipates your team’s every whim, you will need to communicate what you need to get the job done. This should include a clear idea of the timeframe required and the budget that will help ensure success of the project. This is particularly essential when considering the quality of the software that you deliver.

4.16.3 Quality

Mature agile processes must ensure that quality is a top priority. This requires a strong focus on robust automated testing and benefits greatly from thorough test planning. Well-written test cases can help supplement even incomplete requirements documents. Mature agile processes cannot survive without a strong focus on quality and testing. W. Edwards Deming, regarded by many as the father of quality management, was well known for explaining that quality must be built in from the very beginning of the software and systems lifecycle. This is particularly true in mature agile processes.

4.17 Deming and Quality Management

Many of the lessons from Deming are a main focus of the agile ALM, and we will point them out throughout this book. Testing is essential, but there are many other ways to build quality into the agile ALM.

4.17.1 Testing versus Building Quality In

Application testing is a must-have. But quality has to be built in from the very beginning. Code reviews and inspections are among the tools that help ensure

quality is built into the product from the very beginning. Ensuring that requirements are well defined is essential for ensuring high-quality systems. The agile ALM provides a comprehensive framework of best practices to help build quality into the product from the very beginning. It is also the best way to help improve productivity.

4.17.2 Productivity

Technology professionals often find themselves mired in the quagmire of trying to get work done efficiently. The mature agile ALM helps avoid mistakes and rework that is so often the reality of today's software and systems development effort. One of the most effective practices to improve productivity is rapid iterative development.

4.18 Agile Maturity in the Enterprise

Implementing processes across any large organization can be very challenging, and agile process maturity should be measured across the enterprise. While we are not advocating comparing groups to each other, which could actually be counterproductive, we do want to have common criteria to help each team plan their own process improvement efforts. It is best to understand processes within the group context itself. We have seen teams that had technical flaws in their processes, tools, or procedures and in one group these issues presented a significant challenge, but for another it was almost irrelevant. For example, we have seen teams lack strong version control procedures but somehow manage to avoid problems that we would have expected through sheer force of will or even manual controls. Obviously these situations are optimal, but still each team may have a very different view of their priorities and pain points. We implement agile processes consistently across the enterprise, while still understanding that each team may have a somewhat different culture, environment, and priorities. We can manage this balance by establishing the goals and objectives while understanding that there could be some difference in processes, tools, and procedures.

4.18.1 Consistency across the Enterprise

Process maturity models can be helpful in establishing common criteria to help ensure consistency across the enterprise. We also use industry standards and frameworks as a source of consistent best practices to implement across the

enterprise. For example, we might ask a team to explain how they implement automated build, package, and deployment, including their procedures to verify and validate that the correct code has been deployed. Teams are often quite up-front about what they are doing well and what could be improved. Helping each team to focus on its own perceived priorities is essential for successful process improvement. But there is also room for ensuring that industry best practices are implemented consistently across the firm. This work requires excellent communication and even some good marketing of the new approach.

4.18.2 Marketing the New Approach

We never assume that teams will just automatically agree to implement the best practices that we advocate. Sometimes, it is best to help a team create its own plan. We balance this approach with enterprise process improvement efforts to essentially market industry best practices using the latest processes and tools. Throughout this effort it is essential to continuously focus on process improvement.

4.19 Continuous Process Improvement

The most effective way to implement mature agile processes is to take an agile and iterative approach to implementing the agile ALM itself. This means that you need to be continuously working toward excellence. Learning from mistakes is par for the course, and effective processes should also be self-correcting.

4.19.1 Self-Correcting

Process improvement is not without its challenges. The important thing is to ensure that your processes correct themselves and evolve. Being able to improve your processes is much easier if you are able to measure them and demonstrate progress over time.

4.20 Measuring the ALM

We tend to be wary of overengineering the measurement process, as some teams tend to try to game the measurement. With any measurement approach, it is important to consider validation up-front. This is especially true with regard to metrics.

4.20.1 Project Management Office (PMO) Metrics

Metrics, including those used in project management, can be very important. More importantly, selecting valid and verifiable metrics is key to ensuring a successful measurement approach leading to quantifiable process improvement. Our experience has been that less is more in this case, and the best approach is to select a few metrics that are valid and verifiable. Establishing an in-house metrics program is very important. It is also important to ensure that your vendors do the same.

4.21 Vendor Management

Vendors often have strong sales and marketing functions that sometimes include information on their processes, which can include metrics. It is important for you to review and understand your vendors' criteria. We have had many times when we were asked to review vendor programs and give our recommendations on ensuring that the vendor approach was aligned with our client's requirements. It has been our experience that many vendors welcome this input and where there are gaps, they should be understood as well. Although agile process maturity is typically focused on software, we often review processes around hardware development as well.

4.22 Hardware Development

Hardware development is often dependent upon a waterfall approach because half an incomplete circuit chip is often not very helpful. Our effort is to align the agile ALM with the engineering lifecycle required to design and implement hardware. This is often required when we consult with firms that create firmware.

4.22.1 Firmware

Firmware is software that must be created and embedded in the hardware that consists of the complete hardware-software component. We view agile process maturity as part of this alignment and have seen teams succeed quite well even when using a hybrid waterfall approach for the hardware and an agile approach for the firmware.

4.23 Conclusion

There are many factors to consider when creating a mature agile process. We have introduced and reviewed many of the issues involved with creating mature agile processes. The agile ALM needs to be aligned with the technology, environment, and culture of the team and the organization within which it will operate. Rarely do we see teams get this right the first time, and the most successful groups take an agile iterative approach to creating their agile ALM.

This page intentionally left blank

Index

Note: Locators followed by an italicized *n* indicate a footnote.

A

A priori change control, 19

A priori changes, 177

Aggressive team members, 331–333

Agile ALM

agile manifesto, 56–57

agile principles, 54–56, 76

customer collaboration, 58–59

designing circuit chips, real-world example, 58

DevOps for customers, 59

documentation, 60

fixed timebox sprints, 57–58

getting started, 50–51

goals of, 49–50

hybrid of agile and non-agile methods.
See Hybrid agile.

