The Addison Wesley Signature Series

A Mike Cohn Signature Book

More Agile Testing

Learning Journeys for the Whole Team

Janet Gregory
Lisa Crispin

Forewords by Elisabeth Hendrickson and Johanna Rothman

Praise for *More Agile Testing*

"I love this book. It will help to create really great testers. That's a good thing, since anyone who reads this will want to have one on their team."

—Liz Keogh, agile coach, Lunivore Limited

"This book will change your thinking and move your focus from *tests* to *testing*. Yes, it is not about the result, but about the activity!"

—Kenji Hiranabe, cofounder of Astah and CEO, Change Vision, Inc.

"To my mind, agile development is about learning—that one word captures the true spirit of what agile is all about. When I had the chance to read through their new book, I could only say, 'Wow! Janet and Lisa have done themselves proud.' This is not a book about testing; this is a book about learning. Their clear explanations are accompanied by great true stories and an impressive list of books, articles, and other resources. Those of us who like learning, who love to dig for more information, can rejoice! I know you're always looking for something interesting and useful; I can guarantee that you will find it here!"

—Linda Rising, coauthor of Fearless Change: Patterns for Introducing New Ideas

"Janet and Lisa's first book, *Agile Testing*, drew some general principles that are still important today but left me wondering, 'how?' In this second book, they adapt those principles to today's development landscape—with mobile, DevOps, and cloud-based applications delivered in increasingly compressed release cycles. Readers get specific testing tools for the mind along with new practices and commentary to accelerate learning. Read it today."

—Matt Heusser, Managing Principal, Excelon Development

"An excellent guide for your team's agile journey, full of resources to help you with every kind of testing challenge you might meet along the way. Janet and Lisa share a wealth of experience with personal stories about how they helped agile teams figure out how to get value from testing. I really like how the book is filled with techniques explained by leading industry practitioners who've pioneered them in their own organizations."

—Rachel Davies, agile coach, unruly and coauthor of Agile Coaching

"Let me net this out for you: agile quality and testing is hard to get right. It's nuanced, context-based, and not for the faint of heart. In order to effectively balance it, you need hard-earned, pragmatic, real-world advice. This book has it—not only from Janet and Lisa, but also from forty additional expert agile practitioners. Get it and learn how to effectively drive quality into your agile products and across your entire organization."

—Bob Galen, Principal Consultant, R Galen Consulting Group, and Author of Agile Reflections *and* Scrum Product Ownership

“Janet and Lisa have done it again. They’ve combined pragmatic life experience with ample storytelling to help people take their agile testing to the next level.”

—*Jonathan Rasmusson, author of* Agile Samurai: How Masters Deliver Great Software

“In this sequel to their excellent first book, Janet and Lisa have embraced the maturity of agile adoption and the variety of domains in which agile approaches are now being applied. In *More Agile Testing* they have distilled the experiences of experts working in different agile organizations and combined them with their own insights into a set of invaluable lessons for agile practitioners. Structured around a range of essential areas for software professionals to consider, the book examines what we have learned about applying agile, as its popularity has grown, and about software testing in the process. There is something for everyone here, not only software testers, but individuals in any business role or domain with an interest in delivering quality in an agile context.”

—*Adam Knight, Director of QA, RainStor*

“This book has it all: practical advice and stories from the trenches. Whether you’ve never heard of agile or you think you’re an expert, there is something here that will help you out. Jump around in the book and try a few things; I promise you will be a better tester and developer for it.”

—*Samantha Laing, agile coach and trainer, Growing Agile*

“*More Agile Testing* is a great collection of stories and ideas that can help you improve not just how you test, but the products you build and the way you work. What I love most about the book is how easy many of the ideas are to try. If one message is clear, it is that regardless of your context and challenges, there are things you can try to improve. Get started today with something small, and nothing will be impossible.”

—*Karen Greaves, agile coach and trainer, Growing Agile*

“*More Agile Testing* is an extensive compilation of experiences, stories, and examples from practitioners who work with testing in agile environments around the world. It covers a broad spectrum, from organizational and hiring challenges, test techniques and practices, to automation guidance. The diversity of the content makes it a great cookbook for anyone in software development who is passionate about improving their work and wants to produce quality software.”

—*Sigurdur Birgisson, quality assistance engineer, Atlassian*

More Agile Testing

More Agile Testing

Learning Journeys for the Whole Team

Janet Gregory
Lisa Crispin

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

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

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

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data
Gregory, Janet, 1953–
More agile testing : learning journeys for the whole team / Janet Gregory, Lisa Crispin.
pages cm
Includes bibliographical references and index.
ISBN 978-0-321-96705-3 (pbk. : alk. paper)
1. Computer software—Testing. 2. Agile software development. I. Crispin, Lisa. II. Title.
QA76.76.T48G74 2015
005.1—dc23
2014027150

Illustrations by Jennifer Sinclair

ISBN-13: 978-0-321-96705-3
ISBN-10: 0-321-96705-4
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, October 2014

To my grandchildren, Lauren, Brayden, and Joe, who kept me laughing and playing throughout this past year.

—Janet

To my family, those still here and those sadly gone, and my dear friends who are part of my chosen family.

—Lisa

This page intentionally left blank

Contents

Foreword by Elisabeth Hendrickson xvii

Foreword by Johanna Rothman xix

Preface xxi

Acknowledgments xxix

About the Authors xxxiii

About the Contributors xxxv

Part I Introduction 1

Chapter 1 How Agile Testing Has Evolved 3

Summary 6

Chapter 2 The Importance of Organizational Culture 7

Investing Time 8

The Importance of a Learning Culture 12

Fostering a Learning Culture 13

Transparency and Feedback Loops 15

Educating the Organization 17

Managing Testers 19

Summary 20

Part II Learning for Better Testing 21

Chapter 3 Roles and Competencies 23
Competencies versus Roles 24
T-Shaped Skill Set 28
Generalizing Specialists 33
Hiring the Right People 36
Onboarding Testers 37
Summary 39

Chapter 4 Thinking Skills for Testing 41
Facilitating 42
Solving Problems 43
Giving and Receiving Feedback 45
Learning the Business Domain 46
Coaching and Listening Skills 48
Thinking Differently 49
Organizing 51
Collaborating 52
Summary 53

Chapter 5 Technical Awareness 55
Guiding Development with Examples 55
Automation and Coding Skills 56
General Technical Skills 59
Development Environments 59
Test Environments 60
Continuous Integration and Source Code Control Systems 62
Testing Quality Attributes 65
Test Design Techniques 67
Summary 67

Chapter 6 How to Learn 69
Learning Styles 69
Learning Resources 72
Conferences, Courses, Meet-ups, and Collaborating 72
Publications, Podcasts, and Online Communities 75
Time for Learning 77
Helping Others Learn 79
Summary 83

Part III Planning—So You Don't Forget the Big Picture 85

Chapter 7 Levels of Precision for Planning 87
Different Points of View 87
Product Release Level 89
Feature Level 92
Story Level 96
Task Level 96
Planning for Regression Testing 97
Visualize What You Are Testing 98
Summary 100

Chapter 8 Using Models to Help Plan 101
Agile Testing Quadrants 101
Planning for Quadrant 1 Testing 105
Planning for Quadrant 2 Testing 105
Planning for Quadrant 3 Testing 106
Planning for Quadrant 4 Testing 107
Challenging the Quadrants 108
Using Other Influences for Planning 113
Planning for Test Automation 115
Summary 116

Part IV Testing Business Value 119

Chapter 9 Are We Building the Right Thing? 121
Start with "Why" 121
Tools for Customer Engagement 123
Impact Mapping 123
Story Mapping 126
The 7 Product Dimensions 129
More Tools or Techniques for Exploring Early 134
Invest to Build the Right Thing 134
Summary 135

Chapter 10 The Expanding Tester's Mindset: Is This *My* Job? 137

Whose Job Is This Anyway? 137

Business Analysis Skills 137

UX Design Skills 140

Documentation Skills 141

Take the Initiative 142

Summary 144

Chapter 11 Getting Examples 145

The Power of Using Examples 145

Guiding Development with Examples 148

ATDD 149

BDD 152

SBE 153

Where to Get Examples 155

Benefits of Using Examples 157

Potential Pitfalls of Using Examples 159

Getting Bogged Down in the Details 159

Lacking Buy-in 160

Too Many Regression Tests 161

Not Enough Is Known Yet 161

The Mechanics of Using Examples to Guide Coding 162

Summary 162

Part V Investigative Testing 163

Chapter 12 Exploratory Testing 165

Creating Test Charters 168

Generating Test Charter Ideas 171

Exploring with Personas 171

Exploring with Tours 174

Other Ideas 175

Managing Test Charters 176

Session-Based Test Management 176

Thread-Based Test Management 178

Exploring in Groups 183

Recording Results for Exploratory Test Sessions 185

Where Exploratory Testing Fits into Agile Testing 188

Summary 190

Chapter 13 Other Types of Testing 191
So Many Testing Needs 192
Concurrency Testing 194
Internationalization and Localization 195
Regression Testing Challenges 200
User Acceptance Testing 201
A/B Testing 203
User Experience Testing 205
Summary 207

Part VI Test Automation 209

Chapter 14 Technical Debt in Testing 211
Make It Visible 212
Work on the Biggest Problem—and Get the Whole Team Involved 217
Summary 220

Chapter 15 Pyramids of Automation 223
The Original Pyramid 223
Alternate Forms of the Pyramid 224
The Dangers of Putting Off Test Automation 227
Using the Pyramid to Show Different Dimensions 231
Summary 235

Chapter 16 Test Automation Design Patterns and Approaches 237
Involve the Whole Team 238
Starting Off Right 239
Design Principles and Patterns 240
Testing through the API (at the Service Level) 241
Testing through the UI 243
Test Maintenance 248
Summary 251

Chapter 17 Selecting Test Automation Solutions 253
Solutions for Teams in Transition 253
Meeting New Automation Challenges with the Whole Team 258
Achieving Team Consensus for Automation Solutions 260

How Much Automation Is Enough? 262
Collaborative Solutions for Choosing Tools 264
Scaling Automation to Large Organizations 264
Other Automation Considerations 268
Summary 269

Part VII What Is Your Context? 271

Chapter 18 Agile Testing in the Enterprise 275
What Do We Mean by "Enterprise"? 275
"Scaling" Agile Testing 276
Dealing with Organizational Controls 278
Coordinating Multiple Teams 283
A System Test Team and Environment 284
Consistent Tooling 289
Coordination through CI 289
Version Control Approaches 290
Test Coverage 291
Managing Dependencies 292
Working with Third Parties 292
Involving Customers in Large Organizations 294
Advantages of Reaching Out beyond the Delivery Team 296
Summary 297

Chapter 19 Agile Testing on Distributed Teams 299
Why Not Colocate? 301
Common Challenges 302
Cultural Issues 302
Language 303
Time Zones 304
Dependencies 305
Planning 305
Strategies for Coping 308
Integrating Teams 308
Communication and Collaboration 309
Collaborating through Tests 311
Offshore Testing 312

Tool Ideas for Distributed Teams 319
Communication Tools 319
Collaboration Tools 319
Summary 322

Chapter 20 Agile Testing for Mobile and Embedded Systems 325
Similar, Yet Different 326
Testing Is Critical 328
Agile Approaches 329
Summary 337

Chapter 21 Agile Testing in Regulated Environments 339
The “Lack of Documentation” Myth 339
Agile and Compliance 340
Summary 346

Chapter 22 Agile Testing for Data Warehouses and Business Intelligence Systems 347
What Is Unique about Testing BI/DW? 348
Using Agile Principles 351
Data—the Critical Asset 352
Big Data 357
Summary 360

Chapter 23 Testing and DevOps 361
A Short Introduction to DevOps 361
DevOps and Quality 363
How Testers Add DevOps Value 371
Summary 376

Part VIII Agile Testing in Practice 379

Chapter 24 Visualize Your Testing 381
Communicating the Importance of Testing 381
Visualize for Continuous Improvement 386
Visibility into Tests and Test Results 390
Summary 392

Chapter 25 Putting It All Together 393
Confidence-Building Practices 394
Use Examples 394
Exploratory Testing 395
Feature Testing 396
Continual Learning 397
Context Sensitivity 399
Keep It Real 401
Create a Shared Vision 402
Summary 405

Appendix A Page Objects in Practice: Examples 407
An Example with Selenium 2—WebDriver 407
Using the `PageFactory` Class 410

Appendix B Provocation Starters 413

Glossary **415**

References **423**

Bibliography **435**

Index **459**

Foreword

By Elisabeth Hendrickson

Just ten years ago, agile was still considered radical. Fringe. Weird. The standard approach to delivering software involved phases: analyze, then design, then code, then test. Integration and testing happened only at the end of the cycle. The full development cycle took months or years.

