Praise for *Agile Testing*

“As Agile methods have entered the mainstream, we’ve learned a lot about how the testing discipline fits into Agile projects. Lisa and Janet give us a solid look at what to do, and what to avoid, in Agile testing.”
—Ron Jeffries, www.XProgramming.com

“An excellent introduction to agile and how it affects the software test community!”
—Gerard Meszaros, Agile Practice Lead and Chief Test Strategist at Solution Frameworks, Inc., an agile coaching and lean software development consultancy

“In sports and music, people know the importance of practicing technique until it becomes a part of the way they do things. This book is about some of the most fundamental techniques in software development—how to build quality into code—techniques that should become second nature to every development team. The book provides both broad and in-depth coverage of how to move testing to the front of the development process, along with a liberal sprinkling of real-life examples that bring the book to life.”
—Mary Poppendieck, Author of *Lean Software Development* and *Implementing Lean Software Development*

“Refreshingly pragmatic. Chock-full of wisdom. Absent of dogma. This book is a game-changer. Every software professional should read it.”
—Uncle Bob Martin, Object Mentor, Inc.

“With *Agile Testing*, Lisa and Janet have used their holistic sensibility of testing to describe a culture shift for testers and teams willing to elevate their test effectiveness. The combination of real-life project experiences and specific techniques provide an excellent way to learn and adapt to continually changing project needs.”
—Adam Geras, M.Sc. Developer-Tester, Ideaca Knowledge Services

“On Agile projects, everyone seems to ask, ‘But, what about testing?’ Is it the development team’s responsibility entirely, the testing team, or a collaborative effort between developers and testers? Or, ‘How much testing should we automate?’ Lisa and Janet have written a book that finally answers these types of questions and more! Whether you’re a tester, developer, or manager, you’ll learn many great examples and stories from the real-world work experiences they’ve shared in this excellent book.”
—Paul Duvall, CTO of Stelligent and co-author of *Continuous Integration: Improving Software Quality and Reducing Risk*

“Finally a book for testers on Agile teams that acknowledges there is not just one right way! *Agile Testing* provides comprehensive coverage of the issues testers face when they move to Agile: from tools and metrics to roles and process. Illustrated with numerous stories and examples from many contributors, it gives a clear picture of what successful Agile testers are doing today.”
—Bret Pettichord, Chief Technical Officer of WatirCraft and Lead Developer of Watir
Agile Testing
Visit informit.com/awss for a complete list of available products.

The Addison-Wesley Signature Series provides readers with practical and authoritative information on the latest trends in modern technology for computer professionals. The series is based on one simple premise: Great books come from great authors. Books in the series are personally chosen by expert advisors, world-class authors in their own right. These experts are proud to put their signatures on the covers, and their signatures ensure that these thought leaders have worked closely with authors to define topic coverage, book scope, critical content, and overall uniqueness. The expert signatures also symbolize a promise to our readers: You are reading a future classic.
To my husband, Bob Downing—you’re the bee’s knees!
—Lisa

To Jack, Dana, and Susan, and to all the writers in my family.
—Janet

And to all our favorite donkeys and dragons.
—Lisa and Janet
This page intentionally left blank
## CONTENTS

<table>
<thead>
<tr>
<th>Foreword by Mike Cohn</th>
<th>xxiii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword by Brian Marick</td>
<td>xxv</td>
</tr>
<tr>
<td>Preface</td>
<td>xxvii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>xxxvii</td>
</tr>
<tr>
<td>About the Authors</td>
<td>xli</td>
</tr>
</tbody>
</table>

### Part I Introduction 1

<table>
<thead>
<tr>
<th>Chapter 1</th>
<th>What Is Agile Testing, Anyway?</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile Values</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>What Do We Mean by “Agile Testing”?</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>A Little Context for Roles and Activities on an Agile Team</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Customer Team</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Developer Team</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Interaction between Customer and Developer Teams</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>How Is Agile Testing Different?</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Working on Traditional Teams</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Working on Agile Teams</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Traditional vs. Agile Testing</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Whole-Team Approach</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 2</th>
<th>Ten Principles for Agile Testers</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s an Agile Tester?</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>The Agile Testing Mind-Set</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>
Applying Agile Principles and Values 21
Provide Continuous Feedback 22
Deliver Value to the Customer 22
Enable Face-to-Face Communication 23
Have Courage 25
Keep It Simple 26
Practice Continuous Improvement 27
Respond to Change 28
Self-Organize 29
Focus on People 30
Enjoy 31
Adding Value 31
Summary 33

Part II Organizational Challenges 35

Chapter 3 Cultural Challenges 37
Organizational Culture 37
Quality Philosophy 38
Sustainable Pace 40
Customer Relationships 41
Organization Size 42
Empower Your Team 44
Barriers to Successful Agile Adoption by Test/QA Teams 44
Loss of Identity 44
Additional Roles 45
Lack of Training 45
Not Understanding Agile Concepts 45
Past Experience/Attitude 48
Cultural Differences among Roles 48
Introducing Change 49
Talk about Fears 49
Give Team Ownership 50
Celebrate Success 50
Management Expectations 52
Cultural Changes for Managers 52
Speaking the Manager’s Language 55
Change Doesn’t Come Easy 56
Be Patient 56
Let Them Feel Pain 56
## CONTENTS

| Build Your Credibility                      | 57 |
| Work On Your Own Professional Development   | 57 |
| Beware the Quality Police Mentality         | 57 |
| Vote with Your Feet                         | 57 |
| Summary                                    | 58 |

### Chapter 4  Team Logistics  59

- Team Structure  59
  - Independent QA Teams  60
  - Integration of Testers into an Agile Project  61
  - Agile Project Teams  64
- Physical Logistics  65
- Resources  66
  - Tester-Developer Ratio  66
  - Hiring an Agile Tester  67
- Building a Team  69
  - Self-Organizing Team  69
  - Involving Other Teams  69
  - Every Team Member Has Equal Value  70
  - Performance and Rewards  70
  - What Can You Do?  71
- Summary  71

### Chapter 5  Transitioning Typical Processes  73

- Seeking Lightweight Processes  73
- Metrics  74
  - Lean Measurements  74
  - Why We Need Metrics  75
  - What Not to Do with Metrics  77
  - Communicating Metrics  77
  - Metrics ROI  78
- Defect Tracking  79
  - Why Should We Use a Defect Tracking System (DTS)?  80
  - Why Shouldn’t We Use a DTS?  82
  - Defect Tracking Tools  83
  - Keep Your Focus  85
- Test Planning  86
  - Test Strategy vs. Test Planning  86
  - Traceability  88
## Contents

Existing Processes and Models 88
  Audits 89
  Frameworks, Models, and Standards 90
Summary 93

### Part III  The Agile Testing Quadrants 95

#### Chapter 6  The Purpose of Testing 97
  The Agile Testing Quadrants 97
  Tests that Support the Team 98
  Tests that Critique the Product 101
  Knowing When a Story Is Done 104
  Shared Responsibility 105
  Managing Technical Debt 106
  Testing in Context 106
Summary 108

#### Chapter 7  Technology-Facing Tests that Support the Team 109
  An Agile Testing Foundation 109
  The Purpose of Quadrant 1 Tests 110
  Supporting Infrastructure 111
  Why Write and Execute These Tests? 112
  Lets Us Go Faster and Do More 112
  Making Testers’ Jobs Easier 114
  Designing with Testing in Mind 115
  Timely Feedback 118
  Where Do Technology-Facing Tests Stop? 119
  What If the Team Doesn’t Do These Tests? 121
  What Can Testers Do? 121
  What Can Managers Do? 122
  It’s a Team Problem 123
  Toolkit 123
  Source Code Control 123
  IDEs 124
  Build Tools 126
  Build Automation Tools 126
  Unit Test Tools 126
Summary 127
Chapter 8  Business-Facing Tests that Support the Team  129
  Driving Development with Business-Facing Tests 129
  The Requirements Quandary 132
    Common Language 134
    Eliciting Requirements 135
    Advance Clarity 140
    Conditions of Satisfaction 142
    Ripple Effects 143
  Thin Slices, Small Chunks 144
  How Do We Know We’re Done? 146
  Tests Mitigate Risk 147
  Testability and Automation 149
  Summary 150

Chapter 9  Toolkit for Business-Facing Tests that Support the Team  153
  Business-Facing Test Tool Strategy 153
  Tools to Elicit Examples and Requirements 155
    Checklists 156
    Mind Maps 156
    Spreadsheets 159
    Mock-Ups 160
    Flow Diagrams 160
    Software-Based Tools 163
  Tools for Automating Tests Based on Examples 164
    Tools to Test below the GUI and API Level 165
    Tools for Testing through the GUI 170
  Strategies for Writing Tests 177
    Build Tests Incrementally 178
    Keep the Tests Passing 179
    Use Appropriate Test Design Patterns 179
    Keyword and Data-Driven Tests 182
  Testability 183
    Code Design and Test Design 184
      Automated vs. Manual Quadrant 2 Tests 185
  Test Management 186
  Summary 186

Chapter 10  Business-Facing Tests that Critique the Product  189
  Introduction to Quadrant 3 190
  Demonstrations 191
Scenario Testing 192
Exploratory Testing 195
  Session-Based Testing 200
  Automation and Exploratory Testing 201
  An Exploratory Tester 201
Usability Testing 202
  User Needs and Persona Testing 202
  Navigation 204
  Check Out the Competition 204
Behind the GUI 204
  API Testing 205
  Web Services 207
Testing Documents and Documentation 207
  User Documentation 207
  Reports 208
Tools to Assist with Exploratory Testing 210
  Test Setup 211
  Test Data Generation 212
  Monitoring Tools 212
  Simulators 213
  Emulators 213
Summary 214

Chapter 11 Critiquing the Product Using Technology-Facing Tests 217
  Introduction to Quadrant 4 217
  Who Does It? 220
  When Do You Do It? 222
  “ility” Testing 223
    Security 223
    Maintainability 227
    Interoperability 228
    Compatibility 229
    Reliability 230
    Installability 231
    “ility” Summary 232
  Performance, Load, Stress, and Scalability Testing 233
    Scalability 233
    Performance and Load Testing 234
    Performance and Load-Testing Tools 234
    Baseline 235
Test Environments 237  
Memory Management 237  
Summary 238  

**Chapter 12  Summary of Testing Quadrants** 241  
Review of the Testing Quadrants 241  
A System Test Example 242  
The Application 242  
The Team and the Process 243  
Tests Driving Development 244  
Unit Tests 244  
Acceptance Tests 245  
Automation 245  
The Automated Functional Test Structure 245  
Web Services 247  
Embedded Testing 248  
Critiquing the Product with Business-Facing Tests 248  
Exploratory Testing 248  
Testing Data Feeds 249  
The End-to-End Tests 249  
User Acceptance Testing 250  
Reliability 250  
Documentation 251  
Documenting the Test Code 251  
Reporting the Test Results 251  
Using the Agile Testing Quadrants 252  
Summary 253  

**Part IV  Automation** 255  

**Chapter 13  Why We Want to Automate Tests and What Holds Us Back** 257  
Why Automate? 258  
Manual Testing Takes Too Long 258  
Manual Processes Are Error Prone 259  
Automation Frees People to Do Their Best Work 259  
Automated Regression Tests Provide a Safety Net 261  
Automated Tests Give Feedback, Early and Often 262  
Tests and Examples that Drive Coding Can Do More 262
<table>
<thead>
<tr>
<th>Tests Are Great Documentation</th>
<th>263</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROI and Payback</td>
<td>264</td>
</tr>
<tr>
<td>Barriers to Automation—Things that Get in the Way</td>
<td>264</td>
</tr>
<tr>
<td>Bret’s List</td>
<td>264</td>
</tr>
<tr>
<td>Our List</td>
<td>265</td>
</tr>
<tr>
<td>Programmers’ Attitude—“Why Automate?”</td>
<td>265</td>
</tr>
<tr>
<td>The “Hump of Pain” (The Learning Curve)</td>
<td>266</td>
</tr>
<tr>
<td>Initial Investment</td>
<td>267</td>
</tr>
<tr>
<td>Code that’s Always in Flux</td>
<td>269</td>
</tr>
<tr>
<td>Legacy Code</td>
<td>269</td>
</tr>
<tr>
<td>Fear</td>
<td>269</td>
</tr>
<tr>
<td>Old Habits</td>
<td>270</td>
</tr>
<tr>
<td>Can We Overcome These Barriers?</td>
<td>270</td>
</tr>
<tr>
<td>Summary</td>
<td>271</td>
</tr>
</tbody>
</table>

**Chapter 14  An Agile Test Automation Strategy**  273

<table>
<thead>
<tr>
<th>An Agile Approach to Test Automation</th>
<th>274</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automation Test Categories</td>
<td>274</td>
</tr>
<tr>
<td>Test Automation Pyramid</td>
<td>276</td>
</tr>
<tr>
<td>What Can We Automate?</td>
<td>279</td>
</tr>
<tr>
<td>Continuous Integration, Builds, and Deploys</td>
<td>280</td>
</tr>
<tr>
<td>Unit and Component Tests</td>
<td>282</td>
</tr>
<tr>
<td>API or Web Services Testing</td>
<td>282</td>
</tr>
<tr>
<td>Testing behind the GUI</td>
<td>282</td>
</tr>
<tr>
<td>Testing the GUI</td>
<td>282</td>
</tr>
<tr>
<td>Load Tests</td>
<td>283</td>
</tr>
<tr>
<td>Comparisons</td>
<td>283</td>
</tr>
<tr>
<td>Repetitive Tasks</td>
<td>284</td>
</tr>
<tr>
<td>Data Creation or Setup</td>
<td>284</td>
</tr>
<tr>
<td>What Shouldn’t We Automate?</td>
<td>285</td>
</tr>
<tr>
<td>Usability Testing</td>
<td>285</td>
</tr>
<tr>
<td>Exploratory Testing</td>
<td>286</td>
</tr>
<tr>
<td>Tests that Will Never Fail</td>
<td>286</td>
</tr>
<tr>
<td>One-Off Tests</td>
<td>286</td>
</tr>
<tr>
<td>What Might Be Hard to Automate?</td>
<td>287</td>
</tr>
<tr>
<td>Developing an Automation Strategy—Where Do We Start?</td>
<td>288</td>
</tr>
<tr>
<td>Where Does It Hurt the Most?</td>
<td>289</td>
</tr>
<tr>
<td>Multi-Layered Approach</td>
<td>290</td>
</tr>
<tr>
<td>Think about Test Design and Maintenance</td>
<td>292</td>
</tr>
<tr>
<td>Choosing the Right Tools</td>
<td>294</td>
</tr>
</tbody>
</table>
Applying Agile Principles to Test Automation
   Keep It Simple 298
   Iterative Feedback 299
   Whole-Team Approach 300
   Taking the Time to Do It Right 301
   Learn by Doing 303
   Apply Agile Coding Practices to Tests 303
Supplying Data for Tests 304
   Data Generation Tools 304
   Avoid Database Access 306
   When Database Access Is Unavoidable or Even Desirable 307
   Understand Your Needs 310
Evaluating Automation Tools 311
   Identifying Requirements for Your Automation Tool 311
   One Tool at a Time 312
   Choosing Tools 313
   Agile-Friendly Tools 316
Implementing Automation 316
Managing Automated Tests 319
   Organizing Tests 319
   Organizing Test Results 322
Go Get Started 324
Summary 324