importance of, 50

organizational culture, 51–52

paradigm shift, 51–52

RAD (rapid application development),
52–54

rapid iterative development, 52–53

requirements, 59–60

standard terminology, 51

user test cases, 59–60

Agile manifesto

items on the left, 56

items on the right, 56, 75

principles of, 56–57. *See also specific principles.*

responding to change over valuing a plan, 118

standards and frameworks, 68

working software over comprehensive documentation, 110–111

Agile principles

agile process maturity, 69–70, 76

under ALM, 76

in IT operations, real-world example,
210

list of, 54–56

service desks, 210

Agile process maturity

adjusting ceremony, 75

agile principles, 76

applying the principles, 69–70

coexisting with non-agile projects, 75

consensus within the agile community,
71

consistency across the enterprise,
78–79

continuous process improvement, 79

delivering on time within budget, 77

in the enterprise, 78–79

epics and stories, 70

firmware development, 80

getting started, 62–64

goals of, 62

hardware development, 80

importance of, 62

IT governance, 75–76

in law enforcement, real-world exam-
ple, 66

marketing the new approach, 79

measuring the ALM, 79–80

one CIO's view, real-world example,
73

PMO (project management office)
metrics, 80

problems with, 71–73

quality, 77

recognition by the agile community,
70–71

recognizing an immature process, 72

- Agile process maturity, *continued*
 - repeatable processes, 76–77
 - requirements, 69–70
 - scalability, 77
 - self correction, 79
 - test cases for trading systems, real-world example, 70
 - transparency, 76
 - vendor management, 80
 - version control, real-world example, 63
- Agile process maturity, overview
 - coexistence with non-agile projects, 68
 - comprehensive processes, 66–67
 - continuous process improvement, 69
 - dysfunctional agile, 65
 - IT governance, 67–68
 - planning, 68–69
 - principles of, 64–65
 - purity measure, 64
 - scalability, 66
 - standards and frameworks, 68
 - traceability, 67
 - transparency, 67
- Agile processes
 - agile development *versus* iterative development, 16–17
 - disasters, transitioning from hybrid agile to agile, 255
 - hybrid of agile and non-agile methods. *See* Hybrid agile.
- Agile service catalog, real-world example, 186
- ALM (application lifecycle management)
 - agile methodology. *See* Agile ALM.
 - build engineering, 98–99
 - change management. *See* Change management.
 - future of, 347–349
 - getting started, 7–8
 - goals of, 4–5
 - importance of, 5–6
 - mainframe. *See* Mainframe ALM.
 - purpose of, 6
 - versus* software delivery lifecycle, 7
 - versus* systems delivery lifecycle, 7
- ALM (application lifecycle management), overview
 - addressing the business silo, 13
 - audit and regulatory compliance, 24
 - automation, 21
 - build engineering, best practices, 18
 - business focus, 11–15
 - change management, best practices, 19
 - change management, goal of, 22
 - CI (continuous integration), 21–22
 - cloud-based computing, 24–25
 - core configuration management, 17–21
 - definition of ALM, 8–10
 - deployment, best practices, 20–21
 - deployment, continuous. *See* CD (continuous deployment).
 - versus* development lifecycle, 9
 - DevOps, 23
 - environment management, best practices, 19
 - financial systems infrastructure, real-world example, 14
 - integration across the enterprise, 25
 - IT governance, 23
 - IT operations, 22–23
 - mature processes *versus* fluid, 16–17
 - QA (quality assurance), 25
 - rapid iterative development, 17
 - release management, best practices, 19–20
 - retrospectives, 23
 - risk, from a business focus, 13–15
 - role of personality, 26
 - scope of, 9
 - SDLC (software development life cycle), 10–11
 - source code management, best practices, 17–18
 - testing, 25
- Ambler, Scott, 347
- American Foundation for the Blind (AFB), 218
- Anderson, Hans Christian, 71–72
- Application design, automating, 111
- Application management, IT operations, 208

- Applications, testing, 39
- Archetypes, 319
- Assessing success, with retrospectives, 235–236
- Attended automation
 - agile ALM, 104
 - continuous deployment, 145
 - DevOps, real-world example, 226
- A-type personality, 331–333
- Audit and regulatory compliance
 - assessing existing practices, 283–284
 - audit and accountability, 277
 - essential requirements, 283
 - external audits, 277
 - getting started, 274–275
 - goals of, 273–274
 - identifying standards and frameworks, 275
 - importance of, 274
 - improving quality and productivity, 283
 - internal audits, 275–276
 - IT controls, 275–276
 - IT governance, 271
 - overview, 24
 - retrospectives, 244
- Audit and regulatory compliance, federal
 - guidelines
 - banking oversight, 282
 - Cobit as framework for IT controls, 280
 - COSO (Committee of Sponsoring Organizations), 279
 - essential components of internal control, 279
 - FINRA (Financial Industry Regulatory Authority, Inc.), 280
 - GAO (Government Accountability Office), 281–282
 - guidelines on internal controls, 282
 - HIPAA (Health Insurance Portability and Accountability Act) (1966), 280–281
 - ISACA Cobit, 281. *See also* Sarbanes-Oxley Act of 2002.
 - management assessment of internal controls, 278–279
 - OCC (Office of the Comptroller of the Currency), 282
 - oversight of securities firms, 280
 - Sarbanes-Oxley Act of 2002, 278–280. *See also* ISACA Cobit.
 - self-administered risk assessment surveys, 280
 - for senior management responsibility, 278–280
- Audit and regulatory compliance, real-world examples
 - audit and accountability, 277
 - internal audits, 276
- Automation
 - application design, 111
 - attended automation, 104
 - change management, real-world example, 173
 - code quality instrumentation, 111–112
 - deployment, 225–227
 - DevOps, 115–116, 230
 - environment management, 114–115
 - epics and stories, creating, 111
 - getting started, 104, 107
 - goals of, 103
 - gold copies, 114–115
 - help desks, 116–117
 - for implementation, 119
 - importance of, 103–104
 - incident management, 117
 - IT operations, 195–196
 - IT workflow, real-world example, 196
 - keyman risk, 108
 - lifecycle management, 109
 - operations, 116
 - overview, 21
 - PMO (project management office), 118
 - problem escalation, 117–118
 - process modeling, 108
 - project management, 118
 - requirements management, 110–111
 - seamless integration, 109–110
 - service desk, 117
 - systems design, 111
 - TDD (test-driven development), 113–114
 - test case management, 112–113

- Automation, *continued*
 - testing the lifecycle, 112
 - tool agnosticism, 106
 - tools for. *See* Tools, for automation.
 - use cases, defining, 119
 - workflow, 108
 - workflow, continuous deployment, 148–150
- Automation, build engineering
 - the application build, 94–95
 - automation tools, 94
 - build robots, 99
 - code scans, 100
 - detection of unauthorized changes, 96–97
 - unit tests, 100
- Autonomy, 339
- B**
- Banking oversight, federal guidelines, 282
- Banking systems
 - change management, real-world example, 165–166
 - continuous deployment, real-world example, 156
- Baseball players and mistakes, real-world example, 236
- Baselining, 96–97
- Bimonthly deployments, real-world example, 146–147
- Blaming, 132–133
- The blind, real-world example of delivering retrospectives, 238–239
- Boehm, Barry, 53
- Books and publications
 - Configuration Management Best Practices: Practical Methods that Work in the Real World*, 18, 348
 - The Software Project Manager's Bridge to Agility*, 6
- Broderick, Stacia, 6
- B-type personality, 331–333
- Build engineering
 - in the ALM, 98–99
 - automation. *See* Automation, build engineering.
 - baselining, 96–97
 - best practices, 18
 - build robots, creating, 99–100
 - build tools, 101
 - building quality in, 100
 - code scans, 100
 - compile dependencies, 98
 - components of the build, 93–94
 - cryptographic hashes, 96
 - definition, 91
 - detecting unauthorized changes, 96–97
 - failing fast, 94–95
 - failure, real-world example, 94
 - getting started, 92–93
 - goals of, 91–92
 - hackers, 95
 - IDEs (integrated development environments), 93
 - importance of, 92
 - independent builds, 99–100
 - instrumenting the code, 101
 - physical configuration audit, 98
 - secure trusted base, creating, 95–96
 - stopping the line, 94–95
 - unit tests, 100
 - version IDs, 97–98
- Build farms. *See* CI (continuous integration), build farms.
- Build management, cloud-based ALM, 289
- Build robots, creating, 99–100
- Build servers. *See* CI (continuous integration), build farms.
- Build tools, 101
- Building quality in, build engineering, 100
- Business continuity, 230
- Business focus, overview, 11–15
- Business management, IT operations, 205–206
- Business silos. *See* Silo mentality.
- C**
- CAB (change advisory board), 176, 202. *See also* CCB (change control board).
- Canary deployment, cloud-based ALM real-world example, 290
- CASE (computer-aided software engineering), 53