If you have never worked in an organization with long cycles and discrete phases, the idea may seem a little weird now, but it was the standard a decade ago.

Back when phases were the norm and agile was still new, the agile community was mostly programmer-centric. Janet and Lisa and a few others from quality and testing were there. However, many in the agile community felt that QA had become irrelevant. They were wrong, of course. QA changed, reshaped to fit the new context, but it did not go away.

It took people like Janet and Lisa to show how QA could be integrated into agile teams instead of bypassed. Their first book together, *Agile Testing*, carefully explained the whole-team approach to quality. They covered the cultural changes needed to fully integrate testing with development. They explained how to overcome barriers. It's a fantastic book, and I highly recommend it.

However, questions remained. How could the practices be adapted to various contexts? How do you start? What should testers learn in order to be more effective?

This book picks up where *Agile Testing* left off and answers those questions and more.

Even if that were all this book did, it would be an excellent sequel.

It’s more than that, though. Within these pages you will find a theme—one that Janet and Lisa have woven so deftly throughout the text you might not even realize it as you are reading. So I am going to call your attention to it: this is a book about adapting.

Reflect-and-adapt is the one simple trick that can enable your organization to find its way to agile. Experiment, try something different, distill lessons learned, repeat. The next thing you know, your organization will be nimble and flexible, able to shift with market demands and deliver incrementally.

This book teaches you about adapting even as it is teaching you about agile testing.

Part II, “Learning for Better Testing,” isn’t just about how you learn as an individual but also about building a learning culture. Part VII, “What Is Your Context?,” isn’t just about variations in agile tailored to different situations; it’s also a field guide to various types of adaptations.

The world is changing so very quickly. Just a decade ago agile was weird; now it is mainstream. Just five years ago, tablets like iPads weren’t even on the market; now they’re everywhere. Practices, tools, technology, and markets are all changing so fast it’s hard to keep up. It’s not enough to learn one way of doing things; you need to know how to discover new ways. You need to adapt.

This book is a fantastic resource for agile testing. It will also help you learn to adapt and be comfortable with change.

I hope you enjoy it as much as I did.

Foreword

By Johanna Rothman

What do testers do? They provide information about the product under test, to expose risks for the team.

That's exactly what Janet Gregory and Lisa Crispin have done in their new book, *More Agile Testing: Learning Journeys for the Whole Team*. Do you have risks in your agility? There are plenty of ideas to help you understand the value of sustainable pace, creating a learning organization, and your role in testing.

Not sure how to test for a given product, on a single team, or in a program? There's an answer for that, too.

How do you work with people in the next cube, down the hall, and across the world? Janet and Lisa have been there and done that. Their focus on roles and not titles is particularly helpful.

There are plenty of images in this book, so you won't have to wonder, "What do they mean?" They show you, not just tell you.

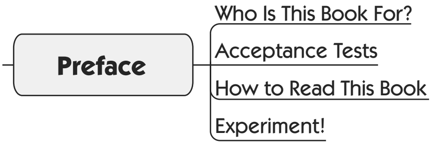
More Agile Testing: Learning Journeys for the Whole Team is much more than a book about testing. It's a book about how to use testing to help your entire team, and by extension, your organization, and transition to agile in a healthy way.

Isn't that what providing information about the organization under test, exposing risks in the organization, is all about?

If you are a tester or a test manager, you need to read this book. If you integrate testing into your organization, you need to read this book. How else will you know what the testers could be doing?

This page intentionally left blank

Preface



This book carries on where our first book, *Agile Testing: A Practical Guide for Testers and Agile Teams*, left off. We avoid repeating what we covered in our first book but give enough context so it stands alone if you have not read *Agile Testing*. We refer to the first book as *Agile Testing* when we think it might be helpful for the reader to explore basic concepts in more detail.

Who Is This Book For?

We assume that you, the reader, are not a beginner in the world of agile testing, that you have some agile and testing experience and now you're looking for help in the areas beyond where *Agile Testing* goes. If you feel that you would like an introduction to agile development that includes some basics of testing in agile before you read this book, *The Agile Samurai* (Rasmussen, 2010) is an excellent place to start.

This book is aimed at anyone who is interested in testing activities on an agile team. In our experience, this includes not only testers and test managers, but programmers, product owners, business analysts, DevOps practitioners, line managers—pretty much everyone.

Acceptance Tests

In addition to sharing what we've learned over the past several years, we wanted to make this book as useful to our readers as the first one. We wanted to know what readers of the first book still needed to know after reading it, so we asked practitioners from the *Agile Testing* mailing list to send us their "acceptance tests" for this second book. We distilled those responses to this list of acceptance tests for *More Agile Testing* and did our best to satisfy these as we wrote the book.

You'll note that we've used a style used in behavior-driven development (BDD), which we'll talk more about in Chapter 11, "Getting Examples":

> Given <precondition>,
>
> When <trigger, action>,
>
> Then <the expected result>.

- Given that I am an agile tester or manager, when I hire new testers with no agile experience, then I'll learn how to bring them up to speed and avoid throwing them into the deep end without a life jacket.
- Given that I am a team member on an agile team, when I finish this book, then I expect to know how to fit exploratory testing in with automated tests and to get a picture of the overall test coverage, without resorting to heavyweight tools.
- Given that I am an experienced agile test manager, when I finish this book, then I will understand how to approach agile testing techniques with multiple teams to allow my successful agile organization to grow.
- Given that I am an experienced agile test manager, when I finish reading this book, then I should have ideas about how to coordinate test automation activities across iterations and teams, with ideas on how to improve.
- Given that I am an experienced agile manager, when I've read this book, then I will understand how other teams have adapted agile testing practices to suit their own context and will have ideas about how to apply them to mine.
- Given that I am an agile team member who is interested in testing, when I finish this book, then I expect to have examples of what tests should and should not look like and how I can design tests effectively.

- Given that I am an experienced agile tester, when I find an interesting topic in this book about which I'd like to learn more, then I can easily find references to web resources or other books.
- Given that I am an experienced agile coach or manager who is reading the book, when I see a concept that would help my team, then I have enough information to be able to devise a strategy to get the team to try an experiment.
- Given that I am an agile team member who is concerned about testing and keeping the customers informed, when I have read this book, then I'll understand good ways to communicate with customer team members about testing activities.
- Given that I am an experienced agile test manager, when I have read this book, then I will know how mainstream adoption of agile is being done, and I will understand the working context of testers from other organizations when they apply for jobs on my team. (*Note:* This acceptance test is not part of this release, but we think some of the examples and stories in the book will help to achieve it.)

How to Read This Book

Though we've organized this book in a way that we feel flows best, you don't have to start with Chapter 1 and keep going. As with *Agile Testing*, you can begin with whatever topics are most useful to you. We try to cover each topic in detail only once, but because so many of these concepts, practices, and principles are interrelated, you'll find that we refer to some ideas in more than one chapter.

Part I: Introduction

Read this part to understand where testing started in agile teams and how it has evolved to become the cornerstone of agile development and continuous delivery of products. Part of successful agile development is an organization's ability to learn what's most critical for long-range success with agile testing.

- Chapter 1, "How Agile Testing Has Evolved"
- Chapter 2, "The Importance of Organizational Culture"

Part II: Learning for Better Testing

Both technology and the craft of testing are continually evolving, and lines between different disciplines are becoming more blurred. Even experienced practitioners have to keep growing their skills. This part includes examples of what testers and other disciplines such as business analysis and coding need to know to meet more difficult testing challenges. We explain the benefits of generalizing specialists and list some of the intangible thinking skills and specific technical testing skills that help testers and teams improve. Different aspects of what and how to learn are covered in the following chapters:

- Chapter 3, "Roles and Competencies"
- Chapter 4, "Thinking Skills for Testing"
- Chapter 5, "Technical Awareness"
- Chapter 6, "How to Learn"

Part III: Planning—So You Don't Forget the Big Picture

Planning "just enough" is a balancing act. While we need to work in small increments, we have to keep an eye on the larger feature set and the entire system. This part covers different aspects of test planning, from the release level down to the task level. It also explores different models such as the agile testing quadrants and some of the adaptations people have suggested.

- Chapter 7, "Levels of Precision for Planning"
- Chapter 8, "Using Models to Help Plan"

Part IV: Testing Business Value

If, like so many agile teams, you deliver robust code in a timely manner, only to find it isn't what the customers wanted after all, the information in this part will help. We cover tools and practices, particularly those from the agile business analysis profession, to help you test ideas and assumptions early and ensure that everyone knows what to deliver. We address other overlapping disciplines and expanding mindsets. This is a big area, so there are several chapters:

- Chapter 9, "Are We Building the Right Thing?"
- Chapter 10, "The Expanding Tester's Mindset: Is This *My* Job?"
- Chapter 11, "Getting Examples"

Part V: Investigative Testing

The programmers have delivered some code to test. Where do you start? If you or your team lacks experience with exploratory testing, you'll find some help here. We outline several exploratory testing techniques such as using personas and tours to help generate test charter ideas, as well as managing charters with session-based test management and thread-based test management.

Along with all those different ways to do exploratory testing, we look at other ways to verify that delivered code meets a wide range of business and user needs. This part covers ways to mitigate risks and generate useful information in several different types of testing that present challenges to agile teams. The investigative testing chapters are

- Chapter 12, "Exploratory Testing"
- Chapter 13, "Other Types of Testing"

Part VI: Test Automation

We see more and more teams finding ways to succeed with test automation. However, for many teams, automated tests produce sporadic failures that are expensive to investigate. The time (cost) spent on each failure may be more than the test is worth. There are plenty of pitfalls in automating tests. In this part we give examples of ways to make technical debt in testing visible. We look at different ways to use the agile testing pyramid effectively to help you think about how to plan your automation. We've introduced a few alternative pyramid models to approach automation from different perspectives. You'll learn ways to design automated tests for optimum reliability and ease of maintenance. This part also includes examples of scaling test automation in a large enterprise company.

The chapters in Part VI are

- Chapter 14, "Technical Debt in Testing"
- Chapter 15, "Pyramids of Automation"
- Chapter 16, "Test Automation Design Patterns and Approaches"
- Chapter 17, "Selecting Test Automation Solutions"

Part VII: What Is Your Context?

Your approach to agile testing will naturally depend on your context. Do you work with large enterprise systems? Maybe you're newly tasked with testing mobile apps or embedded software. Perhaps your team is challenged with finding good ways to test data that helps businesses make decisions. Have you wondered how agile can work in testing regulated software? Finally, we look at the synergies between testing and the DevOps movement. The chapters in this part cover a variety of areas, so we have included a number of stories from people who are currently working in those situations. Some of these chapters may not apply to your working environment today, but tomorrow—who knows?

- Chapter 18, "Agile Testing in the Enterprise"
- Chapter 19, "Agile Testing on Distributed Teams"
- Chapter 20, "Agile Testing for Mobile and Embedded Systems"
- Chapter 21, "Agile Testing in Regulated Environments"
- Chapter 22, "Agile Testing for Data Warehouses and Business Intelligence Systems"
- Chapter 23, "Testing and DevOps"

Part VIII: Agile Testing in Practice

We wrap up the book with a look at how teams can visualize quality and testing, and a summary of agile testing practices that will give your team confidence as you make release decisions. Creating a shared vision for your team is critical to success, and we share a model to help bring testing activities to the whole team. If you're feeling a bit overwhelmed right now and aren't sure where to start, read these chapters first:

- Chapter 24, "Visualize Your Testing"
- Chapter 25, "Putting It All Together"

The book also includes two appendixes: Appendix A, "Page Objects in Practice: Examples," and Appendix B, "Provocation Starters."