Part V  An Iteration in the Life of a Tester 327

Chapter 15  Tester Activities in Release or Theme Planning 329
   The Purpose of Release Planning 330
   Sizing 332
      How to Size Stories 332
      The Tester’s Role in Sizing Stories 333
      An Example of Sizing Stories 334
   Prioritizing 338
      Why We Prioritize Stories 338
      Testing Considerations While Prioritizing 339
   What’s in Scope? 340
      Deadlines and Timelines 340
      Focus on Value 341
Writing Task Cards 389
Deciding on Workload 393
Testable Stories 393
Collaborate with Customers 396
High-Level Tests and Examples 397
Reviewing with Customers 400
Reviewing with Programmers 400
Test Cases as Documentation 402
Summary 403

Chapter 18 Coding and Testing 405
Driving Development 406
Start Simple 406
Add Complexity 407
Assess Risk 407
Coding and Testing Progress Together 409
Identify Variations 410
Power of Three 411
Focus on One Story 411
Tests that Critique the Product 412
Collaborate with Programmers 413
Pair Testing 413
“Show Me” 413
Talk to Customers 414
Show Customers 414
Understand the Business 415
Completing Testing Tasks 415
Dealing with Bugs 416
Is It a Defect or Is It a Feature? 417
Technical Debt 418
Zero Bug Tolerance 418
It’s All about Choices 419
Decide Which Bugs to Log 420
Choose When to Fix Your Bugs 421
Choose the Media You Should Use to Log a Bug 423
Alternatives and Suggestions for Dealing with Bugs 424
Start Simple 428
Facilitate Communication 429
Testers Facilitate Communication 429
Distributed Teams 431
<table>
<thead>
<tr>
<th>Page Number</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>475</td>
<td>Customer Expectations</td>
</tr>
<tr>
<td>475</td>
<td>Production Support</td>
</tr>
<tr>
<td>475</td>
<td>Understand Impact to Business</td>
</tr>
<tr>
<td>476</td>
<td>Summary</td>
</tr>
<tr>
<td>479</td>
<td>Part VI Summary</td>
</tr>
<tr>
<td>481</td>
<td>Chapter 21 Key Success Factors</td>
</tr>
<tr>
<td>482</td>
<td>Success Factor 1: Use the Whole-Team Approach</td>
</tr>
<tr>
<td>482</td>
<td>Success Factor 2: Adopt an Agile Testing Mind-Set</td>
</tr>
<tr>
<td>484</td>
<td>Success Factor 3: Automate Regression Testing</td>
</tr>
<tr>
<td>484</td>
<td>Success Factor 4: Provide and Obtain Feedback</td>
</tr>
<tr>
<td>486</td>
<td>Success Factor 5: Build a Foundation of Core Practices</td>
</tr>
<tr>
<td>486</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>487</td>
<td>Test Environments</td>
</tr>
<tr>
<td>487</td>
<td>Manage Technical Debt</td>
</tr>
<tr>
<td>488</td>
<td>Working Incrementally</td>
</tr>
<tr>
<td>488</td>
<td>Coding and Testing Are Part of One Process</td>
</tr>
<tr>
<td>489</td>
<td>Synergy between Practices</td>
</tr>
<tr>
<td>489</td>
<td>Success Factor 6: Collaborate with Customers</td>
</tr>
<tr>
<td>490</td>
<td>Success Factor 7: Look at the Big Picture</td>
</tr>
<tr>
<td>491</td>
<td>Summary</td>
</tr>
<tr>
<td>493</td>
<td>Glossary</td>
</tr>
<tr>
<td>501</td>
<td>Bibliography</td>
</tr>
<tr>
<td>509</td>
<td>Index</td>
</tr>
</tbody>
</table>
This page intentionally left blank
“Quality is baked in,” the programmers kept telling me. As part of a proposed acquisition, my boss had asked me to perform some final due diligence on the development team and its product. We’d already established that the company’s recently launched product was doing well in the market, but I was to make sure we were not about to buy more trouble than benefit. So I spent my time with the development team. I was looking for problems that might arise from having rushed the product into release. I wondered, “Was the code clean? Were there modules that could only be worked on by one developer? Were there hundreds or thousands of defects waiting to be discovered?” And when I asked about the team’s approach to testing, “Quality is baked in” was the answer I got.

Because this rather unusual colloquialism could have meant just about anything, I pressed further. What I found was that this was the company founder’s shorthand for expressing one of quality pioneer W. Edwards Deming’s famous fourteen points: Build quality into the product rather than trying to test it in later.

The idea of building quality into their products is at the heart of how agile teams work. Agile teams work in short iterations in part to ensure that the application remains at a known state of quality. Agile teams are highly cross-functional, with programmers, testers, and others working side by side throughout each iteration so that quality can be baked into products through techniques such as acceptance-test driven development, a heavy emphasis on automated testing, and whole-team thinking. Good agile teams bake quality in by building their products continuously, integrating new work within minutes of its being completed. Agile teams utilize techniques such as refactoring and a preference for simplicity in order to prevent technical debt from accumulating.
Learning how to do these things is difficult, and especially so for testers, whose role has been given scant attention in previous books. Fortunately, the book you now hold in your hands answers questions on the mind of every tester who’s beginning to work on an agile project, such as:

- What are my roles and responsibilities?
- How do I work more closely with programmers?
- How much do we automate, and how do we start automating?

The experience of Lisa and Janet shines through on every page of the book. However, this book is not just their story. Within this book, they incorporate dozens of stories from real-world agile testers. These stories form the heart of the book and are what makes it so unique. It’s one thing to shout from the ivory tower, “Here’s how to do agile testing.” It’s another to tell the stories of the teams that have struggled and then emerged agile and victorious over challenges such as usability testing, legacy code that resists automation, transitioning testers used to traditional phase-gate development, testing that “keeps up” with short iterations, and knowing when a feature is “done.”

Lisa and Janet were there at the beginning, learning how to do agile testing back when the prevailing wisdom was that agile teams didn’t need testers and that programmers could bake quality in by themselves. Over the years and through articles, conference presentations, and working with their clients and teams, Lisa and Janet have helped us see the rich role to be filled by testers on agile projects. In this book, Lisa and Janet use a test automation pyramid, the agile testing quadrants of Brian Marick (himself another world-class agile tester), and other techniques to show how much was missing from a mind-set that said testing is necessary but testers aren’t.

If you want to learn how to bake quality into your products or are an aspiring agile tester seeking to understand your role, I can think of no better guides than Lisa and Janet.
Imagine yourself skimming over a landscape thousands of years ago, looking at the people below. They’re barely scraping out a living in a hostile territory, doing some hunting, some fishing, and a little planting. Off in the distance, you see the glitter of a glacier. Moving closer, you see that it’s melting fast and that it’s barely damming a huge lake. As you watch, the lake breaks through, sweeping down a riverbed, carving it deeper, splashing up against cliffs on the far side of the landscape—some of which collapse.

As you watch, the dazed inhabitants begin to explore the opening. On the other side, there’s a lush landscape, teaming with bigger animals than they’ve ever seen before, some grazing on grass with huge seed heads, some squabbling over mounds of fallen fruit.

People move in. Almost immediately, they begin to live better. But as the years fly past, you see them adapt. They begin to use nets to fish in the fast-running streams. They learn the teamwork needed to bring down the larger animals, though not without a few deaths along the way. They find ever-better ways to cultivate this new grass they’ve come to call “wheat.”

As you watch, the mad burst of innovation gives way to a stable solution, a good way to live in this new land, a way that’s taught to each new generation. Although just over there, you spy someone inventing the wheel . . .

In the early years of this century, the adoption of Agile methods sometimes seemed like a vast dam breaking, opening up a way to a better—more productive, more joyful—way of developing software. Many early adopters saw benefits right away, even though they barely knew what they were doing.
Some had an easier time of it than others. Programmers were like the hunters in the fable above. Yes, they had to learn new skills in order to hunt bison, but they knew how to hunt rabbits, more or less, and there were plenty of rabbits around. Testers were more like spear-fishers in a land where spear-fishing wouldn’t work. Going from spear-fishing to net-fishing is a much bigger conceptual jump than going from rabbit to bison. And, while some of the skills—cleaning fish, for example—were the same in the new land, the testers had to invent new skills of net-weaving before they could truly pull their weight.

So testing lagged behind. Fortunately, we had early adopters like Lisa and Janet, people who dove right in alongside the programmers, testers who were not jealous of their role or their independence, downright pleasant people who could figure out the biggest change of all in Agile testing: the tester’s new social role.

As a result, we have this book. It’s the stable solution, the good way for testers to live in this new Agile land of ours. It’s not the final word—we could use the wheel, and I myself am eager for someone to invent antibiotics—but what’s taught here will serve you well until someone, perhaps Lisa and Janet, brings the next big change.
We were early adopters of Extreme Programming (XP), testing on XP teams that weren’t at all sure where testers or their brand of testing fit in. At the time, there wasn’t much in the agile (which wasn’t called agile yet) literature about acceptance testing, or how professional testers might contribute. We learned not only from our own experiences but from others in the small agile community. In 2002, Lisa co-wrote *Testing Extreme Programming* with Tip House, with lots of help from Janet. Since then, agile development has evolved, and the agile testing community has flourished. With so many people contributing ideas, we’ve learned a whole lot more about agile testing.

Individually and together, we’ve helped teams transition to agile, helped testers learn how to contribute on agile teams, and worked with others in the agile community to explore ways that agile teams can be more successful at testing. Our experiences differ. Lisa has spent most of her time as an agile tester on stable teams working for years at a time on web applications in the retail, telephony, and financial industries. Janet has worked with software organizations developing enterprise systems in a variety of industries. These agile projects have included developing a message-handling system, an environmental-tracking system, a remote data management system (including an embedded application, with a communication network as well as the application), an oil and gas production accounting application, and applications in the airline transportation industry. She has played different roles—sometimes tester, sometimes coach—but has always worked to better integrate the testers with the rest of the team. She has been with teams from as little as six months to as long as one-and-a-half years.

With these different points of view, we have learned to work together and complement each other’s skill sets, and we have given many presentations and tutorials together.
WHY WE WROTE THIS BOOK

Several excellent books oriented toward agile development on testing and test patterns have been published (see our bibliography). These books are generally focused on helping the developer. We decided to write a book aimed at helping agile teams be more successful at delivering business value using tests that the business can understand. We want to help testers and quality assurance (QA) professionals who have worked in more traditional development methodologies make the transition to agile development.

We’ve figured out how to apply—on a practical, day-to-day level—the fruits of our own experience working with teams of all sizes and a variety of ideas from other agile practitioners. We’ve put all this together in this book to help testers, quality assurance managers, developers, development managers, product owners, and anyone else with a stake in effective testing on agile projects to deliver the software their customers need. However, we’ve focused on the role of the tester, a role that may be adopted by a variety of professionals.

Agile testing practices aren’t limited to members of agile teams. They can be used to improve testing on projects using traditional development methodologies as well. This book is also intended to help testers working on projects using any type of development methodology.

Agile development isn’t the only way to successfully deliver software. However, all of the successful teams we’ve been on, agile or waterfall, have had several critical commonalities. The programmers write and automate unit and integration tests that provide good code coverage. They are disciplined in the use of source code control and code integration. Skilled testers are involved from the start of the development cycle and are given time and resources to do an adequate job of all necessary forms of testing. An automated regression suite that covers the system functionality at a higher level is run and checked regularly. The development team understands the customers’ jobs and their needs, and works closely together with the business experts.

People, not methodologies or tools, make projects successful. We enjoy agile development because its values, principles, and core practices enable people to do their best work, and testing and quality are central to agile development. In this book, we explain how to apply agile values and principles to your unique testing situation and enable your teams to succeed. We have more about that in Chapter 1, “What Is Agile Testing, Anyway?” and in Chapter 2, “Ten Principles for Agile Testers.”
**HOW WE WROTE THIS BOOK**

Having experienced the benefits of agile development, we used agile practices to produce this book. As we began work on the book, we talked to agile testers and teams from around the globe to find out what problems they encountered and how they addressed them. We planned how we would cover these areas in the book.

We made a release plan based on two-week iterations. Every two weeks, we delivered two rough-draft chapters to our book website. Because we aren’t co-located, we found tools to use to communicate, provide “source code control” for our chapters, deliver the product to our customers, and get their feedback. We couldn’t “pair” much real-time, but we traded chapters back and forth for review and revision, and had informal “stand-ups” daily via instant message.

Our “customers” were the generous people in the agile community who volunteered to review draft chapters. They provided feedback by email or (if we were lucky) in person. We used the feedback to guide us as we continued writing and revising. After all the rough drafts were done, we made a new plan to complete the revisions, incorporating all the helpful ideas from our “customers.”

Our most important tool was mind maps. We started out by creating a mind map of how we envisioned the whole book. We then created mind maps for each section of the book. Before writing each chapter, we brainstormed with a mind map. As we revised, we revisited the mind maps, which helped us think of ideas we may have missed.

Because we think the mind maps added so much value, we’ve included the mind map as part of the opening of each chapter. We hope they’ll help you get an overview of all the information included in the chapter, and inspire you to try using mind maps yourself.

**OUR AUDIENCE**

This book will help you if you’ve ever asked any of the following excellent questions, which we’ve heard many times:

- If developers are writing tests, what do the testers do?
- I’m a QA manager, and our company is implementing agile development (Scrum, XP, DSDM, name your flavor). What’s my role now?
I’ve worked as a tester on a traditional waterfall team, and I’m really excited by what I’ve read about agile. What do I need to know to work on an agile team?

What’s an “agile tester”?

I’m a developer on an agile team. We’re writing code test-first, but our customers still aren’t happy with what we deliver. What are we missing?

I’m a developer on an agile team. We’re writing our code test-first. We make sure we have tests for all our code. Why do we need testers?

I coach an agile development team. Our QA team can’t keep up with us, and testing always lags behind. Should we just plan to test an iteration behind development?

I’m a software development manager. We recently transitioned to agile, but all our testers quit. Why?

I’m a tester on a team that’s going agile. I don’t have any programming or automation skills. Is there any place for me on an agile team?

How can testing possibly keep up with two-week iterations?

What about load testing, performance testing, usability testing, all the other “ilities”? Where do these fit in?

We have audit requirements. How does agile development and testing address these?

If you have similar questions and you’re looking for practical advice about how testers contribute to agile teams and how agile teams can do an effective job of testing, you’ve picked up the right book.

There are many “flavors” of agile development, but they all have much in common. We support the Agile Manifesto, which we explain in Chapter 1, “What Is Agile Testing, Anyway?” Whether you’re practicing Scrum, Extreme Programming, Crystal, DSDM, or your own variation of agile development, you’ll find information here to help with your testing efforts.