- CBOE (Chicago Board Options Exchange) shut down, real-world example, 329
- CCB (change control board), 176, 291.
See also CAB (change advisory board).
- CD (continuous deployment)
 - addressing the culture, 141
 - attended automation, 145
 - banking system, real-world example, 156
 - bimonthly deployments, real-world example, 146–147
 - breaking into smaller pieces, 145–146
 - CI (continuous integration), 138
 - container-based deployment, 144
 - versus* continuous delivery, 22, 139–140
 - copying files, 142
 - data processing director, real-world example, 149
 - definition, 22
 - deployment pipeline, 141–142
 - dress rehearsal, 154–155
 - eliminating problems, real-world example, 41
 - emergency medical tech, real-world example, 142
 - environments that mirror production, 152–153
 - ergonomics, 150
 - failure, 155
 - getting started, 141
 - goals of, 139–140
 - Hibernate, real-world example, 153–154
 - identifying dependencies, 152
 - importance of, 140
 - Kanban, 148–150
 - Maven, real-world example, 153–154
 - monitoring, real-world example, 152
 - moving targets, real-world example, 143–144
 - nuclear power plant, real-world example, 150
 - overview, 22
 - plan B, 155–156
 - police force, real-world example, 149
 - practicing, 146–147
 - rapid incremental deployment, 143–144
 - repeatability, 147–148
 - risk assessment, 153–154
 - risk management, 153–154
 - risk management container-based deployment, 144–145
 - sarin gas, real-world example, 154–155
 - smoke testing, 156–157
 - in the software development process, 41
 - traceability, 147–148
 - training, 147
 - trusted base, 151–152
 - validation, 150–151
 - verification, 150–151
 - walkthroughs, 154–155
 - WIP (work in progress), 149
 - workflow automation, 148–150
- Center for Internet Security (CIS), 209
- Centralized service desks, 210
- Ceremony
 - adjusting, 75
 - in agile process maturity, 62
 - definition, 12
 - retrospectives, 245
- Change advisory board (CAB), 176, 202. *See also* CCB (change control board).
- Change control
 - bypassing on mainframe ALM, 301
 - in the software development process, 47
- Change control board (CCB), 176, 291.
See also CAB (change advisory board).
- Change evaluation, standards and frameworks, 204
- Change management
 - in ALM, 166
 - best practices, 19
 - CAB (change advisory board), 176
 - CCB (change control board), 176
 - change ecosystem, 167
 - cloud-based ALM, 290–292
 - collecting feedback, 171–172
 - command center, 169–170
 - communication, 165–166

- Change management, *continued*
 - compliance, 164
 - continuous process improvement, 183–184
 - cross-enterprise coordination, 180–181
 - cross-platform coordination, 180
 - escalating problems, 172–173
 - event monitoring, 168–169
 - feedback loops, 171
 - fiefdoms, 181
 - getting started, 163–164, 177
 - goal of, 22, 161
 - importance of, 162
 - incident response, 170–172
 - IT operations, 205–206
 - last responsible moment, 118
 - normal changes, 175
 - organizational structure, 176
 - overview, 22
 - pre-approved changes, 174, 175
 - a priori change control, 19
 - problem management, 172–173
 - problems *versus* incidents, 172–173
 - publishing changes back to the system. *See* Rebasing.
 - QA (quality assurance), 167–168
 - real-world example, 205–206
 - risk assessment, 164–165
 - risk management, 164–165
 - SaaS change control, 182–183
 - SEPG (software engineering process group), 166
 - in the software development process, 33–34
 - specialized change control, 182
 - standard changes, 175
 - standards and frameworks, 202, 205
 - testing, 167–168
 - traceability, 164
 - troubleshooting, 169
 - vendor change control, 182
- Change management, change control topology
 - configuration control, 178–179
 - E-change control, 179–180
 - emergency change control, 179
 - gatekeeping, 177
 - normal changes, 180
 - overview, 176–177
 - preapproved changes, 180
 - a priori changes, 177
 - process change control, 179
 - RFC (requests for change), 177–178
 - SEPG (software engineering process group), 179
 - standard changes, 180
- Change management, process description
 - change request boards, 174. *See also*
 - CAB (change advisory board);
 - CCB (change control board).
 - entry/exit criteria, 174–175
 - overview, 173–174
 - post-implementation reviews, 175. *See also* Retrospectives.
 - pre-approved changes, 174
- Change management, real-world examples
 - automation system, 173
 - banking systems, 165–166
 - collecting feedback, 171–172
 - global incident response, 170
 - in a government agency, 181
 - mainframe outage, 171
 - negative attitudes towards, 163
 - problems, learning from, 173
 - QA (quality assurance), 168
 - service providers, 183
 - storage, 162
 - technical debt, 165
 - troubleshooting, 169
 - upgrading a GPS, 183
- Change planning, software development process requirements, 36
- Change request boards, change management, 174
- Chaos monkeys, real-world example, 227
- Cherry picking, 124
- Chicago Board Options Exchange (CBOE) shut down, real-world example, 329
- CI (configuration item)
 - change status, tracking, 202
 - versus* CI (continuous integration), 151*n*
 - naming conventions, 203
 - status accounting, 203
 - version IDs, embedding, 97
 - version IDs, verifying, 151

- CI (continuous integration)
 - across the enterprise, 135–136
 - blaming, 132–133
 - build and deploy framework, 129
 - challenges of, 123–124
 - cherry picking, 124
 - code reviews, 127
 - collaboration, 131–132
 - communication, 131–132
 - continuous deployment, 138
 - definition, 121
 - deployment, 136
 - fingering, 132–133
 - getting started, 123
 - goals of, 121–122
 - identifying milestone releases, 138
 - importance of, 122–123
 - integrating smaller units, 126
 - late-binding integration, 122, 124
 - Lean processes, 137–138
 - left-shift preflight builds, 129
 - merges, problems with, 125
 - overview, 21–22
 - preflight builds, 129
 - principles of, 123
 - rapid iterative development, 86–87
 - real-world example, 40
 - rebasings, 125
 - right-shift preflight builds, 129
 - server, selecting, 134–135
 - in the software development process, 39
 - testing, 136
 - traceability, 130–131
 - training and support, 136
 - tuning, 137–138
 - vendor-provided resources, 129
 - version control, 124–125
- CI (continuous integration)
 - versus* CI (configuration item), 151*n*
- CI (continuous integration), build farms
 - cloud computing, 128–129
 - definition, 127
 - ON-PREM (on premises) hypervisors, 128–129
 - real-world example, 128
 - virtualization, 128–129
- CI (continuous integration), frequency
 - benefits of, 126–127
 - best practices, 124–125
 - broken builds, fixing, 127
 - finding issues, 126
 - nightly builds, 133–134
 - problems, fixing, 126–127
- CI (continuous integration), real-world
 - examples
 - build farms, 128
 - information overload, 131
 - merges, 122
 - off-shore support and collaboration, 132
 - process managers, 137
 - stock trading, 124
 - tax preparation, 134
- CI (continuous integration), tools for
 - CI server, selecting, 134–135
 - shared repositories, selecting, 135
- CIRT (critical incident response team), 189–190
- CIS (Center for Internet Security), 209
- Cloud capabilities, 287–288
- Cloud-based ALM
 - build farms, 128–129
 - change management, 290–292
 - cloud capabilities, 287–288
 - CMDB (configuration management database), 296
 - community editions of vendor tools, 287
 - cost control, 296
 - customer interface, 293–294
 - development environments, 295
 - DevOps, 296
 - DML (definitive media library), 296
 - environment management, 295–296
 - getting started, 286–287
 - goals of, 285–286
 - gold copies, 295–296
 - importance of, 286
 - IT operations, 209
 - iterative development, 293
 - managing the lifecycle, 292
 - overview, 24–25
 - PaaS (Platform-as-a-Service), 287
 - planning, 296
 - risk management, 294
 - SaaS (Software-as-a-Service), 287, 293