Other Elements

Since teams use such a wide variety of agile practices and approaches, we've tried to keep our terminology as generic as possible. To make sure we have a common language with you, we've included a glossary of the terms we use.

You'll find icons in the margins throughout the book where we'd like to draw your attention to a specific practice. You'll find all six icons in Chapter 1, "How Agile Testing Has Evolved," and Chapter 25, "Putting It All Together." An example of the icon for learning can be seen next to following paragraph.

We hope you'll want to learn more about some of the practices, techniques, and tools that we cover. Please check the bibliography for references to books, websites, articles, and blogs. We've sorted it by part so you can find more information easily when you're reading. Sources that are mentioned directly in the book are listed alphabetically in the reference list for easy lookup.

The mind map overview from *Agile Testing* is included on the book website, www.agiletester.com, so that you can get a feel for what was covered there if you haven't already read it.

EXPERIMENT!

Linda Rising encouraged us years ago to try small experiments, evaluate the results, and keep iterating to chip away at problems and achieve goals. If you read something in this book that sounds as if it might be useful for you or your team, give it a try for an iteration or two. Use your retrospectives to see if it's helping, and tweak as necessary. If it doesn't work, you learned something, and you can try something different.

We hope you will find many experiments to try in these pages.

Acknowledgments

This book has been a group effort. Please learn about all the wonderful practitioners who shared their stories as sidebars in "About the Contributors." Many are success stories, some describe lessons learned the hard way, but we know all will benefit you, the reader.

We're extremely grateful to Jennifer Sinclair for her wonderful illustrations. She came up with such creative ideas to help us get across some important concepts.

We referenced the ideas of so many other people who have taken ideas from *Agile Testing*, adapted them to meet their needs, and were willing to share with the world—thank you.

Our tireless reviewers helped us shape the book and cover the right topics. We're especially grateful to Mike Talks, Bernice Niel Ruhland, and Sherry Heinze, who slogged through every chapter, in some cases multiple times. Thanks to Augusto Evangelisti, Gojko Adzic, Adam Knight, Steve Rogalsky, Aldo Rall, Sharon Robson, James Lyndsay, JeanAnn Harrison, Ken Rubin, Geoff Meyer, Adam Yuret, and Mike Cohn for their valuable feedback. Each of our story contributors also helped review the chapters that included their stories.

Special thanks to our technical reviewers, whose feedback on our final draft was immensely helpful: Tom Poppendieck, Liz Keogh, Markus Gärtner, and George Dinwiddie.

Thank you, Christopher Guzikowski, for making this book possible in the first place, and Olivia Basegio, for answering a thousand questions and keeping us organized. We are grateful to our developmental editor, Chris Zahn, Kesel Wilson our production editor, and to Barbara Wood for doing the final copy edit. It was wonderful working with the Addison-Wesley crew again.

Thanks to a new English grad, Bea Paiko, who did a preliminary copy edit that helped us write a bit more cleanly. Thank you, Mike Cohn, for letting us be part of a great group of agile authors. Thanks to Ellen Gottesdiener and Mary Gorman for sharing some of their book-writing process tips with us; those helped us organize the book more easily.

We are both fortunate to have worked alongside so many amazing people over the years who taught us so much about delivering valuable software. They are too numerous to name here, but we refer to some in the text and the bibliography. We're lucky to be part of a generous global software community.

Finally, a thank-you to our wonderful, supportive family and friends.

Janet's personal thanks:
Thank you to my husband, Jack, for all the contracts reviewed, suppers prepared, and errands run, and for letting me work long into the evenings. I know I pretty much ignored you again for as long as it took to write this book. Your encouragement kept me going.

Lisa, we complement each other in our writing styles, and I think that is what makes us a great team. Thank you for providing a great place for reviewing our first draft and a chance to meet your donkeys.

And finally, I want to acknowledge the power of wireless capability and the Internet. While writing this book, I traveled north to Helsinki, Finland, and camped in Grande Prairie, Canada. I was south to Johannesburg in South Africa and camped in Botswana and Zimbabwe, writing between watching lions and elephants. As well I was in Australia, although I did not test wireless in the outback there. I even was as high as 3,000 meters (~10,000 feet) in Peru. There were only a few places where I could not connect at all. This writing was truly a distributed team effort.

Lisa's personal thanks:
Thanks to my husband, Bob Downing, without whose support I could never write or present anything. He never guessed that one day he'd be out mucking a donkey pen while I slaved over a keyboard. He has kept me and all our pets well fed and well loved. You're still the bee's knees, my dear!

Thank you, Janet, for keeping us on track and doing so much of the heavy lifting to get us organized, writing, and coming up with so many great visuals. Working with you is always a privilege, a learning experience, and a lot of fun. And I also thank Janet's husband, Jack, for his help with the fine print and for enabling Janet to share all this fun and hard work with me!

If readers learn a fraction of what I've learned while writing this book, I'll consider it a success!

This page intentionally left blank

About the Authors

Janet Gregory is an agile testing coach and process consultant with DragonFire Inc. She is coauthor with Lisa Crispin of *Agile Testing: A Practical Guide for Testers and Agile Teams* (Addison-Wesley, 2009) and *More Agile Testing: Learning Journeys for the Whole Team* (Addison-Wesley, 2015). She is also a contributor to *97 Things Every Programmer Should Know*. Janet specializes in showing agile teams how testers can add value in areas beyond critiquing the product, for example, guiding development with business-facing tests. Janet works with teams to transition to agile development and teaches agile testing courses and tutorials worldwide. She contributes articles to publications such as *Better Software*, *Software Test & Performance Magazine*, and *Agile Journal* and enjoys sharing her experiences at conferences and user group meetings around the world. For more about Janet's work and her blog, visit www.janetgregory.ca. You can also follow her on Twitter: @janetgregoryca.

Lisa Crispin is the coauthor with Janet Gregory of *Agile Testing: A Practical Guide for Testers and Agile Teams* (Addison-Wesley, 2009) and *More Agile Testing: Learning Journeys for the Whole Team* (Addison-Wesley, 2015); she is also coauthor with Tip House of *Extreme Testing* (Addison-Wesley, 2002), and a contributor to *Experiences of Test Automation* by Dorothy Graham and Mark Fewster (Addison-Wesley, 2011) and *Beautiful Testing* (O'Reilly, 2009). Lisa was honored by her peers who voted her the Most Influential Agile Testing Professional Person at Agile Testing Days 2012. Lisa enjoys working as a tester with an awesome agile team. She shares her experiences via writing, presenting, teaching, and participating in agile testing communities around the world. For more about Lisa's work, visit www.lisacrispin.com, and follow @lisacrispin on Twitter.

This page intentionally left blank

About the Contributors

Gojko Adzic is a strategic software delivery consultant who works with ambitious teams to improve the quality of their software products and processes. He specializes in agile and lean quality improvement, in particular agile testing, specification by example, and behavior-driven development. Gojko is the author of *Specification by Example* (Adzic, 2011), winner of the 2012 Jolt award; *Impact Mapping* (Adzic, 2012); *Bridging the Communication Gap* (Adzic, 2009); an award-winning blog; and other testing- and agile-related books. In 2011, he was voted by peers as the most influential agile testing professional.

Matt Barcomb is passionate about cultivating sustainably adaptive organizations, enjoys being out-of-doors, loves puns, and thrives on guiding companies toward more rewarding and productive self-organizing cultures. Matt has done this in his roles as a product development executive, organizational design consultant, agile coach, development team manager, and programmer. He believes that evolving companies to customer-focused humanistic systems is the biggest challenge facing businesses today. As such, he has dedicated an inordinate amount of his time and energy to finding ways of helping organizations become better places to work.

Susan Bligh has been in the IT industry for seventeen years and has an enthusiasm for business process and operational excellence through the use of technology. She is currently a lead business analyst at an oil and gas company in Calgary, Alberta, Canada. Susan has led business analyst efforts for large-scale projects affecting many disciplines and across broad geographies. She has previously worked in software development, training, and client management, as well as database administration.

She has a degree in computer science with a minor in management from the University of Calgary.

Paul Carvalho is dedicated to helping software development teams deliver high levels of quality with confidence. He inspires collaborative, agile, test-infected teams with a holistic approach to quality. Paul has devoted over twenty years to learning and applying testing approaches, models, methods, techniques, and tools to enlighten decision makers. He passes on that knowledge to individuals and organizations through coaching, consulting, training, writing, and speaking internationally. Paul is passionate about understanding human ecosystems for delivering great products that satisfy and delight customers, which he finds to be a natural fit with the agile community. Connect with him through STAQS.com.

Augusto Evangelisti is a software development professional, blogger, and foosball player with a great interest in people, software quality, and agile and lean practices. He enjoys cooking, eating, learning, and helping agile teams exceed customer expectations while having fun.

David Evans is an experienced agile consultant, coach, and trainer with over twenty-five years of IT experience. A thought leader in the field of agile quality, he has provided training and consultancy for clients in the UK, United States, Ireland, Sweden, Germany, France, Australia, Israel, South Africa, and Singapore. A regular speaker at events and conferences across Europe, David was voted Best Keynote Speaker at Agile Testing Days 2013. He has also had several papers published in international IT journals. He currently lives and works in the UK, where he is a partner, along with Gojko Adzic, in Neuri Consulting LLP. He can be reached at david.evans@neuri.co.uk on email and @DavidEvans66 on Twitter.

Kareem Fazal is a platform software senior development engineer in the Dell Enterprise Solutions Group. He has seven-plus years of experience in the firmware industry working on automation and product development. He joined Dell in 2010 as test lead and then transitioned into the firmware development organization to lead automation strategies and product development.

Benjamin Frempong, a senior test engineer in the Dell Enterprise Solutions Group, has over ten years of experience leading hardware and software QA programs in Dell's Client and Enterprise organizations. He is currently focused on helping teams implement efficient and sustainable test automation strategies.

Chris George has been a software tester and question asker since 1996, working for a variety of UK companies making tools for database development, data reporting, and digital content broadcasting. During that time he has explored, investigated, innovated, invented, planned, automated, stressed, reported, loaded, coded, and estimated on both traditional (waterfall) and agile software teams. He also presents at software conferences on testing topics and writes a blog, www.mostly-testing.co.uk.

Mary Gorman, a leader in business analysis and requirements, is vice president of quality and delivery at EBG Consulting. Mary coaches product teams, facilitates discovery workshops, and trains stakeholders in collaborative practices essential for defining high-value products. She speaks and writes for the agile, business analysis, and project management communities. A Certified Business Analysis Professional, Mary helped develop the IIBA's *A Guide to the Business Analysis Body of Knowledge* and certification exam. She also served on the task force that created PMI's Professional in Business Analysis role delineation. Mary is coauthor of *Discover to Deliver* (Gottesdiener and Gorman, 2012).

Ellen Gottesdiener, founder and principal of EBG Consulting, helps people discover and deliver the right software products at the right time. Ellen is an internationally recognized leader in agile product and project management practices, product envisioning and roadmapping, business analysis and requirements, retrospectives, and collaboration. As an expert facilitator, coach, and trainer, Ellen works with clients around the world and speaks frequently at a diverse range of industry conferences. She is coauthor of *Discover to Deliver* (Gottesdiener and Gorman, 2012) and author of two other acclaimed books: *Requirements by Collaboration* (Gottesdiener, 2002) and *The Software Requirements Memory Jogger* (Gottesdiener, 2005).

Jon Hagar is an independent consultant working in software product integrity, verification, and validation testing at Grand Software Testing. Jon publishes regularly, including a book on mobile/embedded software testing: *Software Test Attacks to Break Mobile and Embedded Devices* (Hagar, 2013). His interests include agile, mobile, embedded, QA, skill building, and lifelong learning.