A User Story for an Agile Testing Book

When Robin Dymond, a managing consultant and trainer who has helped many teams adopt lean and agile, heard we were writing this book, he sent us the user story he’d like to have fulfilled. It encapsulates many of the requirements we planned to deliver.
Acceptance conditions:

- My concerns and fears about losing control of testing are addressed.
- My concerns and fears about having to write code (never done it) are addressed.
- As a tester I understand my new value to the team.
- As a tester new to Agile, I can easily read about things that are most important to my new role.
- As a tester new to Agile, I can easily ignore things that are less important to my new role.
- As a tester new to Agile, I can easily get further detail about agile testing that is important to MY context.

Were I to suggest a solution to this problem, I think of Scrum versus XP. With Scrum you get a simple view that enables people to quickly adopt Agile. However, Scrum is the tip of the iceberg for successful agile teams. For testers who are new, I would love to see agile testing ideas expressed in layers of detail. What do I need to know today, what should I know tomorrow, and what context-sensitive things should I consider for continuous improvement?

We’ve tried to provide these layers of detail in this book. We’ll approach agile testing from a few different perspectives: transitioning into agile development, using an agile testing matrix to guide testing efforts, and explaining all the different testing activities that take place throughout the agile development cycle.
HOW TO USE THIS BOOK

If you aren’t sure where to start in this book, or you just want a quick overview, we suggest you read the last chapter, Chapter 21, “Key Success Factors,” and follow wherever it leads you.

Part I: Introduction

If you want quick answers to questions such as “Is agile testing different than testing on waterfall projects?” or “What’s the difference between a tester on a traditional team and an agile tester?,” start with Part I, which includes the following chapters:

- Chapter 1: What Is Agile Testing, Anyway?
- Chapter 2: Ten Principles for Agile Testers

These chapters are the “tip of the iceberg” that Robin requested in his user story. They include an overview of how agile differs from a traditional phased approach and explore the “whole team” approach to quality and testing.

In this part of the book we define the “agile testing mind-set” and what makes testers successful on agile teams. We explain how testers apply agile values and principles to contribute their particular expertise.

Part II: Organizational Challenges

If you’re a tester or manager on a traditional QA team, or you’re coaching a team that’s moving to agile, Part II will help you with the organizational challenges faced by teams in transition. The “whole team” attitude represents a lot of cultural changes to team members, but it helps overcome the fear testers have when they wonder how much control they’ll have or whether they’ll be expected to write code.

Some of the questions answered in Part II are:

- How can we engage the QA team?
- What about management’s expectations?
- How should we structure our agile team, and where do the testers fit?
- What do we look for when hiring an agile tester?
- How do we cope with a team distributed across the globe?
Part II also introduces some topics we don’t always enjoy talking about. We explore ideas about how to transition processes and models, such as audits or SOX compliance, that are common in traditional environments.

Metrics and how they’re applied can be a controversial issue, but there are positive ways to use them to benefit the team. Defect tracking easily becomes a point of contention for teams, with questions such as “Do we use a defect-tracking system?” or “When do we log bugs?”

Two common questions about agile testing from people with traditional test team experience are “What about test plans?” and “Is it true there’s no documentation on agile projects?” Part II clears up these mysteries.

The chapters in Part II are as follows:

- Chapter 3: Cultural Challenges
- Chapter 4: Team Logistics
- Chapter 5: Transitioning Typical Processes

**Part III: The Agile Testing Quadrants**

Do you want more details on what types of testing are done on agile projects? Are you wondering who does what testing? How do you know whether you’ve done all the testing that’s needed? How do you decide what practices, techniques, and tools fit your particular situation? If these are your concerns, check out Part III.

We use Brian Marick’s Agile Testing Quadrants to explain the purpose of testing. The quadrants help you define all the different areas your testing should address, from unit level tests to reliability and other “ilities,” and everything in between. This is where we get down into the nitty-gritty of how to deliver a high-quality product. We explain techniques that can help you to communicate well with your customers and better understand their requirements. This part of the book shows how tests drive development at multiple levels. It also provides tools for your toolkit that can help you to effectively define, design, and execute tests that support the team and critique the product. The chapters include the following:

- Chapter 6: The Purpose of Testing
- Chapter 7: Technology-Facing Tests that Support the Team
Chapter 8: Business-Facing Tests that Support the Team
Chapter 9: Toolkit for Business-Facing Tests that Support the Team
Chapter 10: Business-Facing Tests that Critique the Product
Chapter 11: Critiquing the Product Using Technology-Facing Tests
Chapter 12: Summary of Testing Quadrants

Part IV: Automation
Test automation is a central focus of successful agile teams, and it’s a scary topic for lots of people (we know, because it’s had us running scared before!). How do you squeeze test automation into short iterations and still get all the stories completed?

Part IV gets into the details of when and why to automate, how to overcome barriers to test automation, and how to develop and implement a test automation strategy that works for your team. Because test automation tools change and evolve so rapidly, our aim is not to explain how to use specific tools, but to help you select and use the right tools for your situation. Our agile test automation tips will help you with difficult challenges such as testing legacy code.

The chapters are as follows:

- Chapter 13: Why We Want to Automate Tests and What Holds Us Back
- Chapter 14: An Agile Test Automation Strategy

Part V: An Iteration in the Life of a Tester
If you just want to get a feel for what testers do throughout the agile development cycle, or you need help putting together all the information in this book, go to Part V. Here we chronicle an iteration, and more, in the life of an agile tester. Testers contribute enormous value throughout the agile software development cycles. In Part V, we explain the activities that testers do on a daily basis. We start with planning releases and iterations to get each iteration off to a good start, and move through the iteration—collaborating with the customer and development teams, testing, and writing code. We end the iteration by delivering new features and finding ways for the team to improve the process.

The chapters break down this way:

- Chapter 15: Tester Activities in Release or Theme Planning
- Chapter 16: Hit the Ground Running
Part VI: Summary

In Chapter 21, “Key Success Factors,” we present seven key factors agile teams can use for successful testing. If you’re having trouble deciding where to start with agile testing, or how to work on improving what you’re doing now, these success factors will give you some direction.

Other Elements

We’ve also included a glossary we hope you will find useful, as well as references to books, articles, websites, and blogs in the bibliography.

JUST START DOING IT—TODAY!

Agile development is all about doing your best work. Every team has unique challenges. We’ve tried to present all the information that we think may help agile testers, their teams, managers, and customers. Apply the techniques that you think are appropriate for your situation. Experiment constantly, evaluate the results, and come back to this book to see what might help you improve. Our goal is to help testers and agile teams enjoy delivering the best and most valuable product they can.

When we asked Dierk König, founder and project manager of Canoo Web-Test, what he thought was the number one success factor for agile testing, he answered: “Start doing it—today!” You can take a baby step to improve your team’s testing right now. Go get started!
This page intentionally left blank
So many people have helped us with this book that it’s hard to know whom to thank first. Chris Guzikowski gave us the opportunity to write this book and kept encouraging us along the way. When we were deciding whether to take on such a mammoth task, Mike Cohn gave us the sage advice that the best reason to write a book is that you have something to say. We sure have lots to say about agile testing. Fortunately, so do lots of other people who were willing to lend us a hand.

Many thanks to Brian Marick and Mike Cohn for writing such kind forewords. We’re honored that Mike selected our book for his signature series. We’re grateful for the many ideas and observations of his that are included in this book.

Brian Marick’s “Agile Testing Matrix” has guided both of us in our agile projects for several years, and it provides the core of Part III. Thank you, Brian, for thinking up the quadrants (and so many other contributions to agile testing) and letting us use them here.

We made constant use of the agile value of feedback. Many thanks to our official reviewers: Jennitta Andrea, Gerard Meszaros, Ron Jeffries, and Paul Duvall. Each one had unique and insightful comments that helped us greatly improve the book. Gerard also helped us be more consistent and correct in our testing terminology, and contributed some agile testing success stories.

Special thanks to two reviewers and top-notch agile testers who read every word we wrote and spent hours discussing the draft chapters with us in person: Pierre Veragen and Paul Rogers. Many of the good ideas in this book are theirs.
We interviewed several teams to learn what advice they would give new agile teams and testers, and solicited success stories and “lessons learned” from colleagues in the agile testing community. Heartfelt thanks to our many contributors of sidebars and quotes, as well as providers of helpful feedback, including (in no particular order) Robin Dymond, Bret Pettichord, Tae Chang, Bob Galen, Erika Boyer, Grig Gheorghiu, Erik Bos, Mark Benander, Jonathan Rasmusson, Andy Pols, Dierk König, Rafael Santos, Jason Holzer, Christophe Louvion, David Reed, John Voris, Chris McMahon, Declan Whelan, Michael Bolton, Elisabeth Hendrickson, Joe Yakich, Andrew Glover, Alessandro Collino, Coni Tartaglia, Markus Gärtner, Megan Sumrell, Nathan Silberman, Mike Thomas, Mike Busse, Steve Perkins, Joseph King, Jakub Oleszkiewicz, Pierre Veragen (again), Paul Rogers (again), Jon Hagar, Antony Marcano, Patrick Wilson-Welsh, Patrick Fleisch, Apurva Chandra, Ken De Souza, and Carol Vaage.

Many thanks also to the rest of our community of unofficial reviewers who read chapters, gave feedback and ideas, and let us bounce ideas off of them, including Tom Poppendieck, Jun Bueno, Kevin Lawrence, Hannu Kokko, Titus Brown, Wim van de Goor, Lucas Campos, Kay Johansen, Adrian Howard, Henrik Kniberg, Shelly Park, Robert Small, Senaka Suriyaachchi, and Erik Petersen. And if we’ve neglected to list you here, it’s not that we value your contribution any less, it’s just that we didn’t keep good enough notes! We hope you will see how your time and effort paid off in the finished book.

We appreciate the groundwork laid by the agile pioneers who have helped us and our teams succeed with agile. You’ll find some of their works in the bibliography. We are grateful for the agile teams that have given us so many open source test tools that help all of our teams deliver so much value. Some of those tools are also listed in the bibliography.

Thanks to Mike Thomas for taking many of the action photos of an agile team that appear in this book. We hope these photos show those of you new to agile testing and development that there’s no big mystery—it’s just good people getting together to discuss, demo, and draw pictures.

Thanks so much to our Addison-Wesley editorial and production team who patiently answered many questions and turned this into the professional-looking book you see here, including Raina Chrobak, Chris Zahn, John Fuller, Sally Gregg, Bonnie Granat, Diane Freed, Jack Lewis, and Kim Arney.
I'm eternally grateful to Janet for agreeing to write this book with me. She kept us organized and on track so we could juggle book writing with our full-time jobs and personal lives. I'm fortunate to have a writing partner whose experience is complementary to mine. Like any successful agile project, this is a true team effort. This has been hard work, but thanks to Janet it has been a lot of fun, too.

I'd like to thank the members of my current team at ePlan Services Inc. (formerly known as Fast401k), which I joined (thanks to Mike Cohn, our first leader) in 2003. All of us learned so much working for Mike that first year, and it's a testament to his leadership that we continue to improve and help the business grow. Thanks to my awesome teammates who have each helped me become a better tester and agile team member, and who were all good sports while Mike Thomas took action photos of us: Nanda Lankapalli, Tony Sweets, Jeff Thuss, Lisa Owens, Mike Thomas, Vince Palumbo, Mike Busse, Nehru Kaja, Trevor Sterritt, Steve Kives, and former but still beloved team members Joe Yakich, Jason Kay, Jennifer Riefenberg, Matt Tierney, and Charles LeRose. I also have been lucky enough to work with the best customer team anywhere. They are too numerous to mention here, but many thanks to them, and in particular to Steve Perkins, Anne Olguin, and Zachary Shannon, who help us focus on delivering value. Thanks also to Mark and Dan Gutrich, founders and leaders of ePlan Services, for giving us all the opportunity to succeed with agile development.

Thanks to Kay and Zhon Johansen for teaching me about mind maps at Agile 2006. I hope we have put this skill to good use in creating this book.

Much gratitude to all my friends and family, whom I neglected terribly during the many months spent writing this book, and who nevertheless supported me constantly. There are too many to mention, but I must specially thank Anna Blake for her constant understanding and provision of donkey therapy. Chester and Ernest, the donkeys of my heart, have kept pulling me along. Dodger didn't make the whole book-writing journey in this world, but his memory continues to lift me up. My little poodle and muse Tango was by my side every minute that I worked on this book at home, joined occasionally by Bruno, Bubba, Olive, Squiggy, Starsky, Bobcat, and Patty. Thanks to my parents for being proud of me and not complaining about my neglect of them during this book-writing time.

I know that my husband, Bob Downing, took a deep breath when I exclaimed, “I have the chance to write another book about agile testing,” but he nevertheless encouraged me constantly and made it possible for me to find the time to write. He kept the “no-kill shelter” running, kept our lives rolling, kept my spirits up, and sustained me with many fabulous meals. He is the light of my life.

—Lisa
Lisa and I made a great team; each of us had our own strengths. When one of us faltered and took some time to recoup, the other picked up the slack. I learned so much from Lisa (thanks go to her for giving me this opportunity), but I also found I learned a lot from myself. Just the process of articulating my thoughts helped to clarify things that had been rumbling around in my brain for a long time. The mindmaps helped immensely, so thanks to Kenji Hiranabe, who gave a workshop at Agile 2007 and made me realize what a powerful yet simple tool mind maps can be.

This book-writing journey was an amazing experience. Thanks to the people on all the teams I worked with who provided so many of the examples in this book.

It’s been a pretty special year all the way around. During the year (or so) it took to write this book, my family increased in size. My two daughters, Dana and Susan, each gave me a grandson—those were some of the times Lisa picked up the slack. I would like to thank my granddaughter Lauren (currently three) for making me leave my computer and play. It kept me sane. Thanks to my sister Colleen who gave me long-distance encouragement many mornings using instant messenger when I was feeling overwhelmed with the sheer number of hours I was putting in.

And a very special thanks to Jack, my husband, who moved his office downstairs when I took over the existing one. There were times when I am sure he felt neglected and wondered if he even had a wife as he spend many long hours alone. However, he was there with me the whole way, encouraging and supporting me in this endeavor.

—Janet
**Lisa Crispin** is an agile testing practitioner and coach. She specializes in showing testers and agile teams how testers can add value and how to guide development with business-facing tests. Her mission is to bring agile joy to the software testing world and testing joy to the agile development world. Lisa joined her first agile team in 2000, having enjoyed many years working as a programmer, analyst, tester, and QA director. Since 2003, she’s been a tester on a Scrum/XP team at ePlan Services, Inc. She frequently leads tutorials and workshops on agile testing at conferences in North America and Europe. Lisa regularly contributes articles about agile testing to publications such as *Better Software* magazine, *IEEE Software*, and *Methods and Tools*. Lisa also co-authored *Testing Extreme Programming* (Addison-Wesley, 2002) with Tip House.

For more about Lisa’s work, visit her websites, www.lisacrispin.com and www.agiletester.ca, or email her at lisa@agiletester.ca.