- Cloud-based ALM, *continued*
 - seamless integrations, 292–293
 - service provider change notification, 291
 - SLAs (service-level agreements), 294
 - test environments, 295
 - tools, 292
- Cloud-based ALM, developing in the cloud
 - build management, 289
 - canary deployment, real-world example, 290
 - deployment, 290
 - nonrepudiation, 290
 - overview, 288
 - release engineering, 289–290
 - source code management, 288–289
- Cloud-based ALM, real-world examples
 - bad service, 292
 - upselling, 292
- CM (configuration management)
 - assessment, 263
 - in ISACA Cobit, 205
 - source code management, 17–18
- CMDB (configuration management database), 115, 296
- Cobit as framework for IT controls, 280
- Code quality instrumentation, automating, 111–112
- Code reviews, CI (continuous integration), 127
- Code scans, 100
- Collaboration. *See also* Communication; DevOps.
 - CI (continuous integration), 131–132
 - DevOps developers and operations, 216–218
 - off-shore support, real-world example, 132
- Collective unconscious, 318–319
- Command center for change management, 169–170
- Commercial off-the-shelf (COTS) software, 32
- Commercial tools *versus* open source, 106–107
- Committee of Sponsoring Organizations (COSO), 279
- Communication. *See also* Collaboration; DevOps; Personality and ALM.
 - anecdote: the ship and the lighthouse, 45
 - change management, 165–166
 - CI (continuous integration), 131–132
 - delivering bad news, 238
 - with management, real-world example, 14–15
 - planning, 197–198
 - rhythms, 319
 - siloed mentality, 44–45
 - with stakeholders, 44–45
 - styles, 317
 - transparency to senior management. *See* IT governance.
 - up the chain of command, 264–265
- Compile dependencies, build engineering, 98
- Complexity management
 - rapid iterative development, 86
 - in the software development process, 33–34
- Compliance, change management, 164
- Comprehensive processes, agile process maturity, 66–67
- Computer-aided software engineering (CASE), 53
- Configuration audits, 203, 228
- Configuration change control, standards and frameworks, 203
- Configuration control, 178–179, 228
- Configuration identification, standards and frameworks, 203
- Configuration item (CI). *See* CI (configuration item).
- Configuration management (CM). *See* CM (configuration management).
- Configuration Management Best Practices: Practical Methods that Work in the Real World*, 18, 348
- Configuration management database (CMDB), 115, 296
- Configuration verification, standards and frameworks, 203
- Conflicts, DevOps developers and operations, 216
- Consensus within the agile community, agile process maturity, 71

- Consistency
 - across the enterprise, agile process maturity, 78–79
 - of purpose, 48
 - Container-based deployment, 144–145, 227–228
 - Continuous delivery
 - versus* continuous deployment, 22, 139–140
 - feature toggle, 22, 139
 - hiding new features from the users. *See* Feature toggle.
 - in the software development process, 41
 - Continuous deployment (CD). *See* CD (continuous deployment).
 - Continuous integration (CI). *See* CI (continuous integration).
 - Continuous process improvement. *See also* Retrospectives, as process improvement.
 - agile process maturity, 69, 79
 - change management, 183–184
 - DevOps, 231
 - IT governance, 270
 - IT operations, 200
 - in the software development process, 48
 - Continuous testing, 311
 - Controlled isolation, rapid iterative development, 85–86
 - Copying files, continuous deployment, 142
 - Core configuration management, overview, 17–21
 - Corporate politics, retrospectives, 245
 - COSO (Committee of Sponsoring Organizations), 279
 - Cost control, cloud-based ALM, 296
 - COTS (commercial off-the-shelf) software, 32
 - Crisis management, retrospectives, 243–244
 - Critical incident response team (CIRT), 189–190
 - Cross-enterprise coordination, change management, 180–181
 - Cross-functional teams, 220–221
 - Cross-platform coordination, change management, 180
 - Cryptographic hashes, 96
 - Csikszentmihalyi, Mihaly, 335–336
 - Customer collaboration, in agile ALM, 58–59
 - Customer interface, cloud-based ALM, 293–294
 - Customers, retrospective participation, 240
 - Cutting corners, real-world example, 44
 - Cybersecurity and the future of ALM, 348–349
- ## D
- Data processing director, real-world example, 149
 - Database administrators, real-world example of communication with, 198
 - The deaf, real-world example of delivering retrospectives, 238–239
 - Defect triage with retrospectives, 243
 - Defects, linking to requirements, 110
 - Definitive media library (DML), 115, 296
 - Delivering on time within budget, agile process maturity, 77
 - Deming, W. Edwards
 - consistency of purpose, 48
 - importance of healthy behaviors, 335–336
 - productivity, 78
 - quality management, 77–78
 - testing *versus* building quality in, 77–78
 - Deming, W. Edwards, driving out fear
 - agile transformation, 72
 - communicating up the chain of command, 265
 - communicating with stakeholders, 44
 - fear of criticism, 224
 - organizational culture, 328
 - planning, 68
 - testing requirements, 34
 - Dependencies, identifying for continuous deployment, 152
 - Dependency control, DevOps, 227–228
 - Deployment
 - automating, 225–227
 - automation, DevOps, 225
 - best practices, 20–21
 - CI (continuous integration), 129, 136

- Deployment, *continued*
 - cloud-based ALM, 290
 - continuous. *See* CD (continuous deployment).
 - goal of, 21
 - rolling back a promotion, 20–21
- Deployment pipeline, 141–142, 225–227
- Designing circuit chips, real-world example, 58
- Designing systems, in the software development process, 37–38
- Developer and operations collaboration, real-world example, 217, 218
- Developer view, on transitioning from hybrid agile to agile, 256
- Developers, retrospective participation, 240
- Developing software. *See* Software development process.
- Development environments, cloud-based ALM, 295
- Development lifecycle, *versus* ALM, 9
- DevOps
 - agile development, 221–222
 - automating, 115–116
 - automating deployment, 225–227
 - automation, 230
 - business continuity, 230
 - cloud-based ALM, 296
 - complexity, 230
 - configuration audits, 228
 - configuration control, 228
 - container-based deployments, 227–228
 - continuous process improvement, 231
 - cross-functional teams, 220–221
 - for customers, 59
 - dependency control, 227–228
 - deployment automation, 225
 - deployment pipeline, 225–227
 - developers and operations, collaboration and conflicts, 216–218
 - in development, 224–227
 - disaster recovery, 230
 - document review, 218
 - driving out silo mentality, 119
 - getting started, 214–215
 - goals of, 213
 - implementing, 215–216
 - importance of, 214
 - information security, 229
 - infrastructure as code, 229–230
 - IT operations, 200
 - knowledge management, 219–220
 - mainframe ALM, 302
 - managing power and influence, 321–323
 - microservices, 227
 - need for rapid change, 218–219
 - organizational ecosystem, 222–223
 - overview, 23
 - positive psychology, 342–344
 - QA (quality assurance), 229
 - retrospectives, 241
 - secure trusted application base, 228
 - in the software development process, 43–44
 - stakeholders, earlier involvement, 223–224
 - team size, 219
 - two-pizza theory, 219
 - waterfall development, 222
- DevOps, moving the process upstream
 - left-shift, 223–224
 - overview, 223
 - right-shift, 224
- DevOps, real-world examples
 - AFB (American Foundation for the Blind), 218
 - attended automation, 226
 - chaos monkeys, 227
 - cross-functional teams, 221
 - deployment automation, 226
 - developer and operations collaboration, 217, 218
 - DevOps in development, 225
 - document review, 218
 - earlier team involvement, 223
 - getting started, 214, 215
 - implementing DevOps, 215–216
 - knowledge management, 220
 - management, effects on team behavior, 221
 - moving the process upstream, 223
 - team size, 219
 - two-pizza theory, 219
 - volleyball behaviors, 221
 - waterfall development, 222