Parimala Hariprasad spent her youth studying people and philosophy. By the time she got to work, she was able to use those learnings to create awesome testers. She has worked as a tester for over ten years for domains such as customer relationship management, security, e-commerce, and health care. Her specialty is coaching and creating great teams—teams that ultimately fired her because she wasn't needed anymore. She has experienced the transition from web to mobile and emphasizes the need for design thinking in testing. She frequently rants on her blog, Curious Tester (http://curioustester.blogspot.com). She tweets at @CuriousTester and can be found on LinkedIn at http://in.linkedin.com/in/parimalahariprasad.

JeanAnn Harrison has been in the software testing and quality assurance field for over fifteen years, including seven years working within a regulatory environment and eight years performing mobile software testing. Her niche is system integration testing with a focus on multitiered system environments involving client/server, web application, and stand-alone software applications. JeanAnn is a regular speaker at many software testing conferences and other events and is a Weekend Testing Americas facilitator. She is always looking to gain inspiration from fellow testers throughout the software testing community and continues to combine her practical experiences with interacting on software quality and testing forums, attending training classes, and remaining active on social media sites.

Mike Heinrich has been working as a tester for over a decade, working in logistics, banking, telecommunications, travel, and utilities. Throughout his career, Mike has focused on data and integration testing. His passion for data and delivering customer value has afforded him the opportunity to present to a number of North American organizations on agile data warehousing and data testing. In his free time, Mike enjoys traveling the world, playing volleyball, and coaching basketball.

Sherry Heinze is a test strategist, tester, QA analyst, and trainer with a broad background in analysis, design, testing, training, implementation, documentation, and user support. For the last 17 years, Sherry has focused on testing from analysis and design forward, sometimes on cross-functional teams, sometimes with teams of testers, sometimes alone. Sherry has extensive experience working in various methodologies with both users and technical staff to identify and test requirements, design, create, test, implement, and support systems.

Matthew Heusser has spent his adult life developing, testing, and managing software projects. Along the way Matt served as a contributing editor for *Software Test & Quality Assurance* magazine, organized the Agile Alliance Sponsored Workshop on Technical Debt, and served on the board of directors for the Association for Software Testing. Perhaps best known for his writing, Matt was the lead editor for *How to Reduce the Cost of Software Testing* (Heusser, 2011) and is currently serving as managing editor for Stickyminds.com. As the managing consultant at Excelon Development, Matt manages key accounts for the company while also doing consulting and writing. You can read more about Matt at the Excelon website, www.xndev.com, or follow him on Twitter: @ heusser.

Michael Hüttermann, a Java champion, is a freelance delivery engineer and expert for DevOps, continuous delivery, and source control management/application life cycle management. He is the author of *Agile ALM* (Hüttermann, 2011a) and *DevOps for Developers* (Hüttermann, 2012). For more information see http://huettermann.net.

Griffin Jones, an agile tester, trainer, and coach, provides consulting on context-driven software testing and regulatory compliance to companies in regulated and unregulated industries. Recently, he was the director of quality and regulatory compliance at iCardiac Technologies, which provides core lab services for the pharmaceutical industry to evaluate the cardiac safety of potential new drugs. Griffin was responsible for all matters relating to quality and FDA regulatory compliance, including presenting the verification and validation (testing) results to external regulatory auditors. He is a host of the Workshop on Regulated Software Testing (WREST) and a member of ASQ, AST, ISST, and RAPS.

Stephan Kämper studied physics, wrote his diploma thesis about holography, and then joined the oceanography group at the University of Bremen. In 2001 he started in software development by joining the test team for an object-oriented database system. He never left software testing and specialized in automated software tests and agile methods. He worked on topics as diverse as precision navigation systems, payment platforms, health care systems, telecommunication, and social networks. Working in these different fields helped him recognize common patterns, which he found useful in software testing. His languages are (in alphabetical order) English, German, and Ruby. Follow him on Twitter at @S_2K, and see his website: www.seasidetesting.com.

Trish Khoo has worked in test engineering and test management for companies such as Google, Campaign Monitor, and Microsoft. She maintains a blog at www.trishkhoo.com and a podcast at testcast.net, enjoys speaking at conferences, and writes articles for technical publications. When she's not doing all of that, she's busy traveling the world, sketching robots, or maybe just sleeping until noon. Trish earned a bachelor's degree in information technology from the University of Queensland, where she graduated with honors.

Adam Knight has been testing data storage and analysis software for ten years, with seven of those spent working in an agile team. Adam is an enthusiastic exponent of exploratory testing approaches backed by discerning use of automation. He is a great believer in creating multi-skilled teams based on rich and unique individual skill sets. At his current employer, RainStor, Adam has overseen the testing and technical support of a large-scale data storage system from its initial release through successful adoption in some of the largest telecommunication and financial services companies in the world. He writes at www.a-sisyphean-task.com.

Cory Maksymchuk is a software developer who is passionate about agile processes and lean software development. He has spent most of the last 12 years working in the Java stack as part of large software development initiatives. His true passion in life is finding elegant solutions to difficult problems, and he truly gets excited about seeing great ideas come to life.

Drew McKinney is a user experience designer with Pivotal Tracker and Pivotal Labs. Before Pivotal, Drew ran Bloomingsoft, a mobile design and development consultancy. In the past Drew has worked with companies such as Disney Animation Studios, Audi USA, Cook Medical, and Deloitte Consulting. He is an active member of the design community and has spoken about design at numerous Indiana and Colorado technology events.

Geoff Meyer, a test architect in the Dell Enterprise Solutions Group, has over twenty-eight years of software industry experience as a developer, manager, business analyst, and test architect. Since 2010, a secondary focus of Geoff's has been fostering the agile-based software development and test practices of more than eight hundred development, test, and user experience engineers across four global design centers. Geoff is an active member and contributor to the Agile Austin community.

Jeff "Cheezy" Morgan, chief technology officer and a cofounder of LeanDog, has been teaching classes and coaching teams on agile and lean techniques since early 2004. Most of his work has focused on the engineering practices used by developers and testers. For the past few years he has experienced great success and recognition for his work focused on helping teams adopt acceptance-test-driven development using Cucumber. He has authored several popular Ruby gems used by software testers and is the author of the book *Cucumber & Cheese—A Testers Workshop* (Morgan, 2013).

Claire Moss became the first discrete mathematics business graduate from the Georgia Institute of Technology in 2003 and immediately jumped into software testing. She has been following this calling ever since, working with agile product teams as a testing teacher, unit and integration test adviser, exploratory tester, and test automator. Although she'll always go back to scrapbooking, her dominant hobby in recent years has been writing, speaking, and nerding about testing. Claire has always had a passion for writing, and she continues to use her evil powers for good on the job and on her blog at http://aclairefication.com.

Aldo Rall started off in testing as a junior programmer at the start of the Y2K bubble. Since then, working in South Africa and the UK, he gained practical experience in testing across a plethora of titles, assignments, and projects. His greatest passion lies in the "people" dimension and how that translates into successful products, teams, and testers. Through this background, he enjoys opportunities to develop, grow, and mature testing, testers, and teams.

Sharon Robson is the software testing practice lead for Software Education. A passionate tester and a natural-born trainer, Sharon delivers and develops courses at all levels of software testing from introductory to advanced. Sharon also focuses on agile and spends a significant amount of her time working with teams (training, coaching, and mentoring) to assist them in their transitions. Sharon is currently researching and writing about agile test approaches in various business domains. She presents at both local and international conferences and contributes to the testing and agile community via blogs, tweets, conference involvement, and mentoring.

Steve Rogalsky, recognizing that software development culture, management, and process can be frustrating and inhibiting, has invested significantly in finding ways to overcome and counteract those effects. He's found that valuing simplicity, respect for people, continuous improvement, and short feedback loops are powerful tools for addressing these shortcomings. Since software development doesn't own those frustrations, he's also been translating what he's learned into other areas of the organization, family life, community groups, and coaching. He speaks regularly at conferences in Canada and the United States, has been featured on InfoQ, cofounded the Winnipeg Agile User Group, and works at Protegra. You can read more about what he's learned at http://WinnipegAgilist.Blogspot.com.

Bernice Niel Ruhland, with over twenty years of professional experience encompassing a variety of technical disciplines, currently serves as the director of quality management programs for ValueCentric LLC. Applying her proficiencies in software programming, testing, assessment, and implementation, Bernice leads the company's software testing department. As the driving force behind ValueCentric's company-wide quality programs, she draws upon practices in the context-driven and agile theories and methodologies to guide foundational efforts. When not working, she maintains a successful blog, www.TheTestersEdge.com, a collection of her observations related to a variety of technical topics including software testing, leadership, and career development.

Huib Schoots is a tester, consultant, and people lover. He shares his passion for testing through coaching, training, and giving presentations on a variety of test subjects. Curious and passionate, he is an agile and context-driven tester who attempts to read everything ever published on software testing. He's also a member of TestNet, AST, and ISST; a black belt in the Miagi-Do School of Software Testing; and coauthor of a book about the future of software testing. Huib maintains a blog on www.magnificant.com and tweets as @huibschoots.

Paul Shannon and **Chris O'Dell** joined the 7digital team in 2010 and 2011 respectively, both starting in the team responsible for the 7digital API. They worked on improving the quality of the testing in the API, and Chris now leads that team, concentrating on improving the platform for continuous delivery, resilience, and scaling. Paul works across all teams in the 7digital technology team that are geared toward continuous improvement and quality-driven software development practices. The team follows a test-first approach with a highly collaborative and visible workflow, and all absolutely love technology and testing.

Jennifer Sinclair has been an artist, art instructor, and educator since 1995. During that time, she has lived in Canada, Japan, and the United States and has worked to improve art exploration for children and adults of all ages and abilities. She has designed and illustrated images for the Alberta Teachers' Association Early Childhood Education Council and the Alberta Education Council in Canada. She is currently working on developing art lessons that integrate easily into the core subjects of elementary education. As a homemaker, freelance artist, and volunteer art instructor, she is passionate about continuing to develop her skills and knowledge and share them with as many people as possible. You can reach her at jvaagesinclair@live.com.

Toby Sinclair joined the software testing business as a university graduate in 2007 and hasn't looked back. He has worked for various software testing consultancies in the UK and is currently working with J. P. Morgan to advance its testing capabilities to support the transition to agile. Toby is an active member of the testing community and can be found on Twitter: @TobyTheTester.

Tony Sweets is a 20-year veteran of the software industry, currently working as an information technology architect. For the past 13 years he has been working on Java enterprise web applications in the financial sector. Tony possesses a wide range of skills but likes to work mostly on Java applications and the tools that make the development process better. Tony holds a bachelor's degree in computer science from the University of Wyoming.

Mike Talks was 26 when he first gave IT "a go" as a career. Before that, he'd been a teacher, research scientist, and data analyst, and his parents worried he'd never "get a proper job." Although originally a programmer, it was in testing where he flourished. He originally worked on long, requirements-rich military waterfall projects in the UK, but since moving to New Zealand he's found himself increasingly working on projects with companies such as Assurity, Kiwibank, and Datacom, where timely delivery is a key factor.

Eveliina Vuolli acts currently as operational development manager in Nokia Solutions and Networks. She has been working with the network management system R&D development team for 15 years, acting in different kinds of roles in the global, multinational organization: integration and verification process owner, project manager and trainer in various areas, and also coach. In addition, she has been involved in the agile transformation in her own product area.

Pete Walen has been in software development for over twenty-five years. He has worked in a variety of roles including developer, business analyst, and project manager. He is an independent consulting contractor who works with test teams over extended periods, coaching them and working to improve their testing techniques and practices. Pete describes himself as a "Software Anthropologist and Tester," which encompasses the examination of how software and people relate and interact. He has worked in a variety of shops using a variety of development methodologies and has adopted an attitude of "do what makes sense" for the organization and the project.