**Janet Gregory** is the founder of DragonFire Inc., an agile quality process consultancy and training firm. Her passion is helping teams build quality systems. For the past ten years, she has worked as a coach and tester, introducing agile practices into both large and small companies. Her focus is working with business users and testers to understand their role in agile projects. Janet’s programming background is a definite plus when she partners with developers on her agile teams to implement innovative agile test automation solutions. Janet is a frequent speaker at agile and testing software conferences, and she is a major contributor to the North American agile testing community.

For more about Janet’s work, visit her websites at www.janetgregory.ca, www.janetgregory.blogspot.com, and www.agiletester.ca, or you can email her at janet@agiletester.ca.
This page intentionally left blank
Like a lot of terminology, “agile development” and “agile testing” mean different things to different people. In this chapter, we explain our view of agile, which reflects the Agile Manifesto and general principles and values shared by different agile methods. We want to share a common language with you, the reader, so we’ll go over some of our vocabulary. We compare and contrast agile development and testing with the more traditional phased approach. The “whole team” approach promoted by agile development is central to our attitude toward quality and testing, so we also talk about that here.

**Agile Values**

“Agile” is a buzzword that will probably fall out of use someday and make this book seem obsolete. It’s loaded with different meanings that apply in different circumstances. One way to define “agile development” is to look at the Agile Manifesto (see Figure 1-1).

Using the values from the Manifesto to guide us, we strive to deliver small chunks of business value in extremely short release cycles.
We use the word “agile” in this book in a broad sense. Whether your team is practicing a particular agile method, such as Scrum, XP, Crystal, DSDM, or FDD, to name a few, or just adopting whatever principles and practices make sense for your situation, you should be able to apply the ideas in this book. If you’re delivering value to the business in a timely manner with high-quality software, and your team continually strives to improve, you’ll find useful information here. At the same time, there are particular agile practices we feel are crucial to any team’s success. We’ll talk about these throughout the book.

**WHAT DO WE MEAN BY “AGILE TESTING”?**

You might have noticed that we use the term “tester” to describe a person whose main activities revolve around testing and quality assurance. You’ll also see that we often use the word “programmer” to describe a person whose main activities revolve around writing production code. We don’t intend that these terms sound narrow or insignificant. Programmers do more than turn a specification into a program. We don’t call them “developers,” because ev-
WHAT DO WE MEAN BY “AGILE TESTING”?

Everyone involved in delivering software is a developer. Testers do more than perform “testing tasks.” Each agile team member is focused on delivering a high-quality product that provides business value. Agile testers work to ensure that their team delivers the quality their customers need. We use the terms “programmer” and “tester” for convenience.

Several core practices used by agile teams relate to testing. Agile programmers use test-driven development (TDD), also called test-driven design, to write quality production code. With TDD, the programmer writes a test for a tiny bit of functionality, sees it fail, writes the code that makes it pass, and then moves on to the next tiny bit of functionality. Programmers also write code integration tests to make sure the small units of code work together as intended. This essential practice has been adopted by many teams, even those that don’t call themselves “agile,” because it’s just a smart way to think through your software design and prevent defects. Figure 1-2 shows a sample unit test result that a programmer might see.

This book isn’t about unit-level or component-level testing, but these types of tests are critical to a successful project. Brian Marick [2003] describes these types of tests as “supporting the team,” helping the programmers know what code to write next. Brian also coined the term “technology-facing tests,” tests that fall into the programmer’s domain and are described using programmer terms and jargon. In Part II, we introduce the Agile Testing Quadrants and examine the different categories of agile testing. If you want to learn more about writing unit and component tests, and TDD, the bibliography will steer you to some good resources.

Figure 1-2  Sample unit test output
If you want to know how agile values, principles, and practices applied to testing can help you, as a tester, do your best work, and help your team deliver more business value, please keep reading. If you’ve bothered to pick up this book, you’re probably the kind of professional who continually strives to grow and learn. You’re likely to have the mind-set that a good agile team needs to succeed. This book will show you ways to improve your organization’s product, provide the most value possible to your team, and enjoy your job.

During a break from working on this chapter, I talked to a friend who works in quality assurance for a large company. It was a busy time of year, and management expected everyone to work extra hours. He said, “If I thought working 100 extra hours would solve our problems, I’d work ‘til 7 every night until that was done. But the truth was, it might take 4,000 extra hours to solve our problems, so working extra feels pointless.” Does this sound familiar?

—Lisa

If you’ve worked in the software industry long, you’ve probably had the opportunity to feel like Lisa’s friend. Working harder and longer doesn’t help when your task is impossible to achieve. Agile development acknowledges the reality that we only have so many good productive hours in a day or week, and that we can’t plan away the inevitability of change.

Agile development encourages us to solve our problems as a team. Business people, programmers, testers, analysts—everyone involved in software development—decides together how best to improve their product. Best of all, as testers, we’re working together with a team of people who all feel responsible for delivering the best possible quality, and who are all focused on testing. We love doing this work, and you will too.

When we say “agile testing” in this book, we’re usually talking about business-facing tests, tests that define the business experts’ desired features and functionality. We consider “customer-facing” a synonym for “business-facing.” “Testing” in this book also includes tests that critique the product and focus on discovering what might be lacking in the finished product so that we can improve it. It includes just about everything beyond unit and component level testing: functional, system, load, performance, security, stress, usability, exploratory, end-to-end, and user acceptance. All these types of tests might be appropriate to any given project, whether it’s an agile project or one using more traditional methodologies.
Agile testing doesn’t just mean testing on an agile project. Some testing approaches, such as exploratory testing, are inherently agile, whether it’s done an agile project or not. Testing an application with a plan to learn about it as you go, and letting that information guide your testing, is in line with valuing working software and responding to change. Later chapters discuss agile forms of testing as well as “agile testing” practices.

**A LITTLE CONTEXT FOR ROLES AND ACTIVITIES ON AN AGILE TEAM**

We’ll talk a lot in this book about the “customer team” and the “developer team.” The difference between them is the skills they bring to delivering a product.

**Customer Team**

The customer team includes business experts, product owners, domain experts, product managers, business analysts, subject matter experts—everyone on the “business” side of a project. The customer team writes the stories or feature sets that the developer team delivers. They provide the examples that will drive coding in the form of business-facing tests. They communicate and collaborate with the developer team throughout each iteration, answering questions, drawing examples on the whiteboard, and reviewing finished stories or parts of stories.

Testers are integral members of the customer team, helping elicit requirements and examples and helping the customers express their requirements as tests.

**Developer Team**

Everyone involved with delivering code is a developer, and is part of the developer team. Agile principles encourage team members to take on multiple activities; any team member can take on any type of task. Many agile practitioners discourage specialized roles on teams and encourage all team members to transfer their skills to others as much as possible. Nevertheless, each team needs to decide what expertise their projects require. Programmers, system administrators, architects, database administrators, technical writers, security specialists, and people who wear more than one of these hats might be part of the team, physically or virtually.
Testers are also on the developer team, because testing is a central component of agile software development. Testers advocate for quality on behalf of the customer and assist the development team in delivering the maximum business value.

**Interaction between Customer and Developer Teams**

The customer and developer teams work closely together at all times. Ideally, they’re just one team with a common goal. That goal is to deliver value to the organization. Agile projects progress in iterations, which are small development cycles that typically last from one to four weeks. The customer team, with input from the developers, will prioritize stories to be developed, and the developer team will determine how much work they can take on. They’ll work together to define requirements with tests and examples, and write the code that makes the tests pass. Testers have a foot in each world, understanding the customer viewpoint as well as the complexities of the technical implementation (see Figure 1-3).

Some agile teams don’t have any members who define themselves as “testers.” However, they all need someone to help the customer team write business-facing tests for the iteration’s stories, make sure the tests pass, and make sure that adequate regression tests are automated. Even if a team does have testers, the entire agile team is responsible for these testing tasks. Our experience with agile teams has shown that testing skills and experience are vital to project success and that testers do add value to agile teams.

![Interaction of Roles](image-url)
HOW IS AGILE TESTING DIFFERENT?

We both started working on agile teams at the turn of the millennium. Like a lot of testers who are new to agile, we didn’t know what to expect at first. Together with our respective agile teams, we’ve worked on we’ve learned a lot about testing on agile projects. We’ve also implemented ideas and practices suggested by other agile testers and teams. Over the years, we’ve shared our experiences with other agile testers as well. We’ve facilitated workshops and led tutorials at agile and testing conferences, talked with local user groups, and joined countless discussions on agile testing mailing lists. Through these experiences, we’ve identified differences between testing on agile teams and testing on traditional waterfall development projects. Agile development has transformed the testing profession in many ways.

Working on Traditional Teams

Neither working closely with programmers nor getting involved with a project from the earliest phases was new to us. However, we were used to strictly enforced gated phases of a narrowly defined software development life cycle, starting with release planning and requirements definition and usually ending with a rushed testing phase and a delayed release. In fact, we often were thrust into a gatekeeper role, telling business managers, “Sorry, the requirements are frozen; we can add that feature in the next release.”

As leaders of quality assurance teams, we were also often expected to act as gatekeepers of quality. We couldn’t control how the code was written, or even if any programmers tested their code, other than by our personal efforts at collaboration. Our post-development testing phases were expected to boost quality after code was complete. We had the illusion of control. We usually had the keys to production, and sometimes we had the power to postpone releases or stop them from going forward. Lisa even had the title of “Quality Boss,” when in fact she was merely the manager of the QA team.

Our development cycles were generally long. Projects at a company that produced database software might last for a year. The six-month release cycles Lisa experienced at an Internet start-up seemed short at the time, although it was still a long time to have frozen requirements. In spite of much process and discipline, diligently completing one phase before moving on to the next, it was plenty of time for the competition to come out ahead, and the applications were not always what the customers expected.
Traditional teams are focused on making sure all the specified requirements are delivered in the final product. If everything isn’t ready by the original target release date, the release is usually postponed. The development teams don’t usually have input about what features are in the release, or how they should work. Individual programmers tend to specialize in a particular area of the code. Testers study the requirements documents to write their test plans, and then they wait for work to be delivered to them for testing.

Working on Agile Teams

Transitioning to the short iterations of an agile project might produce initial shock and awe. How can we possibly define requirements and then test and deliver production-ready code in one, two, three, or four weeks? This is particularly tough for larger organizations with separate teams for different functions and even harder for teams that are geographically dispersed. Where do all these various programmers, testers, analysts, project managers, and countless specialties fit in a new agile project? How can we possibly code and test so quickly? Where would we find time for difficult efforts such as automating tests? What control do we have over bad code getting delivered to production?

We’ll share our stories from our first agile experiences to show you that everyone has to start somewhere.

Lisa’s Story

My first agile team embraced Extreme Programming (XP), not without some “learning experiences.” Serving as the only professional tester on a team of eight programmers who hadn’t learned how to automate unit tests was disheartening. The first two-week iteration felt like jumping off a cliff.

Fortunately, we had a good coach, excellent training, a supportive community of agile practitioners with ideas to share, and time to learn. Together we figured out some ins and outs of how to integrate testing into an agile project—indeed, how to drive the project with tests. I learned how I could use my testing skills and experience to add real value to an agile team.

The toughest thing for me (the former Quality Boss) to learn was that the customers, not I, decided on quality criteria for the product. I was horrified after the first iteration to find that the code crashed easily when two users logged in concurrently. My coach patiently explained, over my strident objections, that our customer, a start-up company, wanted to be able to show features to potential customers. Reliability and robustness were not yet the issue.

I learned that my job was to help the customers tell us what was valuable to them during each iteration, and to write tests to ensure that’s what they got.

—Lisa
My first foray into the agile world was also an Extreme Programming (XP) engagement. I had just come from an organization that practiced waterfall with some extremely bad practices, including giving the test team a day or so to test six months of code. In my next job as QA manager, the development manager and I were both learning what XP really meant. We successfully created a team that worked well together and managed to automate most of the tests for the functionality. When the organization downsized during the dot-com bust, I found myself in a new position at another organization as the lone tester with about ten developers on an XP project.

On my first day of the project, Jonathan Rasmusson, one of the developers, came up to me and asked me why I was there. The team was practicing XP, and the programmers were practicing test-first and automating all their own tests. Participating in that was a challenge I couldn't resist. The team didn't know what value I could add, but I knew I had unique abilities that could help the team. That experience changed my life forever, because I gained an understanding of the nuances of an agile project and determined then that my life's work was to make the tester role a more fulfilling one.

—Janet

Read Jonathan's Story
Jonathan Rasmusson, now an Agile Coach at Rasmusson Software Consulting, but Janet's coworker on her second agile team, explains how he learned how agile testers add value.

So there I was, a young hotshot J2EE developer excited and pumped to be developing software the way it should be developed—using XP. Until one day, in walks a new team member—a tester. It seems management thought it would be good to have a QA resource on the team.

That's fine. Then it occurred to me that this poor tester would have nothing to do. I mean, as a developer on an XP project, I was writing the tests. There was no role for QA here as far as I could see.

So of course I went up and introduced myself and asked quite pointedly what she was going to do on the project, because the developers were writing all the tests. While I can't remember exactly how Janet responded, the next six months made it very clear what testers can do on agile projects.

With the automation of the tedious, low-level boundary condition test cases, Janet as a tester was now free to focus on much greater value-add areas like exploratory testing, usability, and testing the app in ways developers hadn't originally anticipated. She worked with the
Agile teams work closely with the business and have a detailed understanding of the requirements. They’re focused on the value they can deliver, and they might have a great deal of input into prioritizing features. Testers don’t sit and wait for work; they get up and look for ways to contribute throughout the development cycle and beyond.

If testing on an agile project felt just like testing on a traditional project, we wouldn’t feel the need to write a book. Let’s compare and contrast these testing methods.

**Traditional vs. Agile Testing**

It helps to start by looking at similarities between agile testing and testing in traditional software development. Consider Figure 1-4.

In the phased approach diagram, it is clear that testing happens at the end, right before release. The diagram is idealistic, because it gives the impression there is as much time for testing as there is for coding. In many projects, this is not the case. The testing gets “squished” because coding takes longer than expected, and because teams get into a code-and-fix cycle at the end.

Agile is iterative and incremental. This means that the testers test each increment of coding as soon as it is finished. An iteration might be as short as one week, or as long as a month. The team builds and tests a little bit of code, making sure it works correctly, and then moves on to next piece that needs to be built. Programmers never get ahead of the testers, because a story is not “done” until it has been tested. We’ll talk much more about this throughout the book.

There’s tremendous variety in the approaches to projects that agile teams take. One team might be dedicated to a single project or might be part of another
bigger project. No matter how big your project is, you still have to start somewhere. Your team might take on an epic or feature, a set of related stories at an estimating meeting, or you might meet to plan the release. Regardless of how a project or subset of a project gets started, you’ll need to get a high-level understanding of it. You might come up with a plan or strategy for testing as you prepare for a release, but it will probably look quite different from any test plan you’ve done before.

Every project, every team, and sometimes every iteration is different. How your team solves problems should depend on the problem, the people, and the tools you have available. As an agile team member, you will need to be adaptive to the team’s needs.