- DevOps in development, real-world example, 225
 - Disaster recovery, 230
 - Disciplined Agile Delivery, 347
 - Disk space shortage, troubleshooting, 189
 - DML (definitive media library), 115, 296
 - Document review, 218
 - Documentation
 - agile ALM, 60
 - on an ambulance, real-world example, 12
 - requirements for transitioning from hybrid agile to agile, 257
 - in the software development process, 42–43
 - working software over comprehensive documentation, 56, 110–111
 - writing, real-world example, 43
 - Dress rehearsal, continuous deployment, 154–155
 - Driving out fear
 - agile transformation, 72
 - communicating up the chain of command, 265
 - communicating with stakeholders, 44
 - fear of criticism, 224
 - organizational culture, 328
 - planning, 68
 - testing requirements, 34
 - Dysfunctional agile, agile process maturity, 65
- E**
- Eccentric behavior in the workplace, 333–335
 - E-change control, 179–180
 - Embedding testers, 312
 - Emergency change control, 179
 - Emergency medical tech, real-world example, 142
 - Emperor’s New Clothes, anecdote, 71–72
 - Enterprise
 - agile process maturity, 78–79
 - cross-enterprise change management, 180–181
 - Environment management
 - automating, 114–115
 - best practices, 19
 - cloud-based ALM, 295–296
 - overview, 19
 - Epics and stories
 - agile process maturity, 70
 - automating creation of, 111
 - definition, 70
 - in the software development process, 36
 - Ergonomics, continuous deployment, 150
 - Escalating problems, change management, 172–173
 - Event monitoring, change management, 168–169
 - Events
 - definition, 168
 - monitoring, 188–189
 - External audits, 277
 - Extremism in the workplace, 333–335
- F**
- Facilitating training, in the software development process, 47–48
 - Facilities management, IT operations, 207
 - Failing fast, definition, 94–95
 - False positives, 96–97
 - Family vacation, real-world example of hybrid agile, 255
 - Feature toggle, 22, 139
 - Federal guidelines. *See* Audit and regulatory compliance, federal guidelines.
 - Feedback
 - change management, real-world example, 171–172
 - from change management, 171–172
 - Feedback loops, change management, 171
 - Fiefdoms, change management, 181
 - Financial systems infrastructure, real-world example, 14
 - Fingering, 132–133
 - Finley, Michael, 332
 - FINRA (Financial Industry Regulatory Authority, Inc.), 280
 - Firmware
 - aligning software to, real-world example, 84
 - development, agile process maturity, 80

Five-factor model of intergroup conflict, 323–324
 Fixed timebox sprints, 57–58
 Fixing what isn't broken, real-world example in retrospectives, 235
 Flooding in an IT facility, real-world example, 207
 Football player, real-world example of retrospectives, 236–237
 Friedman, Meyer, 331
 Functional requirements, 35–36
 Functional testing, 39
 Future of ALM, 347–349

G

GAO (Government Accountability Office), 281–282
 Gatekeeping, 177
 Gold copies, 114–115, 295–296
 Government agency, real-world example of change management, 181
 GPS upgrade, real-world example of change management, 183
 Group dynamics. *See* Personality and ALM, group dynamics.

H

Hackers, 95
 Hardware development, agile process maturity, 80
 Health Insurance Portability and Accountability Act (HIPAA) (1966), 280–281
 Hedge fund trading systems, real-world examples
 IT governance, 268
 IT operations, 188
 Help desks. *See also* Service desks.
 automating, 116–117
 avatars, real-world example, 194–195
 real-world example, 193
 virtual, real-world example, 194–195
 Help desks, IT operations
 developers on, 195
 overview, 192–193
 remote work, 194
 virtual, 193–195
 Hibernate, real-world example, 153–154
 Hidden agile, real-world example, 250

Hierarchy of needs and drives, 339
 HIPAA (Health Insurance Portability and Accountability Act) (1966), 280–281
 Hybrid agile
 coexisting with non-agile projects, 68
 definition, 15, 249
 getting started, 250–251
 goals of, 249
 importance of, 250
 pragmatic choices, 251
 versus waterfall method, 251–252, 254, 256

Hybrid agile, real-world examples
 family vacation, 255
 hidden agile, 250
 making a baby in one month, 254
 management decision making, 258
 measuring agility, 252
 Hybrid agile, transitioning to agile
 agile disasters, 255
 choosing an agile pilot, 253
 decisions at last responsible moment, 257
 defining requirements, 254
 developer view, 256
 documenting requirements, 257
 information radiators, 256
 IT governance requirements, 258
 mature agile, 258
 organizational ecosystem, 257–258
 overview, 252–253
 securing sensitive information, 256
 technology risk, 257
 tracking progress, 255
 versus waterfall method, 256

I

IDEs (integrated development environments), 93
 Immature processes, recognizing, 72
 Incident escalation, real-world example, 199
 Incident management, automating, 117
 Incident response, 170–172, 190
 Incidents. *See also* Problems.
 CIRT (critical incident response team), 189–190
 escalating, 198–200

- identifying with retrospectives, 242–243
- IT operations, 212
- monitoring, 189–190
- versus* problems, 172–173
- retrospectives, 236–237
- Information overload, real-world example, 131
- Information radiators, 256
- Information security, DevOps, 229
- Infrastructure as code, 229–230
- In-group behaviors, 320–321
- Instrumenting code, 101
- Insurance company use of RAD, real-world example, 53
- Integrated development environments (IDEs), 93
- Integration across the enterprise
 - coordinating across systems, 307
 - enterprise ecosystem, 308
 - getting started, 306–307
 - goals of, 305
 - importance of, 305–306
 - interfaces, 307–308
 - multiplatform, 307
 - overview, 25
 - procurement and standards, real-world example, 306
 - release coordination, 308
- Intergroup conflict. *See* Personality and ALM, intergroup conflict.
- Internal audits, 275–276
- International corporations, cultures of, 317
- Introspection and the postmortem, 327–329
- ISACA Cobit, 205, 281
- ISO 12207, 30–31
- ISO 15288, 32
- IT controls, audit and regulatory compliance, 275–276
- IT governance
 - agile process maturity, 67–68, 75–76
 - audit and regulatory compliance, 271
 - communicating up the chain of command, 264–265
 - continuous process improvement, 270
 - delays, 268–269
 - getting started, 262–263
 - goals of, 261–262
 - importance of, 262
 - learning from mistakes, 270
 - organizational ecosystem, 270
 - overview, 23
 - requirements for transitioning hybrid agile to agile, 258
 - retrospectives, 244
 - risk management, 266–267
 - scalability and resources, 268
 - time and resource management, 267–268
 - workload assessment, 265–266
- IT governance, real-world examples
 - configuration management assessment, 263
 - hedge funds, 268
 - police force, 265
 - reporting risks, 267
 - senior management, best practices, 270
 - senior management, decision making, 263
 - senior management, role of, 264
 - tool selection, 266
 - trading firms, 268
- IT governance, senior management
 - communicating up the chain of command, 264–265
 - decision making, 263
 - excessive direct involvement, 269
 - reporting risks, 267, 269
- IT operations
 - application management, 208
 - automating, 116
 - automation, 195–196
 - business management, 205–206
 - change management, 205–206
 - CIRT (critical incident response team), 189–190
 - CIS (Center for Internet Security), 209
 - cloud based, 209
 - communication planning, 197
 - continuous process improvement, 200
 - controls, 206
 - DevOps, 200
 - facilities management, 207
 - getting started, 186–187
 - goals of, 185–186

- IT operations, *continued*
 - importance of, 186
 - incident escalation, 198–200
 - incidents, 212
 - interfacing with vendor operations, 209
 - knowledge management, 195–196, 212
 - management, 206–207
 - middleware support, 208
 - organizational silos, 197
 - outsourcing, 209
 - overview, 22–23
 - problem escalation, 198–200
 - problems, 212
 - product management, 205–206
 - production support, 191–192
 - retrospective participation, 241
 - security, 208–209
 - shared services, 208
 - technical management, 206
 - workflow automation, 196
- IT operations, help desks. *See also* IT operations, service desks.
 - developers on, 195
 - overview, 192–193
 - remote work, 194
 - virtual, 193–194
 - virtual world, 194–195
- IT operations, monitoring the environment
 - events, 188–189
 - incidents, 189–190
 - problem management, 190–191
- IT operations, real-world examples
 - agile principles, 210
 - agile service catalog, 186
 - change management, 205–206
 - communication planning, 197–198
 - database administrators, communication with, 198
 - escalating problems and incidents, 199
 - fixing what's not broken, 187
 - flooding, 207
 - hedge fund trading systems, 188
 - help desk avatars, 194–195
 - help desks, 193
 - incident response, 190
 - IT facilities management, 207
 - KCG (Knight Capital Group), 187
 - knowledge management, 195–196
 - mainframe programmers, 191–192
 - offshoring production support, 191–192
 - outsourcing service desks, 211–212
 - rebooting the system, 190
 - segregation of duties, 207
 - standards and frameworks, 201
 - troubleshooting disk space shortage, 189
 - VCS (version control system) failure, 197
 - virtual help desks, 194–195
 - workflow automation, 196
 - working across time zones, 193
- IT operations, service desks. *See also* IT operations, help desks.
 - agile principles, 210
 - centralized, 210
 - outsourcing, 211–212
 - overview, 210
 - specialized, 211
 - staffing, 211–212
 - vendor escalation, 211
 - virtual, 211
- IT operations, standards and frameworks
 - CAB (change advisory board), 202
 - change evaluation, 204
 - change management, 205
 - change management processes, 202
 - configuration audit, 203
 - configuration change control, 203
 - configuration identification, 203
 - configuration management, 205
 - configuration verification, 203
 - ISACA Cobit, 205
 - ITIL v3, 201–204
 - knowledge management, 204–205
 - need for, 201
 - overview, 201
 - RCV (release control and validation framework). *See* ITIL v3.
 - RDM (release and deployment management), 203–204
 - request fulfillment, 204
 - SACM (service asset and configuration management), 202–203