Mary Walshe helps teams deliver successful solutions to business problems and plays a major role in striving for a kaizen culture in these teams. Mary was the driving force behind the introduction of acceptance-test-driven development in her department. She has been working in the industry for four years, and currently she works on a team in Paddy Power as an agile tester. Her team is using kanban to help them measure their experiments and in order to continually improve. In her spare time Mary runs adventure races, mountain bikes, and just recently found a new love for skiing.

Christin Wiedemann, after finishing her Ph.D. in physics at Stockholm University in 2007, began working as a software developer. Christin soon discovered that she found testing to be more challenging and creative, and she joined the testing company AddQ Consulting. There, she worked as a tester, test lead, and trainer, giving courses on agile testing, test design, and exploratory testing. In late 2011, Christin moved to Vancouver, Canada, joining Professional Quality Assurance. In her roles as tester, test lead, trainer, and speaker, Christin uses her scientific background and pedagogic abilities to continually develop her own skills and those of others.

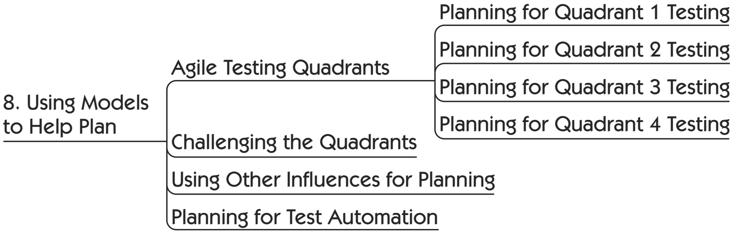
Lynn Winterboer, with a proven background in a variety of data projects and agile practices, teaches and coaches data warehouse/business intelligence teams on how to effectively apply agile principles and practices to their work. For more than fifteen years, Lynn has served in numerous roles within the analytics, business intelligence, and data warehousing space. She very well understands the unique set of challenges faced by teams in this area that want to benefit from the incremental style of agile development; Lynn leverages her experience and training to help deliver practical solutions for her clients and students. Lynn can be reached at www.LynnWinterboer.com.

Cirilo Wortel is an independent tester and trainer from the Netherlands. In 2006 Cirilo first got involved in agile software development. He has worked with several enterprise companies, coaching and helping to implement test automation during their agile adoption. Cirilo cohosted, with Janet Gregory, a master class in agile testing for several years in the Netherlands. He has contributed back to the community by founding the Federation of Agile Testers, the largest agile testing user group in the Netherlands, and is a frequent speaker at international conferences. With several colleagues at Xebia, Cirilo developed Xebium, an automation tool for web applications.

Alexei Zheglov is dedicated to discovering and practicing new methods of managing and leading the improvement of modern, complex, knowledge-intensive work. He came to this after a long software engineering career, during which he learned to see and to solve many problems in software delivery. Alexei presents his findings frequently at conferences in Canada and abroad. He is recognized as a Kanban Coaching Professional and an Accredited Kanban Trainer. Alexei lives in Waterloo, Ontario, Canada. His blog can be found at http://connected-knowledge.com.

Chapter 8

Using Models to Help Plan


8. Using Models to Help Plan
Agile Testing Quadrants
Planning for Quadrant 1 Testing
Planning for Quadrant 2 Testing
Planning for Quadrant 3 Testing
Planning for Quadrant 4 Testing
Challenging the Quadrants
Using Other Influences for Planning
Planning for Test Automation

As agile development becomes increasingly mainstream, there are established techniques that experienced practitioners use to help plan testing activities in agile projects, although less experienced teams sometimes misunderstand or misuse these useful approaches. Also, the advances in test tools and frameworks have somewhat altered the original models that applied back in the early 2000s. Models help us view testing from different perspectives. Let's look at some foundations of agile test planning and how they are evolving.

Agile Testing Quadrants

The agile testing quadrants (the Quadrants) are based on a matrix Brian Marick developed in 2003 to describe types of tests used in Extreme Programming (XP) projects (Marick, 2003). We've found the Quadrants to be quite handy over the years as we plan at different levels of precision. Some people have misunderstood the purpose of the Quadrants. For example, they may see them as sequential activities instead of a taxonomy of testing types. Other people disagree about which testing activities belong in which quadrant and avoid using the Quadrants altogether. We'd like to clear up these misconceptions.

Figure 8-1 is the picture we currently use to explain this model. You'll notice we've changed some of the wording since we presented it in *Agile*

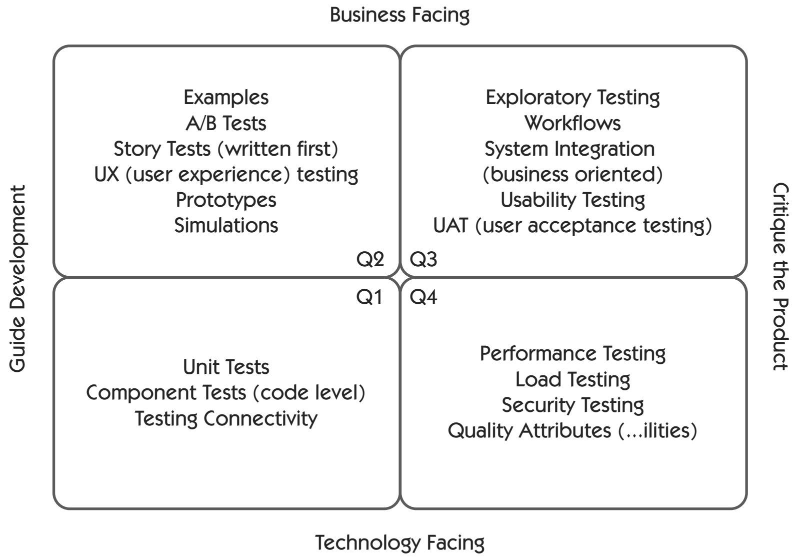


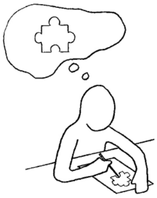
Figure 8-1 Agile testing quadrants

Testing. For example, we now say "guide development" instead of "support development." We hope this makes it clearer.

It's important to understand the purpose behind the Quadrants and the terminology used to convey their concepts. The quadrant numbering system does *not* imply any order. You don't work through the quadrants from 1 to 4, in a sequential manner. It's an arbitrary numbering system so that when we talk about the Quadrants, we can say "Q1" instead of "technology-facing tests that guide development." The quadrants are

- **Q1:** technology-facing tests that guide development
- **Q2:** business-facing tests that guide development
- **Q3:** business-facing tests that critique (evaluate) the product
- **Q4:** technology-facing tests that critique (evaluate) the product

The left side of the quadrant matrix is about preventing defects before and during coding. The right side is about finding defects and discovering missing features, but with the understanding that we want to find them as fast as possible. The top half is about exposing tests to the business, and the bottom half is about tests that are more internal to the team but equally important to the success of the software product. “Facing” simply refers to the language of the tests—for example, performance tests satisfy a business need, but the business would not be able to read the tests; they are concerned with the results.



Most agile teams would start with specifying Q2 tests, because those are where you get the examples that turn into specifications and tests that guide coding. In his 2003 blog posts about the matrix, Brian called Q2 and Q1 tests “checked examples.” He had originally called them “guiding” or “coaching” examples and credits Ward Cunningham for the adjective “checked.” Team members would construct an example of what the code needs to do, check that it doesn’t do it yet, make the code do it, and check that the example is now true (Marick, 2003). We include prototypes and simulations in Q2 because they are small experiments to help us understand an idea or concept.

In some cases it makes more sense to start testing for a new feature using tests from a different quadrant. Lisa has worked on projects where her team used performance tests for a spike for determination of the architecture, because that was the most important quality attribute for the feature. Those tests fall into Q4. If your customers are uncertain about their requirements, you might even do an investigation story and start with exploratory testing (Q3). Consider where the highest risk might be and where testing can add the most value.

Most teams concurrently use testing techniques from all of the quadrants, working in small increments. Write a test (or check) for a small chunk of a story, write the code, and once the test is passing, perhaps automate more tests for it. Once the tests (automated checks) are passing, use exploratory testing to see what was missed. Perform security or load testing, and then add the next small chunk and go through the whole process again.

Michael Hüttermann adds “outside-in, barrier-free, collaborative” to the middle of the quadrants (see Figure 8-2). He uses behavior-driven development (BDD) as an example of barrier-free testing. These tests are written in a natural, ubiquitous “given_when_then” language that’s accessible to customers as well as developers and invites conversation

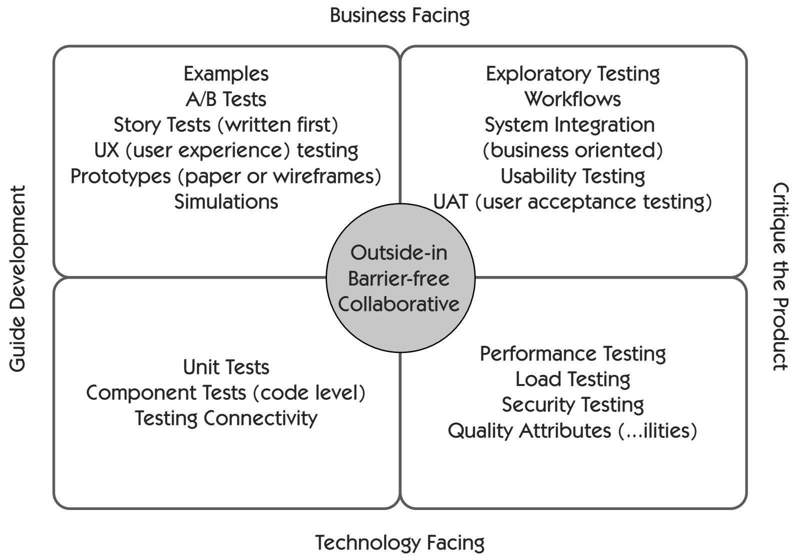


Figure 8-2 Agile testing quadrants (with Michael Hüttermann's adaptation)

between the business and the delivery team. This format can be used for both Q1 and Q2 checking. See Michael's *Agile Record* article (Hüttermann, 2011b) or his book *Agile ALM* (Hüttermann, 2011a) for more ideas on how to augment the Quadrants.

The Quadrants are merely a taxonomy or model to help teams plan their testing and make sure they have all the resources they need to accomplish it. There are no hard-and-fast rules about what goes in which quadrant. Adapt the Quadrants model to show what tests your team needs to consider. Make the testing visible so that your team thinks about testing first as you do your release, feature, and story planning. This visibility exposes the types of tests that are currently being done and the number of people involved. Use it to provoke discussions about testing and which areas you may want to spend more time on.

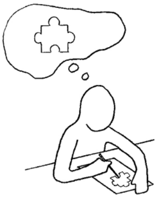
When discussing the Quadrants, you may realize there are necessary tests your team hasn't considered or that you lack certain skills or resources to be able to do all the necessary testing. For example, a team that Lisa worked on realized that they were so focused on turning business-facing examples into Q2 tests that guide development that they were completely ignoring the need to do performance and security testing. They added in user stories to research what training and tools they would need and then budgeted time to do those Q4 tests.

Planning for Quadrant 1 Testing

Back in the early 1990s, Lisa worked on a waterfall team whose programmers were required to write unit test plans. Unit test plans were definitely overkill, but thinking about the unit tests early and automating all of them were a big part of the reason that critical bugs were never called in to the support center. Agile teams don't plan Q1 tests separately. In test-driven development (TDD), also called test-driven design, testing is an inseparable part of coding. A programmer pair might sit and discuss some of the tests they want to write, but the details evolve as the code evolves. These unit tests guide development but also support the team in the sense that a programmer runs them prior to checking in his or her code, and they are run in the CI on every single check-in of code.

There are other types of technical testing that may be considered as guiding development. They might not be obvious, but they can be critical to keeping the process working. For example, let's say you can't do your testing because there is a problem with connectivity. Create a test script that can be run before your smoke test to make sure that there are no technical issues. Another test programmers might write is one to check the default configuration. Many times these issues aren't known until you start deploying and testing.