Rather than creating tests from a requirements document that was created by business analysts before anyone ever thought of writing a line of code, someone will need to write tests that illustrate the requirements for each story days or hours before coding begins. This is often a collaborative effort between a
business or domain expert and a tester, analyst, or some other development team member. Detailed functional test cases, ideally based on examples provided by business experts, flesh out the requirements. Testers will conduct manual exploratory testing to find important bugs that defined test cases might miss. Testers might pair with other developers to automate and execute test cases as coding on each story proceeds. Automated functional tests are added to the regression test suite. When tests demonstrating minimum functionality are complete, the team can consider the story finished.

If you attended agile conferences and seminars in the early part of this decade, you heard a lot about TDD and acceptance testing but not so much about other critical types of testing, such as load, performance, security, usability, and other “ility” testing. As testers, we thought that was a little weird, because all these types of testing are just as vital on agile projects as they are on projects using any other development methodology. The real difference is that we like to do these tests as early in the development process as we can so that they can also drive design and coding.

If the team actually releases each iteration, as Lisa’s team does, the last day or two of each iteration is the “end game,” the time when user acceptance testing, training, bug fixing, and deployments to staging environments can occur. Other teams, such as Janet’s, release every few iterations, and might even have an entire iteration’s worth of “end game” activities to verify release readiness. The difference here is that all the testing is not left until the end.

As a tester on an agile team, you’re a key player in releasing code to production, just as you might have been in a more traditional environment. You might run scripts or do manual testing to verify all elements of a release, such as database update scripts, are in place. All team members participate in retrospectives or other process improvement activities that might occur for every iteration or every release. The whole team brainstorms ways to solve problems and improve processes and practices.

Agile projects have a variety of flavors. Is your team starting with a clean slate, in a greenfield (new) development project? If so, you might have fewer challenges than a team faced with rewriting or building on a legacy system that has no automated regression suite. Working with a third party brings additional testing challenges to any team.

Whatever flavor of development you’re using, pretty much the same elements of a software development life cycle need to happen. The difference
with agile is that time frames are greatly shortened, and activities happen concurrently. Participants, tests, and tools need to be adaptive.

The most critical difference for testers in an agile project is the quick feedback from testing. It drives the project forward, and there are no gatekeepers ready to block project progress if certain milestones aren’t met.

We’ve encountered testers who resist the transition to agile development, fearing that “agile development” equates with chaos, lack of discipline, lack of documentation, and an environment that is hostile to testers. While some teams do seem to use the “agile” buzzword to justify simply doing whatever they want, true agile teams are all about repeatable quality as well as efficiency. In our experience, an agile team is a wonderful place to be a tester.

**Whole-Team Approach**

One of the biggest differences in agile development versus traditional development is the agile “whole-team” approach. With agile, it’s not only the testers or a quality assurance team who feel responsible for quality. We don’t think of “departments,” we just think of the skills and resources we need to deliver the best possible product. The focus of agile development is producing high-quality software in a time frame that maximizes its value to the business. This is the job of the whole team, not just testers or designated quality assurance professionals. Everyone on an agile team gets “test-infected.” Tests, from the unit level on up, drive the coding, help the team learn how the application should work, and let us know when we’re “done” with a task or story.

An agile team must possess all the skills needed to produce quality code that delivers the features required by the organization. While this might mean including specialists on the team, such as expert testers, it doesn’t limit particular tasks to particular team members. Any task might be completed by any team member, or a pair of team members. This means that the team takes responsibility for all kinds of testing tasks, such as automating tests and manual exploratory testing. It also means that the whole team thinks constantly about designing code for testability.

The whole-team approach involves constant collaboration. Testers collaborate with programmers, the customer team, and other team specialists—and not just for testing tasks, but other tasks related to testing, such as building infrastructure and designing for testability. Figure 1-5 shows a developer reviewing reports with two customers and a tester (not pictured).
The whole-team approach means everyone takes responsibility for testing tasks. It means team members have a range of skill sets and experience to employ in attacking challenges such as designing for testability by turning examples into tests and into code to make those tests pass. These diverse viewpoints can only mean better tests and test coverage.

Most importantly, on an agile team, anyone can ask for and receive help. The team commits to providing the highest possible business value as a team, and the team does whatever is needed to deliver it. Some folks who are new to agile perceive it as all about speed. The fact is, it’s all about quality—and if it’s not, we question whether it’s really an “agile” team.

Your situation is unique. That’s why you need to be aware of the potential testing obstacles your team might face and how you can apply agile values and principles to overcome them.
SUMMARY

Understanding the activities that testers perform on agile teams helps you show your own team the value that testers can add. Learning the core practices of agile testing will help your team deliver software that delights your customers.

In this chapter, we’ve explained what we mean when we use the term “agile testing.

- We showed how the Agile Manifesto relates to testing, with its emphasis on individuals and interactions, working software, customer collaboration, and responding to change.
- We provided some context for this book, including some other terms we use such as “tester,” “programmer,” “customer,” and related terms so that we can speak a common language.
- We explained how agile testing, with its focus on business value and delivering the quality customers require, is different from traditional testing, which focuses on conformance to requirements.
- We introduced the “whole-team” approach to agile testing, which means that everyone involved with delivering software is responsible for delivering high-quality software.
- We advised taking a practical approach by applying agile values and principles to overcome agile testing obstacles that arise in your unique situation.
This page intentionally left blank
Abbot GUI test tool, 127
Acceptance tests. See also Business-facing tests
definition, 501
Remote Data Monitoring system example, 245
UAT (user acceptance testing) compared with, 130
Ad hoc testing, 198
Adaptability, skills and, 39–40
ADEPT (AS400 Displays for External Prototyping and Testing), 117–118
Advance clarity
customers speaking with one voice, 373–374
determining story size, 375–376
gathering all viewpoints regarding requirements, 374–375
overview of, 140–142, 373
Advance preparation
downside of, 373
how much needed, 372–373
Agile development
Agile manifesto and, 3–4
barriers to. See Barriers to adopting agile development
team orientation of, 6
Agile Estimating and Planning (Cohn), 331, 332
Agile manifesto
team focus, 30
statement of, 4
value statements in, 21
Agile principles. See Principles, for agile testers
Agile testers. See also Testers
agile testing mind-set, 482–483
definition, 4
giving all team members equal weight, 31
hiring, 67–69
what they are, 19–20
Agile testing
definition, 6
as mind-set, 20–21
what we mean, 4–7
Agile values, 3–4
Alcea’s FIT IssueTrack, 84
Alpha tests, 466–467
ant, 284
as build tool, 126
continual builds and, 175, 291
AnthillPro, 126
ANTS Profiler Pro, 234
Apache JMeter. See JMeter
API-layer functional test tools, 168–170
Fit and FitNesse, 168–170
overview of, 168
testing web Services, 170
API testing
automating, 282
overview of, 205–206
APIs (application programming interfaces), 501
Appleton, Brad, 124
Application under test (AUT), 246
Applications
  integration testing with external applications, 459
  Remote Data Monitoring system example, 242–243
Architecture
  incremental approach to testing, 114
  layered, 116
  Quadrant 1 tests and, 99
  scalability and, 104, 221
  testable, 30, 115, 182, 184, 267
AS400 Displays for External Prototyping and Testing (ADEPT), 117–118
Assumptions, hidden
  agile testers response to, 25
  failure to detect, 32
  questions that uncover, 136
  worst-case scenarios and, 334
Attitude
  agile testing mind-set, 482–483
  barriers to adopting agile development, 48 vs. skills, 20
Audits, compliance with audit requirements, 89–90
AUT (application under test), 143, 225, 246, 317
Authorization, security testing and, 224
Automated regression testing
  key success factors, 484
  release candidates and, 458
  as a safety net, 261–262
Automated test lists, test plan alternatives, 353–354
Automation
  code flux and, 269
  of deployment, 232
  driving development with, 262–263
  of exploratory testing, 201
  fear of, 269–270
  feedback from, 262
  freeing people for other work, 259–261
  of functional test structure, 245–247
  home-brewed test, 175
  investment required, 267–268
  learning curve, 266–267
  legacy code and, 269
  maintainability and, 227–228
  manual testing vs., 258–259
  obstacles to, 264–265
  old habits and, 270
  overview of, 255
  programmers’ attitude regarding, 265–266
  reasons for, 257–258
  responding to change and, 29
  ROI and, 264
  task cards and, 394–395
  testability and, 149–150
  tests as documentation, 263–264
Automation strategy
  agile coding practices and, 303–304
  applying one tool at a time, 312–313
  data generation tools, 304–305
  database access and, 306–310
  design and maintenance and, 292–294
  developing, 288–289
  identifying tool requirements, 311–312
  implementing, 316–319
  iterative approach, 299–300
  keep it simple, 298–299
  learning by doing, 303
  managing automated tests, 319
  multi-layered approach to, 290–292
  organizing test results, 322–324
  organizing tests, 319–322
  overview of, 273
  principles, 298
  record/playback tools and, 294, 296–297
  starting with area of greatest pain, 289–290
  taking time to do it right, 301–303
  test automation pyramid, 276–279
  test categories, 274–276
  tool selection, 294–298, 313–316
  understanding purpose of tests and, 310–311
  what can be automated, 279–285
  what might be difficult to automate, 287–288
what should not be automated, 285–287
whole team approach, 300–301

Automation tools, 164–177
  API-layer functional test tools, 168–170
  builds and, 126
  GUI test tools, 170–176
  overview of, 164–165
  unit-level test tools, 165–168
  web services test tool, 170

B
Bach, James, 195, 200, 212
Bach, Jonathan, 201
Back-end testing
  behind the GUI, 282
  non-UI testing, 204–205
Bamboo, 126
Barriers to adopting agile development, 44–49
  conflicting or multiple roles, 45
  cultural differences among roles, 48–49
  lack of training, 45
  lack of understanding of agile concepts, 45–48
  loss of identity, 44–45
  overview of, 44
  past experience and attitudes, 48
Baselines
  break-test baseline technique, 363
  performance, 235–237
Batch
  files, 251
  processing, 345
  scheduling process, 182
BDD (Behavior-driven development)
  easyb tool, 166–168
  tools for Quadrant 1 tests, 127
Beck, Kent, 26, 99
Benander, Mark, 51
Benchmarking, 237
Berczuk, Stephen, 124
Beta testing, 466–467
Big picture
  agile testers focus on, 23
  high-level tests and examples, 397–402
  key success factors, 490–491
  peril of forgetting, 148
  regression tests and, 434
Bolton, Michael, 195
Bos, Erik, 114
Boundary conditions
  API testing and, 205
  automation and, 11
  data generation tools and, 304
  identifying test variations, 410
  writing test cases for, 137
Boyer, Erika, 140, 163, 372, 432
Brainstorming
  automation giving testers better work, 260
  prior to iteration, 370, 381
  quadrants as framework for, 253
  taking time for, 301
  testers, 121
Break-test baseline technique, 363
Browsers, compatibility testing and, 230
Budget limits, 55
Bug tracking. See Defect tracking
Bugs. See Defects
Build
  automating, 280–282
  challenging release candidate builds, 473
  definition, 501
  incremental, 178–179
  speeding up, 118–119
Build automation tools, 126, 282
Build/Operate/Check pattern, 180
Build tools, 126
BuildBeat, 126
Business analysts, 374
Business expert role
  agreement regarding requirements, 428, 430
  common language and, 134, 291, 414
  on customer team, 6–7
  iteration demo and, 443
  language of, 291
  Power of Three and, 482
  tools geared to, 134
Business-facing tests
  agile testing as, 6
  Quadrants 2 & 3, 97–98
  technology-facing tests compared with, 120
Business-facing tests, critiquing the product
  (Quadrant 3), 189–215
  acceptance tests, 245
  API testing, 205–206
  demonstrations, 191–192
  emulator tools, 213–214
  end-to-end tests, 249–250
  exploratory testing, 195–202, 248–249
  generating test data, 212
  GUI testing, 204
  monitoring tools, 212–213
  overview of, 189–191
  reports, 208–210
  scenario testing, 192–195
  session-based testing, 200–201
  setting up tests, 211–212
  simulator tools, 213
  tools for exploratory testing, 210–211
  usability testing, 202–204
  user acceptance testing, 250
  user documentation, 207–208
  web services testing, 207
Business-facing tests, supporting team
  (Quadrant 2), 129–151
  advance clarity, 140–142
  automating functional tests, 245–247
  common language and, 134–135
  conditions of satisfaction and, 142–143
  doneness, 146–147
  driving development with, 129–132
  eliciting requirements, 135–140
  embedded testing, 248
  incremental approach, 144–146
  requirements quandary and, 132–134
  ripple effects, 143–144
  risk mitigation and, 147–149
  testability and automation, 149–150
  toolkit for. See Toolkit (Quadrant 2)
  web services testing, 247–248
Business impact, 475–476

Business value
  adding value, 31–33
  as goal of agile development, 5–8, 69, 454
  metrics and, 75
  release cycles and, 3
  role, function, business value pattern, 155
  team approach and, 16
  Busse, Mike, 106, 235, 284, 313
  Buwalda, Hans, 193