- SCMP (software configuration management plan), 203
 - service management processes. *See* ITIL v3.
 - status accounting, 203
 - Items on the left, agile manifesto, 56
 - Items on the right, agile manifesto, 56
 - Iterative development, 16–17, 293
 - ITIL, retrospectives, 242–243
 - ITIL v3, 201–204
- J**
- Jung, Carl, 319
- K**
- Kanban, in continuous deployment, 148–150
 - KCG (Knight Capital Group), real-world example, 187
 - Keyman risk, 108, 317
 - Knowledge base, creating in the software development process, 47–48
 - Knowledge management
 - DevOps, 219–220
 - IT operations, 212
 - real-world example, 195–196
 - standards and frameworks, 204–205
- L**
- Last responsible moment
 - decisions on transitioning from hybrid agile to agile, 257
 - definition, 118, 257
 - planning decisions, 68
 - Late-binding integration, 122, 124
 - Law enforcement, real-world process
 - maturity example, 66
 - Lean processes, CI (continuous integration), 137–138
 - Learned complacency, 329–331
 - Learned helplessness, 325–327
 - Leffingwell, Dean, 347
 - Left-shift
 - moving the process upstream, 223–224
 - preflight builds, 129
 - Lifecycle management
 - automating, 109
 - cloud-based ALM, 292
 - Lifecycle phases, defining in the software development process, 41–42
 - Lifecycle testing, automating, 112
 - Lifeguard rule, QA and testing real-world example, 310
 - The lighthouse and the ship, anecdote, 45
- M**
- Mainframe ALM
 - DevOps, 302
 - getting started, 300–302
 - goals of, 299
 - importance of, 299–300
 - overview, 25
 - Mainframe ALM, real-world examples
 - bypassing change control, 301
 - defining the ALM, 300–301
 - mainframe culture, 300
 - outages, 171
 - programmers, 191–192
 - root access, 302
 - tribal knowledge, 300–301
 - Mainframe culture, 300
 - Maintenance and bugfixes, in the software development process, 46
 - Maintenance of the lifecycle, in the software development process, 47
 - Making a baby in one month, hybrid agile real-world example, 254
 - Management. *See also* Senior management.
 - decision making, hybrid agile real-world example, 258
 - effects on team behavior, real-world example, 221
 - traits of strong leaders, 336
 - Marketing the new agile approach, 79
 - Martin, James, 53
 - Maslow, Abraham, 339
 - Mature agile
 - hybrid agile, transitioning to agile, 258
 - one CIO’s view of agile process maturity, real-world example, 73
 - Mature processes *versus* fluid, 16–17
 - Maven, real-world example, 153–154
 - Measuring agility, real-world example, 252
 - Meetings, retrospectives, 241

- Merges
 continuous integration problems with, 125
 real-world example, 122
- Metrics
 measuring agility, real-world example, 252
 measuring the ALM, 79–80
 PMO (project management office)
 metrics, 80
 retrospectives, 245
- Microservices, 227
- Middleware support, IT operations, 208
- Milestone releases, identifying, 138
- Mistakes
 as feedback loops, retrospectives, 236, 237
 management reaction to, 327
- Mistakes, learning from
 crises as opportunities, 48
 IT governance, 270
 in a police force, real-world example, 52
 positive psychology of, 340–342
- Monitoring continuous deployment, real-world example, 152
- Motivation through threats, 334
- Moving the process upstream
 left-shift, 223–224
 overview, 223
 real-world example, 223
 right-shift, 224
- N**
- New York Stock Exchange crash, 97–98
- Nonfunctional requirements, 36
- Nonrepudiation, 290
- Normal changes, 175, 180
- Nuclear power plant
 continuous deployment, real-world example, 150
 testing, real-world example, 39
- O**
- OCC (Office of the Comptroller of the Currency), 282
- OCEAN (openness, conscientiousness, extraversion, agreeableness, neuroticism) model of intergroup conflict, 323–324
- Off-shore support and collaboration, real-world example, 132
- Offshoring production support, real-world example, 191–192
- ON-PREM (on premises) hypervisors, 128–129
- Open source tools *versus* commercial, 106–107
- Operations. *See* IT operations.
- Organizational culture, agile ALM, 51–52
- Out-group behaviors, 320–321
- Outsourcing
 IT operations, 209
 service desks, 211–212
- Overly agreeable people, 323–325
- Oxley, Michael, 278
- P**
- PaaS (Platform-as-a-Service), 287
- Paradigm shift for agile ALM, 51–52
- Personality and ALM
 archetypes, 319
 collective unconscious, 318–319
 communication rhythms, 319
 communication styles, 317
 goals of, 315
 importance of, 315–316
 international corporations, 317
 keyman risk, 317
 managerial conflicts, real-world example, 316
 organizational structures, 317–318
 in retrospectives, 237
 role of, overview, 26
- Personality and ALM, getting started
 organizational psyche, 318–319
 understanding the culture, 316–318
- Personality and ALM, group dynamics
 driving out silos, 320–321
 in-group and out-group behaviors, 320–321
 managing power and influence, 321–323
 overview, 320
- Personality and ALM, intergroup conflict
 desired personality traits, 328
 five-factor model, 323–324
 introspection and the postmortem, 327–329

- learned helplessness, 325–327
- management reaction to mistakes, 327
- OCEAN (openness, conscientiousness, extraversion, agreeableness, neuroticism) model, 323–324
- overly agreeable people, 323–325
- Personality and ALM, positive psychology
 - autonomy, 339
 - benefits of, 335–337
 - in DevOps, 342–344
 - hierarchy of needs and drives, 339
 - learning from mistakes, 340–342
 - pillars of, 337–338
 - team motivation, 339–340
 - traits of strong leaders, 336
- Personality and ALM, stress management
 - aggressive team members, 331–333
 - eccentric behavior in the workplace, 333–335
 - extremism in the workplace, 333–335
 - learned complacency, 329–331
 - motivation through threats, 334
 - personality types, 331–333
 - type A and B personalities, 331–333
- Personality types, 331–333
- Physical configuration audit, 98, 115
- Pilot system. *See* Proof of technology.
- Planning
 - agile process maturity, 68–69
 - as a barrier to efficiency, 5
 - cloud-based ALM, 296
 - testing processes, 311–313
- Platform-as-a-Service (PaaS), 287
- Platforms, cross-platform change management, 180
- PMO (project management office)
 - automating, 118
 - metrics, 80
- POC (proof-of-concept), 106, 119
- Police force, real-world examples
 - continuous deployment, 149
 - IT governance, 265
- Positive psychology. *See* Personality and ALM, positive psychology.
- Postmortems, introspection, 327–329
- Preapproved changes, 174, 175, 180
- Preflight builds, 129
- Principles, of agile process maturity, 64–65
- Probing and questioning, retrospectives, 241
- Problem escalation
 - automating, 117–118
 - IT operations, 198–200
 - real-world example, 199
- Problem management
 - change management, 172–173
 - CIRT (critical incident response team), 189–190
- Problems. *See also* Incidents.
 - CIRT (critical incident response team), 189–190
 - escalating, 198–200
 - identifying with retrospectives, 242–243
 - versus* incidents, 172–173
 - IT operations, 212
 - learning from, real-world example, 173
 - monitoring, 189–191
 - versus* problems, 172–173
 - retrospectives, 236–237
- Process change control, 179
- Process managers, real-world example, 137
- Process maturity. *See* Agile process maturity.
- Process modeling, automating, 108
- Processes
 - adjusting ceremony, 75
 - testing, 39
- Product management, IT operations, 205–206
- Production support
 - IT operations, 191–192
 - in the software development process, 45–46
- Productivity, improving through audit and regulatory compliance, 283
- Project management, automating, 118
- Project management office (PMO)
 - automating, 118
 - metrics, 80
- Proof of technology, real-world example, 8
- Proof-of-concept (POC), 106, 119
- Psychology of personality. *See* Personality and ALM, positive psychology.