Planning for Quadrant 2 Testing



Q2 tests help with planning at the feature or story level. Part IV, "Testing Business Value," will explore guiding development with more detailed business-facing tests. These tests or checked examples are derived from collaboration and conversations about what is important to the feature or story. Having the right people in a room to answer questions and give specific examples helps us plan the tests we need. Think about the levels of precision discussed in the preceding chapter; the questions and the examples get more precise as we get into details about the stories. The process of eliciting examples and creating tests from them fosters collaboration across roles and may identify defects in the form of hidden assumptions or misunderstandings before any code is written.

Show everyone, even the business owners, what you plan to test; see if you're standing on anything sacred, or if they're worried you're missing something that has value to them.

Creating Q2 tests doesn't stop when coding begins. Lisa's teams have found it works well to start with happy path tests. As coding gets under way and the happy path tests start passing, testers and programmers flesh out the tests to encompass boundary conditions, negative tests, edge cases, and more complicated scenarios.

Planning for Quadrant 3 Testing

Testing has always been central to agile development, and guiding development with customer-facing Q2 tests caught on early with agile teams. As agile teams have matured, they've also embraced Q3 testing, exploratory testing in particular. More teams are hiring expert exploratory testing practitioners, and testers on agile teams are spending time expanding their exploratory skills.

Planning for Q3 tests can be a challenge. We can start defining test charters before there is completed code to explore. As Elisabeth Hendrickson explains in her book *Explore It!* (Hendrickson, 2013), charters let us define where to explore, what resources to bring with us, and what information we hope to find. To be effective, some exploratory testing might require completion of multiple small user stories, or waiting until the feature is complete. You may also need to budget time to create the user personas that you might need for testing, although these may already have been created in story-mapping or other feature-planning exercises. Defining exploratory testing charters is not always easy, but it is a great way to share testing ideas with the team and to be able to track what testing was completed. We will give examples of such charters in Chapter 12, "Exploratory Testing," where we discuss different exploratory testing techniques.

One strategy to build in time for exploratory testing is writing stories to explore different areas of a feature or different personas. Another strategy, which Janet prefers, is having a task for exploratory testing for each story, as well as one or more for testing the feature. If your team uses a definition of "done," conducting adequate exploratory testing might be part of that. You can size individual stories with the assumption that you'll spend a significant amount of time doing exploratory testing. Be aware that unless time is specifically allocated during task creation, exploratory testing often gets ignored.

Q3 also includes user acceptance testing (UAT). Planning for UAT needs to happen during release planning or as soon as possible. Include your customers in the planning to decide the best way to proceed. Can they come into the office to test each new feature? Perhaps they are in a different country and you need to arrange computer sharing. Work to get the most frequent and fastest feedback possible from all of your stakeholders.

Planning for Quadrant 4 Testing

Quadrant 4 tests may be the easiest to overlook in planning, and many teams tend to focus on tests to guide development. Quadrant 3 activities such as UAT and exploratory testing may be easier to visualize and are often more familiar to most testers than Quadrant 4 tests. For example, more teams need to support their application globally, so testing in the internationalization and localization space has become important. Agile teams have struggled with how to do this; we include some ideas in Chapter 13, "Other Types of Testing."

Some teams talk about quality attributes with acceptance criteria on each story of a feature. We prefer to use the word *constraints*. In *Discover to Deliver* (Gottesdiener and Gorman, 2012), Ellen Gottesdiener and Mary Gorman recommend using Tom and Kai Gilb's Planguage (their planning language; see the bibliography for Part III, "Planning—So You Don't Forget the Big Picture," for links) to talk about these constraints in a very definite way (Gilb, 2013).

If your product has a constraint such as "Every screen must respond in less than three seconds," that criterion doesn't need to be repeated for every single story. Find a mechanism to remind your team when you are discussing the story that this constraint needs to be built in and must be tested. Liz Keogh describes a technique to write tests about how capabilities such as system performance can be monitored (Keogh, 2014a). Organizations usually know which operating systems or browsers they are supporting at the beginning of a release, so add them as constraints and include them in your testing estimations. These types of quality attributes are often good candidates for testing at a feature level, but if it makes sense to test them at the story level, do so there; think, "Test early." Chapter 13, "Other Types of Testing," will cover a few different testing types that you may have been struggling with.

Challenging the Quadrants

Over the years, many people have challenged the validity of the Quadrants or adjusted them slightly to be more meaningful to them. We decided to share a couple of these stories because we think it is valuable to continuously challenge what we "know" to be true. That is how we learn and evolve to improve and meet changing demands.

Gojko's Challenge to the Quadrants

> **Gojko Adzic**, *an author and strategic software delivery consultant, challenges the validity of the Quadrants in the current software delivery era.*

The agile testing quadrants model is probably the one thing that everyone remembers about the original *Agile Testing* book. It was an incredibly useful thinking tool for the software delivery world then—2008. It helped me facilitate many useful discussions on the big picture missing from typical programmers' view of quality, and it helped many testers figure out what to focus on. The world now, as of 2014, looks significantly different. There has been a surge in the popularity of continuous delivery, DevOps, Big Data analytics, lean startup delivery, and exploratory testing. The Quadrants model is due for a serious update.

One of the problems with the original Quadrants model is that it was easily misunderstood as a sequence of test types—especially that there is some kind of division between things before and things after development.

This problem is even worse now than in 2008. With the surge in popularity of continuous delivery, the dividing line is getting more blurred and is disappearing. With shorter iterations and continuous delivery, it's generally difficult to draw the line between activities that support the team and those that critique the product. Why would performance tests not be aimed at supporting the team? Why are functional tests not critiquing the product? Why is UAT separate from functional testing? I always found the horizontal dimension of the Quadrants difficult to justify, because critiquing the product can support the team quite effectively if it is done in a timely way. For example, specification by example helps teams to completely merge functional tests and UAT into something that is continuously checked during development. Many teams I worked with recently run performance tests during development, primarily not to mess things up with frequent changes. These are just two examples where things on the right side of the Quadrants are now used more to support the team than anything else. With lean startup methods, products get a lot of critiquing even before a single line of production code is written.

Dividing tests into those that support development and those that evaluate the product does not really help to facilitate useful discussions anymore, so we need a different model—in particular, one that helps to address the eternal issue of so-called nonfunctional requirements, which for many people actually means, “It’s going to be a difficult discussion, so let’s not have it.” The old Quadrants model puts “ilities” into a largely forgotten quadrant of technical tests after development. But things like security, performance, scalability, and so on are not really technical; they imply quite a lot of business expectations, such as compliance, meeting service-levels agreements, handling expected peak loads, and so on. They are also not really nonfunctional, as they imply quite a lot of functionality such as encryption, caching, and work distribution. This of course is complicated by the fact that some expectations in those areas are not that easy to define or test for—especially the unknown unknowns. If we treat these as purely technical concerns, the business expectations are often not explicitly stated or verified. Instead of nonfunctional, these concerns are often dysfunctional. And although many “ilities” are difficult to prove before the software is actually in contact with its real users, the emergence of A/B split testing techniques over the last five years has made it relatively easy, cheap, and low risk to verify those things in production.

Another aspect of testing not really captured well by the first book’s Quadrants is the surge in popularity and importance of exploratory testing. In the old model, exploratory testing is something that happens from the business perspective in order to evaluate the product (often misunderstood as after development). In many contexts, well documented in Elisabeth Hendrickson’s book on exploratory testing (Hendrickson, 2013) and James Whittaker’s book *How Google Tests Software* (Whittaker et al., 2012), exploratory testing can be incredibly useful for the technical perspective as well and, more importantly, is something that should be done during development.

The third aspect that is not captured well by the early Quadrants is the possibility to quantify and measure software changes through usage analytics in production. The surge in popularity of Big Data analytics, especially combined with lean startup and continuous delivery models, enables teams to test relatively cheaply things that were very expensive to test ten years ago—for example, true performance impacts. When the original *Agile Testing* book came out, serious performance testing often meant having a complete hardware copy of the production system. These days, many teams de-risk those issues with smaller, less risky continuous changes, whose impact is measured directly on a subset of the production environment. Many teams also look at their production log trends to spot unexpected and previously unknown problems quickly.

We need to change the model (Figure 8-3) to facilitate all those discussions, and I think that the current horizontal division isn't helping anymore. The context-driven testing community argues very forcefully that looking for expected results isn't really testing; instead, they call that checking. Without getting into an argument about what is or isn't testing, I found the division to be quite useful for many recent discussions with clients. Perhaps that is a more useful second axis for the model: the difference between looking for expected outcomes and analyzing unknowns, aspects without a definite yes/no answer, where results require skillful analytic interpretation. Most of the innovation these days seems to happen in the second part anyway. Checking for expected results, from both a technical and business perspective, is now pretty much a solved problem.

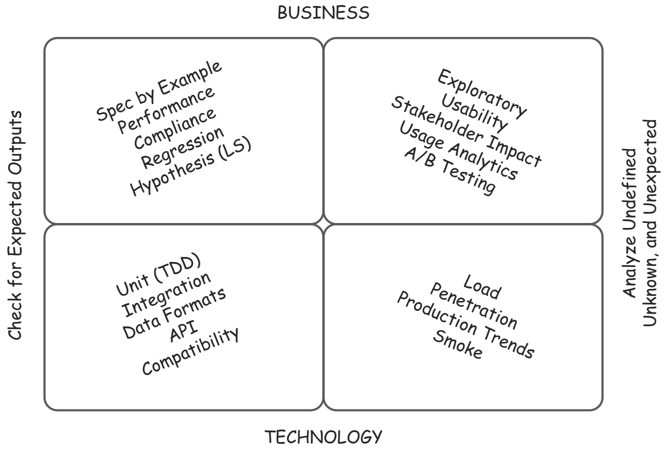


Figure 8-3 Gojko Adzic's version of the agile testing quadrants

Thinking about checking expected outcomes versus analyzing outcomes that weren't predefined helps to explain several important issues facing software delivery teams today:

Security concerns could be split easily into functional tests for compliance such as encryption, data protection, authentication, and so on (essentially all checking for predefined expected results), and penetration/investigations (not predefined). This will help to engage the delivery team and business sponsors in a more useful discussion about describing the functional part of security up front.

Performance concerns could be divided into running business scenarios to prove agreed-upon service levels and capacity, continuous delivery style (predefined), and load tests (where will it break?). This will help to engage the delivery team and business in defining performance expectations and prevent people from treating performance as a purely technical concern. By avoiding the support the team/evaluate the product divisions, we allow a discussion of executing performance tests in different environments and at different times.

Exploration would become much more visible and could be clearly divided between technical and business-oriented exploratory tests. This can support a discussion of technical exploratory tests that developers should perform or that testers can execute by reusing existing automation frameworks. It can also support an overall discussion of what should go into business-oriented exploratory tests.

Build-measure-learn product tests would fit into the model nicely, and the model would facilitate a meaningful discussion of how those tests require a defined hypothesis and how that is different from just pushing things out to see what happens through usage analytics.

We can facilitate a conversation on how to spot unknown problems by monitoring production logs as a way of continuously testing technical concerns that are difficult to check and expensive to automate before deployment, but still useful to support the team. By moving the discussion away from supporting development or evaluating the product toward checking expectations or inspecting the unknown, we would also have a nice way of differentiating those tests from business-oriented production usage analytics.

Most importantly, by using a different horizontal axis, we can raise awareness about a whole category of things that don't fit into typical test plans or test reports but are still incredibly valuable. The early Quadrants were useful because they raised awareness about a whole category of things in the upper-left corner that most teams weren't really thinking of but are now taken as common sense. The 2010s Quadrants need to help us raise awareness about some more important issues for today.