C
  Canonical data, automating databases and, 308–309
  Canoo WebTest
    automating GUI tests, 184, 186
    GUI regression test suite, 291
    GUI smoke tests, 300
    GUI test tools, 174–175
    organizing tests and, 320
    scripts and, 320
    XML Editor for, 125
  Capability Maturity Model Integration (CMMI), 90–91
  Capture-playback tool, 267
  Celebrating successes
    change implementation and, 50–52
    iteration wrap up and, 449–451
  Chandra, Apurva, 377
  Chang, Tae, 53–54
  Change
    celebrating successes, 50–52
    giving team ownership, 50
    introducing, 49
    not coming easy, 56–57
    responsiveness to, 28–29
    talking about fears, 49–50
  Checklists
    release readiness, 474
    tools for eliciting examples and requirements, 156
  CI. See Continuous integration (CI)
  CI Factory, 126
  CMMI (Capability Maturity Model Integration), 90–91
Co-location, team logistics and, 65–66
Coaches
  adjusting to agile culture and, 40
  learning curve and, 266
  providing encouragement, 69
  skill development and, 122
  training and, 45–46
Cockburn, Alistair, 115
Code
  automation and code flux, 269
  automation and legacy code, 269
  automation strategy and, 303–304
  documentation of, 251
  standards, 227
  writing testable, 115
Code coverage, release metrics, 360–364
Coding and testing, 405–441
  adding complexity, 407
  alternatives for dealing with bugs, 424–428
  choosing when to fix bugs, 421–423
  collaborating with programmers, 413–414
  dealing with bugs, 416–419
  deciding which bugs to log, 420–421
  driving development and, 406
  facilitating communication, 429–432
  focusing on one story, 411–412
  identifying variations, 410
  iteration metrics, 435–440
  media for logging bugs, 423–424
  overview of, 405
  Power of Three for resolving differences in viewpoint, 411
  regression testing and, 432–434
  resources, 434–435
  risk assessment, 407–409
  as simultaneous process, 409–410, 488–489
  starting simple, 406, 428–429
  talking to customers, 414–415
  tests that critique the product, 412–413
Cohn, Mike, 50, 155, 276, 296, 331, 332
Collaboration
  with customers, 396–397
  key success factors, 489–490
  with programmers, 413–414
  whole team approach, 15–16
Collino, Alessandro, 103, 363
Communication
  common language and, 134–135
  with customer, 140, 396–397
  DTS (Defect Tracking System) and, 83
  facilitating, 23–25, 429–432
  product delivery and, 462–463
  size as challenge to, 42–43
  between teams, 69–70
  test results, 357–358
Comparisons, automating, 283
Compatibility testing, 229–230
Component tests
  automating, 282
  definition, 501
  supporting function of, 5
Conditions of satisfaction
  business-facing tests and, 142–143
  definition, 501–502
Context-driven testing
  definition, 502
  quadrants and, 106–107
Continuous build process
  failure notification and, 112
  feedback and, 119
  FitNesse tests and, 357
  implementing, 114
  integrating tools with, 175, 311
  source code control and, 124
  what testers can do, 121
Continuous feedback principle, 22
Continuous improvement principle, 27–28
Continuous integration (CI)
  automating, 280–282
  as core practice, 486–487
  installability and, 231–232
  Remote Data Monitoring system example, 244
  running tests and, 111–112
Conversion, data migration and, 460–461
Core practices
  coding and testing as one process, 488–489
  continuous integration, 486–487
Core practices, continued
  incremental approach, 488
  overview of, 486
  synergy between practices, 489
  technical debt management, 487–488
  test environments, 487
Courage, principles, 25–26, 71
Credibility, building, 57
Critiquing the product
  business facing tests. See Business-facing tests,
  critiquing the product (Quadrant 3)
  technology-facing tests. See
  Technology-facing tests, critiquing the
  product (Quadrant 4)
CrossCheck, testing Web Services, 170
CruiseControl, 126, 244, 291
Cultural change, 37. See also Organizations
Cunningham, Ward, 106, 168, 506
Customer expectations
  business impact and, 475–476
  production support, 475
Customer-facing test. See Business-facing tests
Customer support, DTS (Defect Tracking System) and, 82
Customer team
  definition, 502
  interaction between customer and developer teams, 8
  overview of, 7
Customer testing
  Alpha/Beta testing, 466–467
  definition, 502
  overview of, 464
  UAT (user acceptance testing), 464–466
Customers
  collaborating with, 396–397, 489–490
  considering all viewpoints during iteration planning, 388–389
  delivering value to, 22–23
  importance of communicating with, 140, 414–415, 444
  iteration demo, 191–192, 443–444
  participation in iteration planning, 384–385
  relationship with, 41–42
  reviewing high-level tests with, 400
  speaking with one voice, 373–374
CVS, source code control and, 124

D
Data
  automating creation or setup, 284–285
  cleanup, 461
  conversion, 459–461
  release planning and, 348
  writing task cards and, 392
  Data-driven tests, 182–183
  Data feeds, testing, 249
  Data generation tools, 304–305
  Data migration, automating, 310, 460
  Databases
    avoiding access when running tests, 306–310
    canonical data and automation, 308–309
    maintainability and, 228
    product delivery and updates, 459–461
    production-like data and automation, 309–310
    setting up/tearing down data for each automated test, 307–308
    testing data migration, 310
  De Souza, Ken, 223
  Deadlines, scope and, 340–341
  Defect metrics
    overview of, 437–440
    release metrics, 364–366
  Defect tracking, 79–86
    DTS (Defect Tracking System), 79–83
    keeping focus and, 85–86
    overview of, 79
    reasons for, 79
    tools for, 83–85
  Defect Tracking System. See DTS (Defect Tracking System)
Defects
  alternatives for dealing with bugs, 424–428
  choosing when to fix bugs, 421–423
dealing with bugs, 416–419
deciding which bugs to log, 420–421
media for logging bugs, 423–424
metrics and, 79
TDD (test-driven development) and, 490
writing task cards and, 391–392
zero bug tolerance, 79, 418–419

Deliverables
“fit and finish” deliverables, 454
nonsoftware, 470
overview of, 468–470

Delivering product
Alpha/Beta testing, 466–467
business impact and, 475–476
communication and, 462–463
customer expectations, 475
customer testing, 464
data conversion and database updates, 459–461
deliverables, 468–470
end game, 456–457
installation testing, 461–462
integration with external applications, 459
nonfunctional testing and, 458–459
overview of, 453
packaging, 474–475
planning time for testing, 455–456
post-development testing cycles, 467–468
production support, 475
release acceptance criteria, 470–473
release management, 470, 474
releasing product, 470
staging environment and, 458
testing release candidates, 458
UAT (user acceptance testing), 464–466
what if it is not ready, 463–464
what makes a product, 453–455

Demos/demonstrations
of an iteration, 443–444
value to customers, 191–192

Deployment, automating, 280–282

Design
automation strategy and, 292–294
designing with testing in mind, 115–118

Detailed test cases
art and science of writing, 178
big picture approach and, 148–149
designing with, 401

Developer team
interaction between customer and developer teams, 8
overview of, 7–8

Development
agile development, 3–4, 6
automated tests driving, 262–263
business-facing tests driving,
129–132
coding driving, 406
post-development testing cycles,
467–468
Development spikes, 381
Development team, 502
diff tool, 283

Distributed teams, 431–432
defect tracing systems, and, 82
physical logistics, 66
online high level tests for, 399
online story board for, 357
responding to change, 29
software-based tools to elicit examples and requirements, and, 163–164

Documentation
automated tests as source of, 263–264
problems and fixes, 417
reports, 208–210
of test code, 251
tests as, 402
user documentation, 207–208

Doneness
knowing when a story is done, 104–105
multitiered, 471–472

Driving development with tests. See TDD (test-driven development)

DTS (Defect Tracking System), 80–83
benefits of, 80–82
choosing media for logging bugs, 424
documenting problems and fixes, 417
DTS (Defect Tracking System), continued
  logging bugs and, 420
  reason for not using, 82–83
Dymond, Robin, xxx
Dynamic analysis, security testing tools, 225

E
easyb behavior-driven development tool, 165–168
EasyMock, 127
Eclipse, 125, 316
Edge cases
  identifying variations, 410
  not having time for, 112
  starting simple and then adding complexity, 406–407
  test cases for, 137
Embedded system, Remote Data Monitoring example, 248
Empowerment, of teams, 44
Emulator tools, 213–214
End game
  Agile testing, 91
  iteration, 14
  product delivery and, 456–457
  release and, 327
End-to-end tests, 249–250
Enjoyment, principle of, 31
Environment, test environment, 347–348
Epic. See also Themes
  definition, 502
  features becoming, 502
  iterations in, 76, 329
  planning, 252
ePlan Services, Inc., xli, 267
Errors, manual testing and, 259
Estimating story size, 332–338
eValid, 234
Event-based patterns, test design patterns, 181
Everyday Scripting with Ruby for Teams, Testers, and You (Marick), 297, 303
Example-driven development, 378–380
Examples
  for eliciting requirements, 136–137
  tools for eliciting examples and requirements, 155–156
Executable tests, 406
Exploratory testing (ET)
  activities, characteristics, and skills (Hagar), 198–200
  attributes of exploratory tester, 201–202
  automation of, 201
  definition, 502–503
  end game and, 457
  explained (Bolton), 195–198
  manual testing and, 280
  monitoring tools, 212
  overview of, 26, 195
Remote Data Monitoring system example, 248–249
session-based testing and, 200–201
setup, 211–212
simulators and emulators, 212–213
tests that critique the product, 412–413
tools for, 210–212
tools for generating test data, 212
what should not be automated, 286
External quality, business facing tests defining, 99, 131
External teams, 43, 457
Extreme Programming. See XP (Extreme Programming)
Extreme Programming Explained (Beck), 26

F
Face-to-face communication, 23–25
Failover tests, 232
Failure, courage to learn from, 25
Fault tolerance, product delivery and, 459
Fear
  barriers to automation, 269–270
  change and, 49–50
Fearless Change (Manns and Rising), 121
Feathers, Michael, 117, 288
Features
  defects vs., 417–418
  definition, 502–503
  focusing on value, 341
Feedback
automated tests providing, 262
continuous feedback principle, 22
iterative approach and, 299–300
key success factors, 484–486
managing tests for, 323–324
Quadrant 1 tests and, 118–119
“Fit and finish” deliverables, 454
Fit (Framework for Integrated Test), 134–135
API-layer functional test tools, 168–169
automation test pyramid and, 278
FIT IssueTrack, Alcea, 83–84
FitNesse
advantages of, 163
API-layer functional test tools, 169–170
automating functional tests with, 30, 145
business-facing tests with, 154, 178
collaboration and, 164
continual builds and, 119, 357
data verification with, 287
doneness and, 472
encouraging use of, 122
examples and, 136, 169
feedback and, 323–324
file parsing rules illustrated with, 205
functional testing behind the GUI, 291, 300
home-grown scripts and, 305
JUnit compared with, 299
keywords or actions words for automating tests, 182–183
manual vs. automated testing, 210
memory demands of, 306
organizing tests and, 319–320
overview of, 168–170
remote testing and, 432
“start, stop, continue” list, 446
support for source code control tools, 320
test automation pyramid and, 278
test cards and, 389–390
test cases as documentation, 402
test design and maintenance, 292
testing database layer with, 284
testing stories, 395
traceability requirements and, 88
user acceptance testing, 295
wikis and, 186
Fleisch, Patrick, 377, 440
Flow diagrams
scenario testing and, 194–195
tools for eliciting examples and requirements, 160–163
Fowler, Martin, 117
Framework for Integrated Test. See Fit (Framework for Integrated Test)
Frameworks, 90–93
ftptt, 234
Functional analysts, 386
Functional testing
compatibility issues and, 230
definition, 502–503
end-to-end tests, 249–250
layers, 246
nonfunctional tests compared with, 225
Remote Data Monitoring system example, 245–247
G
Galen, Bob, 455–456, 471
Gärtner, Markus, 395, 476
Geographically dispersed teams
coping with, 376–378
facilitating communication and, 431–432
Gheorghiu, Grig, 225–226, 234
Glover, Andrew, 166
Greenfield projects
code testing and, 116
definition, 502–503
GUI (graphical user interface)
automation strategy and, 293
code flux and, 269
standards, 227
GUI smoke tests
Canoo WebTest and, 300
continual builds and, 119
defect metrics, 437
GUI test tools, 170–176
  Canoo Web Test, 174–175
  “home-brewed” test automation tools, 175
  open source test tools, 172
  overview of, 170–171
  record/playback tools, 171–172
  Ruby with Watir, 172–174
  Selenium, 174
GUI testing
  API testing, 205–206
  automating, 282–283, 295–296
  automation test pyramid and, 278
  GUI smoke tests, 119, 300, 437
  overview of, 204
  Web service testing, 207

I
  IBM Rational ClearCase, 124
  IDEs (Integrated Development Environments)
    definition, 502–503
    log analysis tools, 212
    tools for Quadrant 1 tests, 124–126
  “ility” testing
    compatibility testing, 229–230
    installability testing, 231–232
    interoperability testing, 228–229
    maintainability testing, 227–228
    reliability testing, 230–231, 250–251
    security testing, 223–227
Impact, system-wide, 342
Implementing Lean Software Development: From Concept to Cash (Poppendieck), 74, 416
Improvement
  approach to process improvement, 448–449
  continuous improvement principle, 27–28
  ideas for improvement from retrospectives, 447–449
Incremental development
  building tests incrementally, 178–179
  as core practice, 488
  “ilities” tests and, 232
  thin slices, small chunks, 144–146
  traditional vs. agile testing, 12–13
Index cards, logging bugs on, 423
Infrastructure
  Quadrant 1 tests, 111–112
  test infrastructure, 319
  test plans and, 346–347
  Installability testing, 231–232
  Installation testing, 461–462
Integrated Development Environments. See IDEs (Integrated Development Environments)
Integration testing
  interoperability and, 229
  product and external applications, 459
IntelliJ IDEA, 125
Internal quality
  measuring internal quality of code, 99
  meeting team standards, 366

H
  Hagar, Jon, 198
  Hardware
    compatibility and, 229
    cost of test environments, 487
    functional testing and, 230
    investing in automation and, 267
    production environment and, 310
    scalability and, 233
    test infrastructure, 319
    testing product installation, 462
  Hendrickson, Elisabeth, 203, 315–316
  High-level test cases, 397–402
    mockups, 398–399
    overview of, 397–398
    reviewing with customers, 400
    reviewing with programmers, 400–401
    test cases as documentation, 402
  Hiring a tester, 67–69
  Holzer, Jason, 220, 448
  Home-grown test tool
    automation tools, 314
    GUI test tools, 175
    test results, 323
  httpperf, 234
  Hudson, 126
Quadrant 1 tests and, 111
speed and, 112
Interoperability testing, 228–229
Investment, automation requiring, 267–268
Iteration
automation strategy and, 299–300
definition, 502–503
demo, 443–444
life of a tester and, 327
pre-iteration activities. See Pre-iteration activities
prioritizing stories and, 338
review, 415, 435–437
traditional vs. agile testing, 12–13
Iteration kickoff, 383–403
collaboration with customers, 396–397
considering all viewpoints, 385–389
controlling workload, 393
high-level tests and examples, 397–402
iteration planning, 383–384
learning project details, 384–385
overview of, 383
testable stories, 393–396
writing task cards, 389–392
Iteration metrics, 435–440
defect metrics, 437–440
measuring progress with, 435–437
overview of, 435
usefulness of, 439–440
Iteration planning
considering all viewpoints, 385–389
controlling workload, 393
learning project details, 384–385
overview of, 383–384
writing task cards, 389–392
Iteration review meeting, 415
Iteration wrap up, 443–451
celebrating successes, 449–451
demo of iteration, 443–444
ideas for improvement, 447–449
retrospectives, 444–445
“start, stop, continue” exercise for retrospectives, 445–447
ITIL (Information Technology Infrastructure Library), 90–91
J
JBehave, 165
JConsole, 234
JMeter
performance baseline tests, 235
performance testing, 223, 234, 313
JMS (Java Messaging Service)
definition, 502–503
integration with external applications and, 243
testing data feeds and, 249
JProfiler, 234
JUnit
FitNesse as alternative for TDD, 299
functional testing, 176
load testing tools, 234–235
unit test tools, 126, 165, 291
JUnitPerf, 234
Just in time development, 369. See also Pre-iteration activities
K
Key success factors
agile testing mind-set, 482–483
automating regression testing, 484
big picture approach, 490–491
coding and testing as one process, 488–489
collaboration with customers, 489–490
continuous integration (CI), 486–487
feedback, 484–486
foundation of core practices, 486
incremental approach (thin slices, small chunks), 488
overview of, 481
synergy between practices, 489
technical debt management, 487–488
test environments, 487
whole team approach, 482
Keyword-driven tests, 182–183
King, Joseph, 176
Knowledge base, DTS, 80–81
Kohl, Jonathan, 201, 204, 211
König, Dierk, 320