Publishing changes back to the system.
See Rebasing.
 Purity measure, agile process maturity,
 64

Q

QA (quality assurance). *See also* Testing.
 change management, 167–168
 continuous testing, 311
 DevOps, 229
 ensuring quality, 313–314
 getting started, 310–311
 goals of, 309
 importance of, 309–310
 overview, 25
 planning the testing process, 311–313
 test cases, creating, 313
 withholding information from, real-
 world example, 113
 QA (quality assurance), real-world
 examples
 bypassing quality assurance, 311
 embedding testers, 312
 first rule for lifeguards, 310
 testing framework, creating, 312
 Quality
 building in, build engineering, 100
 building in *versus* testing, 77
 improving through audit and regula-
 tory compliance, 283

R

RAD (rapid application development),
 52–54
 Rapid incremental deployment, 143–144
 Rapid iterative development
 agile ALM, 52–53
 in ALM, overview, 17
 CI (continuous integration), 86–87
 controlled isolation, 85–86
 designing architecture, 87
 development view, 85
 getting started, 84–85
 goals of, 83
 importance of, 84
 managing complexity, 86
 technical risk, 85, 87
 technology, 87
 VCS (version control system), 87

RDM (release and deployment manage-
 ment), 203–204
 Repeatability, continuous deployment,
 147–148
 Rebasing, 125
 Rebooting the system, real-world exam-
 ple, 190
 Recognition by the agile community,
 agile process maturity, 70–71
 Red tape. *See* Ceremony.
 Regulatory compliance. *See* Audit and
 regulatory compliance.
 Release engineering, cloud-based ALM,
 289–290
 Release management
 best practices, 19–20
 goal of, 20
 overview, 19–20
 Repeatable processes, agile process matu-
 rity, 76–77
 Request fulfillment, standards and frame-
 works, 204
 Requests for change (RFCs), 177–178
 Requirements
 agile ALM, 59–60
 agile process maturity, 69–70
 ALM effect on, 12–13
 for audit and regulatory compliance,
 283
 linking to defects, 110
 in the software development process.
See Software development pro-
 cess, requirements.
 tracking to test cases, 110
 for transitioning from hybrid agile to
 agile, 254, 257
 Requirements management, automating,
 110–111
 Resource and time management, IT gov-
 ernance, 267–268
 Responding to change over valuing a
 plan, 56, 118
 Retrospectives. *See also* Reviews.
 audit and regulatory compliance,
 244
 corporate politics, 245
 as crisis management, 243–244
 defect triage, 243
 DevOps, cross-functional view, 241

- epics and stories, 241
 - getting started, 234
 - goals of, 234
 - importance of, 234
 - incidents and problems, 242–243
 - as leadership, 241–242
 - metrics and measurement, 245
 - overview, 23
 - probing and questioning, 241
 - red tape, 245
 - risk management, 244
 - running the meeting, 241
 - supporting IT governance, 244
 - supporting ITIL, 242–243
 - use cases, 241–242
 - vendor management, 244–245
 - Retrospectives, as process improvement. *See also* Continuous process improvement.
 - assessing success, 235–236
 - delivering bad news, 238
 - incidents and problems, 236–237
 - mistakes as feedback loops, 236, 237
 - overview, 235
 - personality factors, 237
 - Retrospectives, delivery modes
 - online, 239
 - in person, 238–239
 - teleconference, 239
 - video conferencing, 239
 - virtual worlds, 239–240
 - Retrospectives, participant perspective
 - customers, 240
 - developers, 240
 - operations, 241
 - testers, 240
 - Retrospectives, real-world examples
 - baseball players and mistakes, 236
 - delivery by the blind or deaf, 238–239
 - fixing what isn't broken, 235
 - football player, 236–237
 - mistakes as feedback loops, 236
 - Reviews. *See also* Retrospectives.
 - after change management, 175
 - code, 127
 - document, 218
 - post-implementation, 175
 - RFCs (requests for change), 177–178
 - Right-shift
 - moving the process upstream, 224
 - preflight builds, 129
 - Risk assessment
 - change management, 164–165
 - continuous deployment, 153–154
 - vendor risks, 32
 - Risk management
 - change management, 164–165
 - cloud-based ALM, 294
 - continuous deployment, 153–154
 - IT governance, 266–267
 - retrospectives, 244
 - self-administered risk assessment surveys, 280
 - Risks
 - from a business focus, 13–15
 - cloud-based resources, 129
 - keyman, 108
 - reporting, real-world example, 267
 - technical risk, rapid iterative development, 85, 87
 - vendor-provided resources, 129
 - Robbins, Harvey, 332
 - Root access, mainframe ALM, 302
 - Rosenman, Ray, 331
 - Royce, Winston, 256
 - Rubin, Ken, 347
- ## S
- SaaS (Software-as-a-Service), 287, 293
 - SaaS change control, change management, 182–183
 - SACM (service asset and configuration management), 202–203
 - SAFE (Scaled Agile Framework), 347
 - Sarbanes, Paul, 278
 - Sarbanes-Oxley Act of 2002, 278–280
 - Sarin gas, real-world example of continuous deployment, 154–155
 - Scalability
 - agile process maturity, 66, 77
 - and resources, IT governance, 268
 - Scientific Management, 10
 - SCMP (software configuration management plan), 203
 - Scope, of ALM, 9
 - Scope creep, real-world example, 11

- SDLC (software development life cycle), 29. *See also* ALM (application lifecycle management); Software development process.
 - developing, real-world example, 42
 - managing, real-world example, 30
 - overview, 10–11
 - versus* software development process, 29–31
 - versus* systems development, 32
- Seamless integrations, cloud-based ALM, 292–293
- Secure trusted base
 - creating, 95–96
 - DevOps, 228
- Securities firms, federal guidelines on oversight of, 280
- Security
 - CIS (Center for Internet Security), 209
 - cryptographic hashes, 96
 - cybersecurity and the future of ALM, 348–349
 - detecting unauthorized changes, 96–97
 - false positives, 96–97
 - hackers, 95
 - information security, DevOps, 229
 - IT operations, 208–209
 - physical configuration audit, 98, 115
 - securing sensitive information, 256
- Segregation of duties, real-world example, 207
- Self correction, agile process maturity, 79
- Seligman, Martin, 326, 335–336, 337, 343
- Senior management of banking, federal guidelines on responsibility, 278–280. *See also* Management.
- Senior management of IT governance
 - communicating up the chain of command, 264–265
 - decision making, 263
 - excessive direct involvement, 269
 - reporting risks, 267, 269
- Senior management of IT governance, real-world examples
 - best practices, 270
 - decision making, 263
 - role of, 264
- SEPG (software engineering process group), 47, 166, 179
- Service asset and configuration management (SACM), 202–203
- Service desk, automating, 117
- Service desks. *See also* Help desks.
 - agile principles, 210
 - centralized, 210
 - outsourcing, 211–212
 - overview, 210
 - specialized, 211
 - staffing, 211–212
 - vendor escalation, 211
 - virtual, 211
- Service providers, real-world example of change management, 183
- Service-level agreements (SLAs), 294
- Shared services, IT operations, 208
- The ship and the lighthouse, anecdote, 45
- Silo mentality
 - business silos, overview, 13
 - communication, 44–45
 - driving out, 119, 320–321
 - IT operations, 197
 - when selecting automation tools, 119
- SLAs (service-level agreements), 294
- Slinger, Michele, 6
- Smoke testing, continuous deployment, 156–157
- Software, real-world example of aligning to firmware, 84
- Software configuration management plan (SCMP), 203
- Software delivery lifecycle, *versus* ALM, 7
- Software development life cycle (SDLC). *See* SDLC (software development life cycle).
- Software development process. *See also* ALM (application lifecycle management); SDLC (software development life cycle).
 - change control, 47
 - CI (continuous integration), 39
 - continuous delivery, 41
 - continuous deployment, 41
 - continuous process improvement, 48
 - creating the knowledge base, 47–48
 - creating the right size process, 46