	✓ CONFIRM	INVESTIGATE
BUSINESS	BUSINESS-FACING EXPECTATIONS	RISKS TO EXTERNAL QUALITY ATTRIBUTES
TECHNOLOGY	TECHNOLOGY-FACING EXPECTATIONS	RISKS TO INTERNAL QUALITY ATTRIBUTES

Figure 8-4 Elisabeth Hendrickson's version of the agile testing quadrants

Elisabeth Hendrickson also presented an alternative to the existing Quadrants in her talk about "The Thinking Tester" (Hendrickson, 2012). It is similar to Gojko's version but has a different look. You can see in Figure 8-4 that she relabeled the vertical columns to "confirm" and "investigate," while the horizontal rows still represent business and technology.

The top left quadrant represents the expectations of the business, which could be in the form of executable (automated) specifications. Others might be represented by paper prototypes or wireframes. At the top right are tests that help investigate risks concerning the external quality of the product. It is very much like the original quadrant's idea of exploratory testing, scenarios, or usability testing. Like Gojko's model, the bottom right quadrant highlights the risks of the internal working of the system.

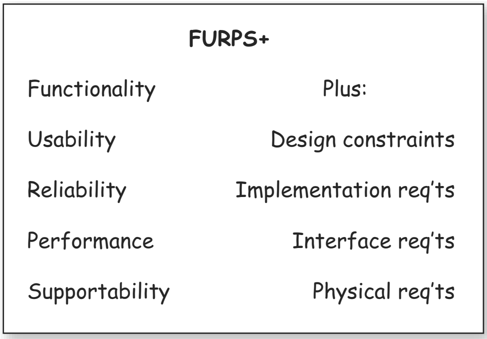
Both of these alternative models provide value. We think there is room for multiple variations to accommodate a spectrum of needs. For example, organizations that are able to adopt continuous delivery are able to think in this space, but many organizations are years from accomplishing that. Check the bibliography for Part III for links to additional testing quadrant models. Use them to help make sure your team covers all the different types of tests you need in order to deliver the right value for your customers.

Using Other Influences for Planning

There are many useful models and ideas for helping us in our test planning, and we shouldn't throw them away. As Tim Ottinger and Jeff Langr have said (Ottinger and Langr, 2009b), a mnemonic for thinking about what are called nonfunctional requirements is still useful. The FURPS model (see Figure 8-5) was developed at Hewlett-Packard and was first publicly elaborated by Grady and Caswell (Wikipedia, 2014f); it is now widely used in the software industry. The + was later added to the model after various campaigns at HP to extend the acronym to emphasize various attributes.

James Whittaker developed a methodology he calls the Attribute Component Capability (ACC) matrix (Whittaker, 2011) to help define what to test based on risk. ACC consists of three different parts that define the system under test: Attributes, Components, and Capabilities. He defines these as:

- **Attributes** (adjectives of the system) are qualities and characteristics that promote the product and distinguish it from the competition; examples are "Fast," "Secure," "Stable," and "Elegant."
- **Component**s (nouns of the system) are building blocks that together constitute the system in question. Some examples of



FURPS+

Functionality

Usability

Reliability

Performance

Supportability

Plus:

Design constraints

Implementation req'ts

Interface req'ts

Physical req'ts

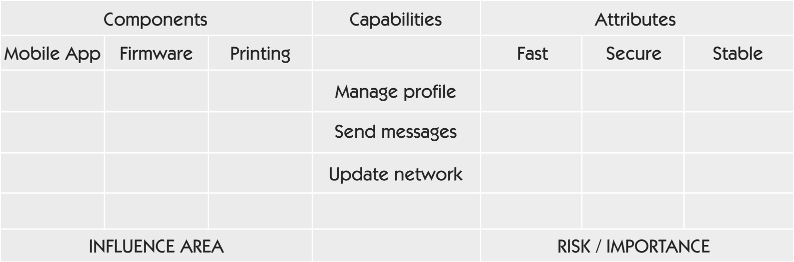
Figure 8-5 FURPS+ flash card (Ottinger and Langr, 2011)

Components are "Firmware," "Printing," and "File System" for an operating system project, or "Database," "Cart," and "Product Browser" for an online shopping site.

- **Capabilitie**s (verbs of the system) describe the abilities of a particular Component to satisfy the Attributes of the system. An example Capability for a shopping site could be "Processes monetary transactions using HTTPS." You can see that this could be a Capability of the "Cart" component when trying to meet the "Secure" Attribute. The most important aspect of Capabilities is that they are testable.

Creating a high-level matrix using this model can be a simple way to visualize your system. Figure 8-6 shows an example of what such a matrix might look like. Gojko Adzic agrees that exposing system characteristics and providing more visibility is definitely a good idea (Adzic, 2010a), though he cautions that while we can learn from other fields, we should be careful about using them as a metaphor for software development.

Use heuristics such as Elisabeth Hendrickson's "Test Heuristics Cheat Sheet" (Hendrickson, 2011) or tried-and-true techniques such as state diagrams or truth tables to think of new ideas for attributes. Combine these ideas with models like the Quadrants so that the conversations about the system constraints or usability can extract clear examples. Using all the tools in your toolbox can only help increase the quality of the product.



Components			Capabilities	Attributes		
Mobile App	Firmware	Printing		Fast	Secure	Stable
			Manage profile			
			Send messages			
			Update network			
INFLUENCE AREA				RISK / IMPORTANCE		

Figure 8-6 ACC example

Planning for Test Automation

Since Mike Cohn came up with his test automation pyramid in 2003, many teams have found it a useful model to plan their test automation. To take advantage of fast feedback, we need to consider at what level our automation tests should be. When we look at the standard pyramid, Figure 8-7, we see three levels.

The lowest level is the base—the unit tests. When we consider testing, we should try to push the tests as low as they can go for the highest return on investment (ROI) and the quickest feedback.

However, when we have business logic where tests need to be visible to the business, we should use collaborative tools that create tests at the service layer (the API) to specify them in a way that documents system behavior. See Chapter 16, "Test Automation Design Patterns and Approaches," for

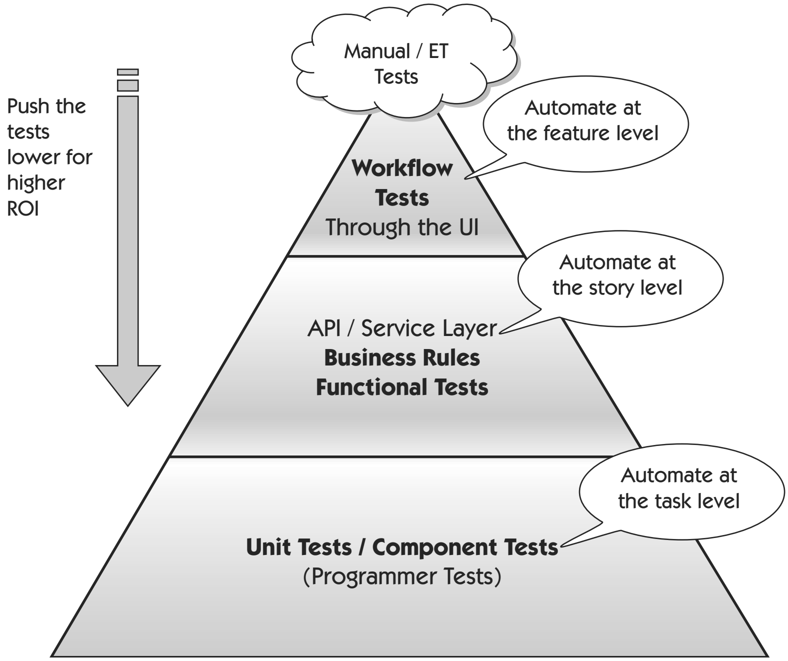

Manual / ET Tests
Push the tests lower for higher ROI
Automate at the feature level
Workflow Tests Through the UI
Automate at the story level
API / Service Layer Business Rules Functional Tests
Automate at the task level
Unit Tests / Component Tests (Programmer Tests)

Figure 8-7 Automation pyramid

more details. It is at this layer that we can automate at the story level so that testing and automation can keep up with the coding.

The top layer of the pyramid consists of the workflow tests through the user interface (UI). If we have a high degree of confidence in the unit tests and the service-level or API-level tests, we can keep these slower, more brittle automated tests to a minimum. See Chapter 15, "Pyramids of Automation," for more detail on alternative pyramid models.

Practices such as guiding development with examples can help define what the best level for the test is. A team's cadence can be set by how well they plan and execute their automation and how well they understand the level of detail they need. Consider also how to make your automation test runs visible, whether displayed in the continuous integration environment or on a monitor that is in the open.

Summary

Models are a useful tool for planning. In this chapter, we covered the following points:

- The agile testing quadrants provide a model for thinking about testing in an agile world.
  - The Quadrants help to emphasize the whole-team responsibility for testing.
  - They provide a visible mechanism for talking about the testing needed.
  - The left side is about guiding development, learning what to build, and preventing defects—testing early.
  - The right side is about critiquing the product, finding defects, and learning what capabilities are still missing.
- Gojko Adzic provides an alternative way to think about the Quadrants if you are in a lean startup or continuous delivery environment.
- We also introduced an alternative quadrant diagram from Elisabeth Hendrickson that highlights confirmatory checks versus investigative testing.

- There are already many tools in our agile testing toolbox, and we can combine them with other models such as the Quadrants to make our testing as effective as possible.
- FURPS and ACC are additional examples of models you can use to help plan based on risk and a variety of quality characteristics.
- The automation pyramid is a reminder to think about automation and to plan for it at the different levels.

This page intentionally left blank

INDEX

Numbers

"5 Whys," in visualization of thinking process, 44
7 Product Dimensions (Gottesdiener and Gorman)
  example applying 7 Product Dimensions, 131–133
  for identifying product needs at all planning levels, 129–131

A

AA-FTT (Agile Alliance Functional Test Tools), 4, 153
A/B tests
  in hypotheses testing, 134
  for mobile apps, 328
  overview of, 203–204
  UX designers and testers using, 140
ACC (Attribute Component Capability) matrix, model use in testing, 113–114
Acceptance tests
  automating, 214, 259
  defined, 415
  high-level, 153
  including UX design in tests, 141
  operational acceptance testing (OAT), 387–389, 419
Acceptance-test-driven development (ATDD)
  agile approaches used with mobile and embedded systems, 329
  confidence-building practices, 394
  functional test tools and, 237
  guiding with examples, 55, 148-152
  reducing defect debt, 214–215
  scaling "Discuss-Distill-Develop-Demo" cycle, 276
Accountability, from audits, 340
Accounting department, managing internal dependencies, 292
Actions, in 7 Product Dimensions, 130
Adzic, Gojko, xxix, 112, 153, 155
  about the contributors section, xxxv
  ATDD and, 151
  on automated tests as living documentation, 339
  on automating acceptance tests, 214
  challenge to Quadrants model, 108–112
  on fostering a learning culture, 10, 13–14
  on impact mapping, 123
  on model use for visibility, 114
  on SBE, 56, 153
  specification workshops, 42
  on testing through the UI, 243
Affinity diagrams, 44
"Agile Acid Test" (Hendrickson), 79
Agile Alliance Functional Test Tools (AA-FTT), 4, 153
Agile ALM (Hüttermann), 104
Agile Manifesto, 15, 87
The Agile Samurai (Rasmussen), xxi
Agile Testing: A Practical Guide for Testers and Agile Teams (Crispin and Gregory), xxi, 3, 282
  collaboration, 239
  customer and developer teams, 27
  defect-tracking systems, 321
  design principles and patterns, 240–241, 243
  documentation, 141, 340
  embedded systems, 326
  end game, 285
  example use, 145, 148, 155
  personas, 171
  SBTM, 176
  seven key success factors, 393
  story board examples, 386
  task creation, 97
  ten principles for testers, 28, 386
  test automation, 209, 223, 237, 254, 262
  test manager role, 19