L
Language, need for common, 134–135
Layered architecture, 116
Lean measurements, metrics, 74–75
Learning
  automation strategy and, 303
  continuous improvement principle, 27
Learning curve, automation and, 266–267, 303
Legacy code, 269
Legacy code rescue (Feathers), 117
Legacy systems
code, 269
  definition, 502–503
  logging bugs and, 421
testing, 117
Lessons Learned in Software Testing (Pettichord), 485
Lessons learned sessions, 383. See also
  Retrospectives
Lightweight processes, 73–74
Lightweight test plans, 350
Load testing. See Performance and load testing
LoadRunner, 234
LoadTest, 234
Logistics, physical, 65–66
LogWatch tool, 212
Loss of identity, QA teams fearing, 44–45
Louvion, Christophe, 63

M
Maintainability testing, 227–228
Management, 52–55
  advance clarity and, 373–374
  cultural change and, 52–54
  overview of, 52
  providing metrics to, 440
Managers
  cultural changes for, 52–54
  how to influence testing, 122–123
  speaking managerís language, 55
Manns, Mary Lynn, 121–122
Manual testing
  automation vs., 258–259
  peril of, 289
Marcano, Antony, 83, 426
Marick, Brian, 5, 24, 97, 134, 170, 203, 303
Martin, Micah, 169
Martin, Robert C., 169
Matrices
  high-level tests and, 398–399
  text matrices, 350–353
Maven, 126
McMahon, Chris, 260
Mean time between failure, reliability testing, 230
Mean time to failure, reliability testing, 230
Media, for logging bugs, 423–424
Meetings
  demonstrations, 71, 192
  geographically dispersed, 376
  iteration kickoff, 372
  iteration planning, 23–24, 244, 331, 384, 389
  iteration review, 71, 415
  pre-planning, 370–372
  release planning, 338, 345
  retrospective, 447
  scheduling, 70
  sizing process and, 336–337
  standup, 177, 429, 462
  team participation and, 32
test planning, 263
Memory leaks, 237–238
Memory management testing, 237–238
Meszaros, Gerald, 99, 111, 113, 138, 146, 182, 204, 291, 296, 430
Metrics, 74–79
  code coverage, 360–364
  communication of, 77–78
  defect metrics, 364–366, 437–440
  iteration metrics, 435–440
  justifying investment in automation, 268
  lean measurements, 74–75
  overview of, 74
  passing tests, 358–360
  reasons for tracking defects, 52, 75–77, 82
release metrics, 358
ROI and, 78–79
what not to do with, 77
XP radar charts, 47–48
Milestones, celebrating successes, 449–450
MIME (Multipurpose Internet Mail Extensions)
definition, 504
testing data feeds and, 249
Mind maps, 156–158
Mind-set
agile testing as, 20–21
key success factors, 482–483
pro-active, 369–370
“Mini-waterfall” phenomenon, 46–47
Mock objects
definition, 504
risk alleviation and, 459
tools for implementing, 127
unit tests and, 114
Mock-ups
facilitating communication and, 430
high-level tests and, 398–399
stories and, 380
tools for eliciting examples and requirements, 160
Model-driven development, 398
Models
quality models, 90–93
UI modeling example, 399
Monitoring tools, 212–213, 235
Multi-layered approach, automation strategy, 290–292
Multipurpose Internet Mail Extensions (MIME)
definition, 504
testing data feeds and, 249

N
Naming conventions, 227
Nant, 126
Navigation, usability testing and, 204
NBhave, 165
NeoLoad, 234
Nessus, vulnerability scanner, 226
.NET Memory Profiler, 234
NetBeans, 125
NetScout, 235
Non-functional testing. See also Technology-facing tests, critiquing the product (Quadrant 4)
delivering product and, 458–459
functional testing compared with, 225
requirements, 218–219
when to perform, 222
North, Dan, 165
NSpec, 165
NUnit, 126, 165
O
Oleszkiewicz, Jakub, 418
One-off tests, 286–287
Open source tools
agile open source test tools, 172–175
automation and, 314–315
GUI test tools, 172
IDEs, 124–125
OpenWebLoad, 234
Operating systems (OSs), compatibility testing and, 230
Organizations, 37–44
challenges of agile development, 35
conflicting cultures, 43
customer relationships and, 41–42
overview of, 37–38
quality philosophy, 38–40
size and, 42–43
sustainable pace of testing and, 40–41
team empowerment, 44
OSs (operating systems), compatibility testing and, 230
Ownership, giving team ownership, 50
P
Packaging, product delivery and, 474–475
Pair programming
code review and, 227
developers trained in, 61
Pair programming, continued
IDEs and, 125
team approach and, 244
Pair testing, 413
Passing tests, release metrics, 358–360
PerfMon, 235
Perforce, 124
Performance and load testing
automating, 283
baselines, 235–237
memory management testing, 237–238
overview of, 234
product delivery and, 458
scalability testing, 233–234
test environment, 237
tools for, 234–235
when to perform, 223
who performs the test, 220–221
Performance, rewards and, 70–71
Perils
forgetting the big picture, 148
quality police mentality, 39
the testing crunch, 416
waiting for Tuesdayís build, 280
you’re not really part of the team, 32
Perkins, Steve, 156, 159, 373
PerlClip
data generation tools, 305
tools for generating test data, 212
Persona testing, 202–204
Pettichord, Bret, 175, 264, 485
Phased and gated development, 73–74, 129
Physical logistics, 65–66
Planning
advance, 43
iteration. See Iteration planning
release/theme planning. See Release planning
testing. See Test planning
PMO (Project Management Office), 440
Pols, Andy, 134
Ports and Adapters pattern (Cockburn), 115
Post-development testing, 467–468
Post-iteration bugs, 421
Pounder, 234
Power of Three
business expert and, 482
finding a common language, 430
good communication and, 33, 490
problem solving and, 24
resolving differences in viewpoint, 401, 411
whole team approach and, 482
Pragmatic Project Automation, 260
Pre-iteration activities, 369–382
advance clarity, 373
benefits of working on stories in advance, 370–372
customers speaking with one voice, 373–374
determining story size, 375–376
evaluating amount of advance preparation needed, 372–373
examples, 378–380
gathering all viewpoints regarding requirements, 374–375
geographically dispersed team and, 376–378
overview of, 369
prioritizing defects, 381
pro-active mindset, 369–370
resources, 381
test strategies and, 380–381
Pre-planning meeting, 370–372
Principles, automation
agile coding practices, 303–304
iterative approach, 299–300
keep it simple, 298–299
learning by doing, 303
overview of, 298
taking time to do it right, 301–303
whole team approach, 300–301
Principles, for agile testers
continuous feedback, 22
continuous improvement, 27–28
courage, 25–26
delivering value to customer, 22–23
enjoyment, 31
face-to-face communication, 23–25
keeping it simple, 26–27
overview of, 21–22
people focus, 30
responsive to change, 28–29
self-organizing, 29–30
Prioritizing defects, 381
Prioritizing stories, 338–340
Pro-active mindset, 369–370
Product
  business value, 31–33
delivery. See Delivering product
tests that critique (Q3 & Q4), 101–104
what makes a product, 453–455
Product owner
  considering all viewpoints during iteration
    planning, 386–389
definition, 504
iteration planning and, 384
Scrum roles, 141, 373
tools geared to, 134
Production
  logging bugs and, 421
  support, 475
Production code
  automation test pyramid and, 277–278
definition, 504
delivering value to, 70
programmers writing, 48
source code control and, 434
synchronization with testing, 322
test-first development and, 113
tests supporting, 303–304
Production-like data, automating databases
  and, 309–310
Professional development, 57
Profiling tools, 234
Programmers
  attitude regarding automation, 265–266
  big picture tests, 397
  collaboration with, 413–414
  considering all viewpoints during iteration
    planning, 387–389
  facilitating communication and, 429–430
  reviewing high-level tests with, 400–401
tester-developer ratio, 66–67
testers compared with, 4, 5
  training, 61
  writing task cards and, 391
Project Management Office (PMO), 440
Projects, PAS example, 176–177
Prototypes
  accessible as common language, 134
  mock-ups and, 160
  paper, 22, 138–139, 380, 400, 414
  paper vs. Wizard of Oz type, 275
  UI (user interface), 107
Pulse, 126
PyUnit unit test tool for Python, 126
QA
QA (quality assurance)
  definition, 504
  in job titles, 31
  independent QA team, 60
  interchangeable with “test,” 59
  whole team approach, 39
  working on traditional teams, 9
Quadrant 1. See Technology-facing tests,
  supporting team (Quadrant 1)
Quadrant 2. See Business-facing tests,
  supporting team (Quadrant 2)
Quadrant 3. See Business-facing tests, critiquing
  the product (Quadrant 3)
Quadrant 4. See Technology-facing tests,
  critiquing the product (Quadrant 4)
Quadrants
  automation test categories, 274–276
  business facing (Q2 & Q3), 97–98
  context-driven testing and, 106–108
  critiquing the product (Q3 & Q4), 104
  managing technical debt, 106
  overview of, 97–98
  as planning guide, 490
  purpose of testing and, 97
  Quadrant 1 summary, 99
  Quadrant 2 summary, 99–100
  Quadrant 3 summary, 101–102
  Quadrant 4 summary, 102–104
  shared responsibility and, 105–106
  story completion and, 104–105
Quadrants, continued
  supporting the team (Q1 & Q2), 100–101
  technology facing (Q1 & Q4), 97–98
Quality
  customer role in setting quality standards, 26
  models, 90–93
  organizational philosophy regarding, 38–40
Quality assurance. See QA (quality assurance)
Quality police mentality, 57
Questions, for eliciting requirements, 135–136

R
Radar charts, XP, 47–48
Rasmusson, Jonathan, 11
Record/playback tools
  automation strategy and, 294, 296–297
  GUI test tools, 171–172
Recovery testing, 459
Redundancy tests, 232
Reed, David, 171, 377
Refactoring
  definition, 504
  IDEs supporting, 124–126
Regression suite, 434
Regression tests, 432–434
  automated regression tests as a safety net, 261–262
  automating as success factor, 484
  checking big picture, 434
  definition, 504
  exploratory testing and, 212
  keeping the build “green,” 433
  keeping the build quick, 433–434
  logging bugs and, 420
  regression suite and, 434
  release candidates and, 458
Release
  acceptance criteria, 470–473
  end game, 327, 456–457
  management, 474
  product delivery, 470
  what if it is not ready, 463–464
Release candidates
  challenging release candidate builds, 473
  definition, 505
  testing, 458
Release metrics
  code coverage, 360–364
  defect metrics, 364–366
  overview of, 358
  passing tests, 358–360
Release notes, 474
Release planning, 329–367
  overview of, 329
  prioritizing and, 338–340
  purpose of, 330–331
  scope, 340–344
  sizing and, 332–337
  test plan alternatives, 350–354
  test planning, 345–350
  visibility and, 354–366
Reliability testing
  overview of, 230–231
Remote Data Monitoring system example, 250–251
Remote Data Monitoring system example
  acceptance tests, 245
  application, 242–243
  applying test quadrants, 252–253
  automated functional test structure, 245–247
  documenting test code, 251
  embedded testing, 248
  end-to-end tests, 249–250
  exploratory testing, 248–249
  overview of, 242
  reliability testing, 250–251
  reporting test results, 251
  team and process, 243–244
  testing data feeds, 249
  unit tests, 244–245
  user acceptance testing, 250
  web services, 247–248
Remote team member. See Geographically dispersed teams
Rogers, Paul, 242, 310, 388, 398

ROI (return on investment)
  automation and, 264
definition, 505
  lean measurement and, 75
  metrics and, 78–79
  speaking managerís language, 55

Role, function, business value pattern, 155

Roles
  conflicting or multiple roles, 45
cultural differences among, 48–49
customer team, 7
developer team, 7–8
interaction of, 8

RPGUnit, 118

RSpec, 165, 318

Ruby Test::Unit, 170

Ruby with Watir
  functional testing, 247
  GUI testing, 285
  identifying defects with, 212
  keywords or actions words for automating tests, 182
  overview of, 172–174
test automation with, 186

RubyMock, 127

Rules, managing bugs and, 425

S

Safety tests, 232

Santos, Rafael, 448

Satisfaction conditions. See Conditions of satisfaction

Scalability testing, 233–234

Scenario testing, 192–193
  flow diagrams and, 194–195
  overview of, 192–195
  soap opera tests, 193

Scope, 340–344
  business-facing tests defining, 134
deadlines and timelines and, 340–341
  focusing on value, 341–342
  overview of, 340
  system-wide impact, 342
Scope, continued
  test plans and, 345
  third-party involvement and, 342–344
Scope creep, 385, 412
Scripts
  automating comparisons, 283
  as automation tools, 297
  conversion scripts, 461
  data generation tools, 305
  exploratory testing and, 211–212
Scrum
  product owner role, 141, 373
  Remote Data Monitoring system example, 244
  sprint reviews, 444
ScrumMaster
  approach to process improvement, 448–449
  sizing stories and, 336–337
  writing task cards and, 391
SDD (story test-driven development)
  identifying variations, 410
  overview of, 262–263
  test-first development and, 263
  testing web services and, 170
Security testing
  outside-in approach of attackers, 225
  overview of, 223–227
  specialized knowledge required for, 220
Selenium
  GUI test tools, 174–175
  implementing automation, 316–318
  open source tools, 163
  test automation with, 186, 316
Self-organization
  principles, 29–30
  self-organizing teams, 69
Session-based testing, 200–201
Setup
  automating, 284–285
  exploratory testing, 211–212
Shared resources
  access to, 43
  specialists as, 301
  writing tasks and, 390
Shared responsibility, 105–106
Shout-Out Shoebox, 450
“Show me,” collaboration with programmers, 413–414
Simplicity
  automation and, 298–299
  coding, 406
  logging bugs and, 428–429
  principle of “keeping it simple,” 26–27
Simulator tools
  embedded testing and, 248
  overview of, 213
Size, organizational, 42–43
Sizing stories, 332–337
  example of, 334–337
  how to, 332–333
  overview of, 332
  tester’s role in, 333–334
Skills
  adaptability and, 39–40
  vs. attitude, 20
  continuous improvement principle, 27
  who performs tests and, 220–221
Small chunks, incremental development, 144–146
SOAP
  definition, 505
  performance tests and, 223, 234
Soap opera tests, 193
soapUI
  definition, 505
  performance tests and, 223, 234
  testing Web Services, 170–171
SOATest, 234
Software-based tools, 163
Software Configuration Management Patterns: Effective Teamwork, Practical Integrations (Berczuk and Appleton), 124
Software Endgames (Galen), 471
Source code control
  benefits of, 255
  overview of, 123–124
  tools for, 124, 320
SOX compliance, 469
Speak with one voice, customers, 373–374
Specialization, 220–221
Speed as a goal, 112
Spikes, development and test, 381
Spreadsheets
  test spreadsheets, 353
  tools for eliciting examples and requirements, 159
Sprint reviews, 444. See also Demos/demonstrations
SQL*Loader, 460
Stability testing, 28
Staging environment, 458
Stand-up meetings, 177, 429, 462
Standards
  maintainability and, 227
  quality models and, 90–93
“Start, stop, continue” exercise, retrospectives, 445–447
Static analysis, security testing tools, 225
Steel thread, incremental development, 144, 338, 345
Stories. See also Business-facing tests
  benefits of working on in advance of iterations, 370–372
  briefness of, 129–130
  business-facing tests as, 130
  determining story size, 375–376
  focusing on one story when coding, 411–412
  identifying variations, 410
  knowing when a story is done, 104–105
  logging bugs and, 420–421
  mock-ups and, 380
  prioritizing, 338–340
  resources and, 381
  scope and, 340
  sizing. See Sizing stories
  starting simple, 133, 406
  story tests defined, 505
  system-wide impact of, 342
  test plans and, 345
test strategies and, 380–381
  testable, 393–396
  treating bugs as, 425
Story boards
  burndown charts, 429
  definition, 505–506
  examples, 356–357
  online, 357, 384
  physical, 356
  stickers and, 355
  tasks, 222, 355, 436
  virtual, 357, 384, 393
  work in progress, 390
Story cards
  audits and, 89
  dealing with bugs and, 424–425
  iteration planning and, 244
  story narrative on, 409
Story test-driven development. See SDD
  (story test-driven development)
Strangler application (Fowler), 116–117
Strategy
  automation. See Automation strategy
test planning vs. test strategy, 86–87
test strategies, 380–381
Strategy, for writing tests
  building tests incrementally, 178–179
  iteration planning and, 372
  keep the tests passing, 179
  overview of, 177–178
  test design patterns, 179–183
testability and, 183–185
Stress testing. See Load testing
Subversion (SVN), 124, 320
Success factors. See Key success factors
Successes, celebrating
  change implementation and, 50–52
  iteration wrap up and, 449–451
Sumrell, Megan, 365, 450
Sustainable pace, of testing, 40–41, 303
SVN (Subversion), 124, 320
SWTBot GUI test tool, 127
Synergy, between practices, 489
System, system-wide impact of story, 342
T