- cutting corners, real-world example, 44
 - designing systems, 37–38
 - DevOps, 43–44
 - documentation, 42–43
 - facilitating training, 47–48
 - lifecycle phases, defining, 41–42
 - maintenance and bugfixes, 46
 - maintenance of the lifecycle, 47
 - production support, 45–46
 - SEPG (software engineering process group), 47
 - software development, 38
 - technical debt, 46
 - Software development process, defining
 - change management, 33–34
 - complexity management, 33–34
 - COTS (commercial off-the-shelf) software, 32
 - epics and stories, 36
 - getting started, 29
 - goals of, 27–28
 - importance of, 28
 - SDLC (software development life cycle), 29
 - versus* SDLC (software development life cycle), 29–31
 - test cases, real-world example, 35
 - use cases, 35–36
 - vendor risk analysis, 32
 - Software development process, requirements
 - change planning, 36
 - defining, 32–33
 - functional, 35–36
 - nonfunctional, 36
 - testing, 35
 - validity, 34
 - workflow for defining, 37
 - Software development process, testing. *See also* QA (quality assurance); TDD (test-driven development); Testing.
 - applications, 39
 - functional, 39
 - nuclear power plants, real-world example, 39
 - overview, 38–39
 - processes, 39
 - unit, 39
 - Software engineering process group (SEPG), 47, 166, 179
 - The Software Project Manager's Bridge to Agility*, 6
 - Software-as-a-Service (SaaS), 287, 293
 - Source code management
 - best practices, 17–18
 - cloud-based ALM, 288–289
 - goal of, 18
 - overview, 17–18
 - Specialized change control, 182
 - Specialized service desks, 211
 - Staffing service desks, 211–212
 - Stakeholders
 - communicating with, 44–45
 - earlier involvement, 223–224
 - Standard changes, 175, 180
 - Standards and frameworks
 - agile process maturity, 68
 - common lifecycle processes. *See* ISO 15288.
 - for IT operations. *See* IT operations, standards and frameworks.
 - real-world example, 201
 - software lifecycle processes. *See* ISO 12207.
 - Status accounting, standards and frameworks, 203
 - Stock trading, real-world continuous integration example, 124
 - Stopping the line
 - build engineering, 94–95
 - TDD (test-driven development), 37
 - Storage, real-world example of change management, 162
 - Stories. *See* Epics and stories.
 - Stress management. *See* Personality and ALM, stress management.
 - Sullivan, Harry Stack, 333
 - Systems delivery lifecycle, *versus* ALM, 7
 - Systems design, automating, 111
 - Systems development, *versus* SDLC, 32
 - Systems thinking, definition, 25
- ## T
- Tax preparation, continuous integration
 - real-world example, 134
 - Taylor, Winslow, 10

- TDD (test-driven development). *See also* Testing.
 automated test scripts, 37
 automating, 113–114
 overview, 37
 stopping the line, 37
- Teams. *See also* Personality and ALM.
 aggressive team members, 331–333
 cross-functional, 220–221
 earlier involvement, 223–224. *See also* Moving the process upstream.
 management, effects on team behavior, 221
 motivating, 339–340
 optimal size, 219
 two-pizza theory of team size, 219
- Technical debt
 change management, real-world example, 165
 in the software development process, 46
- Technical management, IT operations, 206
- Technical risk, rapid iterative development, 85, 87
- Technology, rapid iterative development, 87
- Technology risk, transitioning from hybrid agile to agile, 257
- Terminology pollution, 140
- Test case management, automating, 112–113
- Test cases. *See also* Use cases.
 linking to defects, 110
 real-world example, 35
 tracking requirements to, 109
 for trading systems, real-world example, 70
 user written, 59–60
- Test environments, cloud-based ALM, 295
- Test-driven development (TDD). *See* TDD (test-driven development).
- Testers
 embedding, 312
 retrospective participation, 240
 withholding information from, real-world example, 113
- Testing. *See also* QA (quality assurance); Software development process, testing; TDD (test-driven development).
 change management, 167–168
 CI (continuous integration), 136
 continuous testing, 311
 ensuring quality, 313–314
 getting started, 310–311
 goals of, 309
 importance of, 309–310
 overview, 25
 planning the testing process, 311–313
 requirements, 35
 test cases, creating, 313
 testing framework, creating, 312
- Testing, real-world examples
 bypassing testing, 311
 embedding testers, 312
 first rule for lifeguards, 310
- Time and motion studies, 10
- Time and resource management, IT governance, 267–268
- Time zones, working across, 193
- Tool agnosticism, 106
- Tools
 for CI (continuous integration), 134–135
 cloud-based ALM, 287, 292
 selecting for IT governance, real-world example, 266
- Tools, for automation
 build engineering, 101
 commercial *versus* open source, 106–107
 evaluating, 106, 119. *See also* POC (proof-of-concept).
 keeping current, 120
 POC (proof-of-concept), 106, 119
 scope of, 109
 seamless integration, 109–110
 selecting, 119
 tool agnosticism, 106
 uses for, 94
- Traceability
 agile process maturity, 67
 change management, 164
 CI (continuous integration), 130–131
 continuous deployment, 147–148

Trading firms, real-world example of IT governance, 268

Training and support
 CI (continuous integration), 136
 continuous deployment, 147
 programs, developing, 120

Transitioning to agile. *See* Hybrid agile, transitioning to agile.

Transparency, agile process maturity, 67, 76

Tribal knowledge, mainframe ALM, 300–301

Troubleshooting, real-world examples
 change management, 169
 disk space shortage, 189

Trusted base, continuous deployment, 151–152

Tuning, CI (continuous integration), 137–138

Two-pizza theory of team size, 219

Type A and B personalities, 331–333

U

Unauthorized changes, detecting, 96–97

Unit testing, 39

Unit tests, build engineering, 100

Upselling from cloud-based ALM, real-world examples, 292

Use cases. *See also* Test cases.
 automating creating of, 119
 definition, 35–36
 retrospectives, 241–242

User stories. *See* Epics and stories.

Utilities. *See* Tools.

V

Validation, continuous deployment, 150–151

Validity, requirements, 34

VCS (version control system), 87, 124–125, 197

Vendor change control, 182

Vendor management
 agile process maturity, 80
 with retrospectives, 244–245

Vendor operations, interfacing with
 escalating problems, 211
 IT operations, 209

Vendor relationships, 120

Vendor-provided resources, CI (continuous integration), 129

Verification, continuous deployment, 150–151

Version control
 CI (continuous integration), 124–125
 real-world example, 63
 VCS (version control system), 87, 124–125, 197

Version IDs, build engineering, 97–98

Virtual help desks, 193–194

Virtualization, build farms, 128–129

Volleyball behaviors, real-world example, 221

W

Walkthroughs, continuous deployment, 154–155

Waterfall development
 DevOps, 222
 dysfunctional processes, 73–74
versus hybrid agile, 251–252, 254, 256
 pitfalls, 73–74
 predicting the future, 16
 real-world example, 222

Winston, Royce, 57, 73

WIP (work in progress), 149

Workflow, defining requirements, 37

Workflow automation
 continuous deployment, 148–150
 overview, 108

Working software over comprehensive documentation, 56, 110–111

Workload assessment, IT governance, 265–266