taxi-f, 212
Tartaglia, Coni, 439, 454, 470, 473
Task boards. See Story boards
Task cards
  automating testing and, 394–395
  iteration planning and, 389–392
  product delivery and, 462–463
Tasks
  completing testing tasks, 415–416
  definition, 505–506
TDD (test-driven development)
  automated tests driving, 262–263
  defects and, 490
  definition, 506
  overview of, 5
Test-First Development compared with, 113–114
  unit tests and, 111, 244–245
Team City, 126
Team structure, 59–65
  agile project teams, 64–65
  independent QA team, 60
  integration of testers into agile project, 61–63
  overview of, 59
  traditional functional structure vs agile structure, 64
Teams
  automation as team effort, 484
  building, 69–71
  celebrating success, 50–52
  co-located, 65–66
  controlling workload and, 393
  customer, 7
  developer, 7–8
  empowerment of, 44
  facilitating communication and, 429–432
  geographically dispersed, 376–378, 431–432
  giving all team members equal weight, 31
  giving ownership to, 50
  hiring agile tester for, 67–69
  interaction between customer and developer teams, 8
  iteration planning and, 384–385
  logistics, 59
  problem solving and, 123
Remote Data Monitoring system example, 243–244
  shared responsibility and, 105–106
  traditional, 9–10
  using tests to support Quadrants 1 and 2, 100–101
  whole team approach. See Whole team approach
  working on agile teams, 10–12
Teardown, for tests, 307–308
Technical debt
  defects as, 418
  definition, 506
  managing, 106, 487–488
Technology-facing tests
  overview of, 5
  Quadrants 1 & 4, 97–98
  Technology-facing tests, critiquing the product (Quadrant 4), 217–239
  baselines, 235–237
  coding and testing and, 412–413
  compatibility testing, 229–230
  installability testing, 231–232
  interoperability testing, 228–229
  maintainability testing, 227–228
  memory management testing, 237–238
  overview of, 217–219
  performance and load testing, 234
  performance and load testing tools, 234–235
  reliability testing, 230–231, 250–251
  scalability testing, 233–234
  security testing, 223–227
  test environment and, 237
  when to use, 222–223
  who performs the test, 220–222
Technology-facing tests, supporting team (Quadrant 1)
  build tools, 126
  designing with testing in mind, 115–118
  ease of accomplishing tasks, 114–115
  IDEs for, 124–126
  infrastructure supporting, 111–112
overview of, 109–110
purpose of, 110–111
source code control, 123–124
speed as benefit of, 112–114
timely feedback, 118–119
toolkit for, 123
unit test tools, 126–127
unit tests, 244–245
what to do if team doesn’t perform these tests, 121–123
where/when to stop, 119–121
Test automation pyramid
multi-layered approach to automation and, 290–291
overview of, 276–279
three little pigs metaphor, 278
Test behind UI, 282
Test cases
adding complexity, 407
as documentation, 402
element-driven development, 379
identifying variations, 410
starting simple, 406
Test coverage (and/or code coverage), 360–364
Test design patterns, 179–183
Build/Operate/Check pattern, 180
data-driven and keyword-driven tests, 182–183
overview of, 179
test genesis patterns (Veragen), 179
time-based, activity, and event patterns, 181
Test doubles
definition, 506
layered architectures and, 116
Test-driven development. See TDD (test-driven development)
Test environments, 237, 487
Test-First Development
definition, 506
TDD (test-driven development) compared with, 113–114
Test management, 186
Test management toolkit (Quadrant 2), 186
Test plan alternatives, 350–354
Test planning, 345–350
automated test lists, test plan alternatives, 353–354
infrastructure and, 346–347
overview of, 86, 345
reasons for writing, 345–346
test environment and, 347–348
test plan alternatives, 350–354
test plans, lightweight 350
test plan sample, 351
test strategy vs., 86–88
traceability and, 88
types of tests and, 346
where to start, 345
Test results
communicating, 357–358
organizing, 322–324
release planning and, 349–350
Test skills. See Skills
Test spikes, 381
Test spreadsheets, 353
Test strategy
iterations, pre-iteration activities and, 380–381
test plan vs., 86–88
Test stubs
definition, 506
integration with external applications and, 459
unit tests and, 127
Test teams, 506–507. See also Teams
Test tools. See also Toolkits
API-layer functional, 168–170
exploratory testing, 210–211
generating test data with, 212
GUI tests, 170–176
home-brewed, 175
home-grown, 314
IDEs, 124–126
performance testing, 234–235
security testing, 225
unit-level tests, 126–127, 165–168
web service tests, 170
Test types
alpha/beta, 466–467
exploratory. See Exploratory testing (ET)
functional. See Functional testing
GUI. See GUI testing
integration, 229, 459
load. See Load testing
performance. See Performance and load testing
reliability, 230–231, 250–251
security, 220, 223–227
stress. See Load testing
unit. See Unit testing
usability. See Usability testing
user acceptance testing. See UAT (user acceptance testing)
Test writing strategy. See Strategy, for writing tests
Testability, 183–185
automated vs. manual Quadrant 2 tests, 185
automation and, 149–150
code design and test design and, 184–185
overview of, 183
of stories, 393–396
Testers
adding value, 12
agile testers, 4, 19–20
agile testing mindset, 20–21
automation allowing focus on more important work, 260
collaboration with customers, 396–397
considering all viewpoints during iteration planning, 386–389
controlling workload and, 393
definition, 507
facilitating communication, 429–430
feedback and, 486
hiring agile tester, 67–69
how to influence testing, 121–122
integration of testers into agile project, 61–63
iterations and, 327
making job easier, 114–115
sizing stories, 333–334
tester-developer ratio, 66–67
writing task cards and, 391
Tester’s bill of rights, 49–50
Testing
coding and testing simultaneously, 409–410
completing testing tasks, 415–416
identifying variations, 410
managing, 320–322
organizing test results, 322–324
organizing tests, 319–322
planning time for, 455–456
post-development cycles, 467–468
quadrants. See Quadrants
release candidates, 458
risk assessment and, 407–409
sustainable pace of, 40–41
traditional vs. agile, 12–15
transparency of tests, 321–322
Testing in context
collection-driven testing and, 106–108
definition, 502
TestNG GUI test tool, 127
Tests that never fail, 286
Text matrices, 350–353
The Grinder, 234
Themes. See also Release planning
definition, 507
prioritizing stories and, 339
writing task cards and, 392
Thin slices, incremental development and, 338
Third parties
compatibility testing and, 230
release planning and, 342–344
software, 163
Tholfsen, Mike, 203
Thomas, Mike, 116, 194
Three little pigs metaphor, 278
Timelines, scope and, 340–341
Toolkit (Quadrant 1)
build tools, 126
IDEs, 124–126
overview of, 123
source code control, 123–124
unit test tools, 126–127
Toolkit (Quadrant 2)
- API-layer functional test tools, 168–170
- automation tools, 164–165
- building tests incrementally, 178–179
- checklists, 156
- flow diagrams, 160–163
- GUI test tools, 170–176
- keep the tests passing, 179
- mind maps, 156–158
- mock-ups, 160
- software-based tools, 163
- spreadsheets, 159
- strategies for writing tests, 177–178
- test design patterns, 179–183
- test management, 186
- testability and, 183–185
- tool strategy, 153–155
- tools for eliciting examples and requirements, 155–156
- unit-level test tools, 165–168
- Web service test tool, 170

Toolkit (Quadrant 3)
- emulator tools, 213–214
- monitoring tools, 212–213
- simulator tools, 213
- user acceptance testing, 250

Toolkit (Quadrant 4)
- baselines, 235–237
- performance and load testing tools, 234–235

Tools
- API-layer functional test tools, 168–170
- automation, 164–165
- data generation, 304–305
- defect tracking, 83–85
- eliciting examples and requirements, 155–156, 159–163
- emulator tools, 213–214
- exploratory testing, 210–211
- generating test data, 212
- GUI test tools, 170–176
- home-brewed, 175
- home-grown, 314
- IDEs, 124–126
- load testing, 234–235
- monitoring, 212–213
- open source, 172, 314–315
- performance testing, 234–235
- for product owners and business experts, 134
- security testing, 225
- simulators, 213
- software-based, 163
- unit-level tests, 126–127, 165–168
- vendor/commercial, 315–316
- web service test tool, 170

Tools, automation
- agile-friendly, 316
- applying one tool at a time, 312–313
- home-brewed, 175
- home-grown, 314
- identifying tool requirements, 311–312
- open source, 314–315
- selecting, 294–298
- vendors, 315–316

Traceability
- DTS and, 82
- matrices, 86
- test planning and, 88
- Tracking, test tasks and status, 354–357

Traditional processes, transitioning. See
- Transitioning traditional processes to agile
  - Traditional teams, 9–10
- Traditional vs. agile testing, 12–15

Training
- as deliverable, 469
- lack of, 45
- Transitioning traditional processes to agile, 73–93
- defect tracking. See Defect tracking
- existing process and, 88–92
- lean measurements, 74–75
- lightweight processes and, 73–74
- metrics and, 74–79
- overview of, 73
- test planning. See Test planning

U

UAT (user acceptance testing)
- post-development testing cycles, 467–468
- product delivery and, 464–466
UAT (user acceptance testing), continued
in Quadrant 3, 102
release planning for, 331, 346
Remote Data Monitoring system example, 250
in test plan, 351
tryng out new features and, 102
writing at iteration kickoff meeting, 372
UI (user interface). See also GUI (graphical user interface)
automation strategy and, 293
modeling and, 399
Unit test tools, 165–168. See also by individual unit tools
behavior-driven development tools, 166–168
list of, 126–127
overview of, 165
Unit testing
automating, 282
BDD (Behavior-driven development), 165–168
definition, 507
metrics and, 76
supporting function of, 5
TDD (test-driven development) and, 111
technology-facing tests, 120
tools for Quadrant 1 tests, 126–127
Usability testing, 202–204
checking out applications of competitors, 204
navigation and, 204
overview of, 202
users needs and persona testing, 202–204
what should not be automated, 285–286
Use cases, 398
User acceptance testing. See UAT (user acceptance testing)
User documentation, 207–208
User interface (UI). See also GUI (graphical user interface)
automation strategy and, 293
modeling and, 399
User story. See Story
User story card. See Story card
User Stories Applied for Agile Software Development (Cohn), 155

V
Vaage, Carol, 330
Value
adding, 31–33
delivering to customer, 22–23
focusing on, 341–342
testers adding, 12
Values, agile, 3–4. See also Principles, for agile testers
Variations, coding and testing and, 410
Velocity
automation and, 255, 484
burnout rate and, 79
database impact on, 228
defects and, 487
definition, 507
maximizing, 370
sustainable pace of testing and, 41
taking time to do it right, 301
technical debt and, 106, 313, 418, 506
Vendors
automation tools, 315–316
capture-playback tool, 267
IDEs, 125
planning and, 342–344
source code control tools, 124
working with, 142, 349
Veragen, Pierre, 76, 163, 179, 295, 363, 372, 444
Version control, 123–124, 186. See also Source Code Control
Viewpoints. See also Big picture
considering all viewpoints during iteration planning, 385–389
gathering all viewpoints regarding requirements, 374–375
Power of Three and, 411
using multiple viewpoints in eliciting requirement, 137–138
Visibility, 354–366
code coverage, 360–364
communicating test results, 357–358
defect metrics, 364–366
number of passing tests, 358–360
overview of, 354
release metrics, 358
tracking test tasks and status, 354–357
Visual Studio, 125
Voris, John, 117

W
Waterfall approach, to development
agile development compared with, 12–13
“mini-waterfall” phenomenon, 46–47
successes of, 112
test plans and, 346
Watir (Web Application Testing in Ruby), 163, 172–174, 320. See also Ruby with Watir
Web Services Description Language (WSDL), 507
Web service testing
automating, 282
overview of, 207
Remote Data Monitoring system example, 247–248
tools for, 170–171
WebLoad, 234
Whelan, Declan, 321
Whiteboards
example-driven development, 379
facilitating communication, 430
modeling, 399
planning diagram, 371
reviewing high-level tests with programmers, 400–401
test plan alternatives, 353–354
Whole team approach, 325
advantages of, 26
agile vs. traditional development, 15–16
automation strategy and, 300–301
budget limits and, 55
finding enjoyment in work and, 31
key success factors, 482, 491
pairing testers with programmers, 279
shared responsibility and, 105–106
team building and, 69
team structure and, 59–62
to test automation, 270
test management and, 322
traditional cross-functional team compared with, 64
value of team members and, 70
Wiki
as communication tool, 164
graphical documentation of examples, 398–399
mockups, 160, 380
requirements, 402
story checklists and, 156
test cases, 372
traceability and, 88
Wilson-Welsh, Patrick, 278
Wizard of Oz Testing, 138–139
Workflow diagrams, 398
Working Effectively With Legacy Code (Feathers), 117, 288
Workload, 393
Worst-case scenarios, 136, 334
Writing tests, strategy for. See Strategy, for writing tests
WSDL (Web Services Description Language), 507

X
XP (Extreme Programming)
agile team embracing, 10–11
courage as core value in, 25
xUnit, 126–127

Y
Yakich, Joe, 316

Z
Zero bug tolerance, 79, 418–419