Agile Testing: A Practical Guide for Testers and Agile Teams (*continued*)
test Quadrants, 65–66, 85, 108
tool selection, 264
tools for eliciting examples and requirements, 123
user acceptance testing, 90, 202
visibility, 381, 391
whole team approach in, xvii
steel threads, 160, 313
ALM. *See* Application life cycle management (ALM)
Analytical skills, requirements for DW/BI testers, 348. *See also* Business analysis (BA)
Analytics software, tools for mobile devices, 327
Andrea, Jennitta, 149
API level (service level)
in automation pyramid, 223–224
consistency in tool selection and, 265
testing through, 241–243
APIs (application programming interfaces)
architecture, 58
coding skills and, 57–58
whole team approach, 258–260
Appelo, Jurgen
on agile development, 16
"Feedback Wrap" workout, 45
Apple example, of value of product design, 140
Application life cycle management (ALM)
Agile ALM (Hüttermann), 104
defined, 415
Architecture, for API, 58
Ariely, Dan, 50
Arrange-Act-Assert pattern (Ottinger and Langr), 240
Articles, learning resources, 75
ATDD by Example (Gärtner), 56
Attribute Component Capability (ACC) matrix, model use in testing, 113–114
Attributes
in ACC matrix, 113
quality attributes. *See* Quality attributes
Audits
accountability from, 340
auditors as stakeholders, 342, 346
Automation, flexibility
of automation tools, 256
of build pipelines, 368
of build verification testing, 369–371
of extract, transform, and load (ETL) processes, 349–350
of provisioning of configuration base states, 374–376
of tests. *See* Test automation

B

Bach, Jon, 177
Backup/restore, managing Big Data, 358
"Bake-offs," in tool testing, 261–262
Barcomb, Matt
about the contributors section, xxxv
on becoming a generalizing specialist, 33–36
on use of test automation pyramid, 228–229
BDD. *See* Behavior-driven development (BDD)
Beck, Kent, 9
"Beginner's mind" (Hunt), 50
Behavior-driven development (BDD)
confidence-building (core) practices, 394
defined, 415–416
evolution of agile testing and, 3
functional test tools and, 237
Given, When, Then style, xxii–xxiii, 147, 152, 156
guiding development with examples, 55, 148, 152–153
use with mobile and embedded systems, 329
Benchmarking
in goal setting, 277
Big Data
challenges of, 357–359
database transactions and, 352
Big picture
benefits of agile principles, 297
feature testing and, 391, 396–397
key success factors, 393
product delivery and, 91
using the Quadrants model, 108
starting off right and, 239
visualizing, 127

Bligh, Susan
  about the contributors section, xxxv–xxxvi
  on user acceptance testing in enterprises, 294–296
BLM (build-measure-learn)
  defined, 416
  for early testing or exploring, 134
Bolton, Michael
  evolution of agile testing and, 3
  on testing as social science, 25
Brainstorming
  achieving consensus for automation solutions, 262, 263
  in collaboration process, 49, 52
  for example sources, 155
  facilitating, 42
  impact mapping, 123
  visualization tools, 44
Branching, continuous integration issues related to, 62–65
Brodwell, Johannes, 312
Browsers
  compatibility issues, 91
  testing browser-based applications, 244
Budd, Andy, 140
Bugs. *See* Defects/bugs
Build pipelines
  automating, 368
  defined, 416
  DevOps and, 367–368
  test environments and, 367
  verification testing, 369–371
Building the right thing. *See* Development, building the right thing
Build-measure-learn (BLM)
  defined, 416
  for early testing or exploring, 134
Business analysis (BA)
  combining with testing, 129, 395
  including business analysts on agile teams, 27–28
  onboarding process for business analysts, 37–38
  skill requirements, 137–139
  testers and, 139–140
Business capabilities. *See* Features
Business intelligence (BI) tests
  applying agile principles to, 351
  challenges of Big Data, 357, 359
  data in, 352–353
  learning to test, 351–352
  managing test data, 355–356
  for performance and scale, 357–359
  solving bad test data problem, 353–354
  unique aspects of, 347–350
Business rules
  levels for testing through the UI, 243
  in test automation pyramid, 233–234
Business value
  determining tester jobs. *See* Testers, determining job responsibilities
  developing the right thing. *See* Development, building the right thing
  examples. *See* Examples, guiding development
  key components of, 119–120, 122
  tests in delivering, 4–5
Buy-in, lack of, 160–161

C
Capabilities, in ACC matrix, 114
Capacity utilization, Zheglov on, 10–12
Card system, for tracking technical debt, 230
Carvalho, Paul
  about the contributors section, xxxvi
  on internationalization and localization testing, 195–199
Change
  adapting to (Ruhland), 15
  end-to-end testing, 229–230
  Fearless Change (Manns and Rising), 19
  learning and, 9
Charles, Fiona, 25
Charters
  creating, 168–171
  generating ideas for, 171
  journeys in creating, 175–176
  managing, 176
  personas in creating, 171–174
  in SBTM, 176–178
  stories as basis of, 175
  in TBMT, 178–183

Charters (*continued*)
tours in creating, 174–175
Charts
example use in everyday life, 146–147
visualizing what you are testing, 100
Clean Coder (Martin), 401
Coaches
coaching retreats, 74
learning from, 71
skills of, 48–49
Cockburn, Alistair, 36
Code/coding
checking business code with SonarQube, 366
creating tests before coding, 96
fast feedback and, 354
integrating with testing in TDD, 105–106
Q2 test guiding, 103
refactoring, 213
technical skills, 56–58
version (or source code) control, 60–65, 420
Cohn, Mike, xxix–xxx, 115–116, 223
Collaboration
in approach to tool selection, 264
collaborative vs. noncollaborative tools, 265
with customers, 147, 393
DevOps and. *See* DevOps
distributed teams and, 309–311
DW/BI tests and, 350
learning from, 73–74
listening and, 49
meeting regulatory needs, 344–346
process (Robson), 52–53
reducing technical debt, 218–220
technical skills and, 56
through tests, 311–312
tools for distributed teams, 319–321
using video chat tools for, 312, 319–320
valuing customer collaboration over contract negotiation, 151–152
whole team approach to testing and, 239
Command line, general technical skills, 59
Communication
conflict management, 318
cultural and language issues, 303–304
distributed teams and, 306–307, 309–311
of the importance of testing, 381–382
time zone issues, 304–305
tools for distributed teams, 319
use of video chat for, 306
Community of practice (CoP)
learning and sharing, 76
testers acting as, 80
Competencies. *See* Roles/competencies
Compliance. *See also* Regulatory environments
part of Release Done, 342
regulatory, 340–341
Component tests, in test automation pyramid, 224–225, 233–234
Concurrency tests, 194–195
Conferences, learning from, 72–74
Confidence-building (core) practices
context sensitivity, 399–400
continual learning, 397–398
example use, 394–395
exploratory testing, 395–396
feature testing, 396–397
keeping it real, 401
Configuration base states, automating provisioning of, 374–376
Conflict management, distributed teams and, 318
Consistency
organizational controls for achieving, 278
in tool selection, 265, 289
Constraints
minimum acceptable result, 277
planning Q4 tests and, 107–108
time constraints and organizational skills, 51
Context
business intelligence systems. *See* Business intelligence (BI) tests
data warehouses. *See* Data warehouses (DWs)
DevOps. *See* DevOps
distributed teams. *See* Distributed teams
embedded systems. *See* Embedded systems/software
enterprise-level testing. *See* Enterprises (large organizations)
overview of, 271–273
regulated environments. *See* Regulatory environments
Context sensitivity
acceptance tests and, 150

alternatives to Quadrant model, 112–113
Big Data and, 5
blurred roles and, 142
in coordination of multiple teams, 283
as core practice, 399–400
cultural issues facing distributed teams, 302–303
education relative to roles and responsibilities, 17
interdependence of software development and infrastructure and operations, 26
learning from spikes before planning, 90
mobile and embedded systems and, 5, 333
problem solving and, 43
Q4 tests (technology-facing tests) and, 65–66
in simplification of approach, 157
starting off right in test automation, 239
TBTM and, 182
test managers and, 19
Context-driven testing, 399
Continual learning. *See also* Learning
acquiring automation and coding skills, 57
BA skills and, 138
as core practice, 397–398
empowering team members for, 13
hiring people who want to learn, 37
time for, 9
from people in other departments, 365
small experiments in, 80
testing for mobile software, 336
T-shaped skills, 28–29
value of domain knowledge, 47
Continuous delivery, DevOps and, 366
Continuous Delivery (Humble and Farley), 361
Continuous improvement
kanban used for, 386–389
positive outcomes of, 297
T-shaped skills, 28
visualization for, 386, 390
Continuous integration (CI)
automated regression testing for, 289–290
building and maintaining test environments and, 60
core development practices, 394
defined, 416
determining how much automation is enough, 262
long-running tests and, 229
mobile and embedded systems and, 329
regression testing and, 97
failing builds and, 263
systems, 62–65
time zone issues, 305
T-shaped skills and, 29
visual cues of technical debt, 217
walking new tester through CI process, 38
Contract negotiation, valuing customer collaboration over, 151–152
Control
in 7 Product Dimensions, 130
dealing with organizational controls, 278–283
Conversations
elicit examples, 239
as means of delivering value, 150–152
value of, 147
Coordination, of multiple teams, 283–284
CoP (Community of practice)
learning and sharing, 76
testers acting as, 80
Core practices. *See* Confidence-building (core) practices
Courses, learning from, 74–75
Co-workers, learning from, 77
Creativity, learning and, 8
Critical thinking, 49
Cross-site scripting, security tests and, 65–66
Cucumber and Cheese (Morgan), 332
The Cucumber Book (Wynne and Hellesøy), 56
Cucumber tool, 265
Cultural issues, distributed teams and, 302–303
Cunningham, Ward
"checked examples" and, 103
coining term "technical debt," 211
evolution of agile testing and, 3
Customers
in *Agile Testing* (Crispin and Gregory), 27
buy-in lacking, 160–161
capturing expectations of, 152
collaborating with, 147, 393
customer-facing tests guiding development, 239
determining purpose of new features and, 122
enterprises involving, 294–296
focus on, 276

Customers (*continued*)
playing role of "bad customer" in exploratory testing, 166–167
teams and, 416
tools for engaging. *See* Tools, for customer engagement
valuing customer collaboration over contract negotiation, 151–152
Cycle time
defined, 416
DevOps shortening, 361–362
reducing, 367
setting goal for, 387–388

D

Dashboard, visibility to tests and test results, 392
Data
in 7 Product Dimensions, 130
business intelligence and, 352–353
challenges of Big Data, 357, 359
managing test data, 249–250, 355–356
solving bad test data problem, 353–354
Data integrity tests, skill requirements for DW/BI testers, 348
Data modeling
abstraction of logical models, 351
for data warehouse, 347
Data warehouses (DWs)
applying agile principles to, 351
challenges of Big Data, 357, 359
data in, 352–353
learning to test, 351–352
managing test data, 355–356
solving bad test data problem, 353–354
succeeding with agile testing in, 400
testing performance and scale, 357–359
unique aspects of testing in, 347–350
Database team, managing internal dependencies, 292
Databases
DW/BI tester skills, 348
general technical skills, 59
de Bono, Edward, 50
Deadlines
building trust and, 17
pressure of unrealistic, 7–8
Debrief sessions
following testing tours (Gärtner), 175
recording results of exploratory testing, 186
SBTM use for training testers, 176
Debugging. *See* Defects/bugs
Decision tables, test design techniques, 67
Default Data pattern (Morgan), 250
Defects/bugs
"bug hunts," 183
catching/tracking, 392
debugging test failures, 250–251
implementing agile at Dell and, 281
making test maintenance visible, 213
preventing, 36, 47, 102, 158, 216, 362
prioritizing fixes, 289
processes for dealing with, 287
reducing defect debt (zero tolerance), 213–216
visual (whiteboard) approach to, 81–82
Defect-tracking systems (DTSs), 321
Definitions/assumptions, in collaboration process, 52
Deliverables
formal documentation of, 92–94
impact mapping and, 124–125
Delivery. *See also* Product release
delivery (development) team, 416–417
product delivery and, 296–297
single and multiple-team cycles, 91
Dell example, of agile journey
challenges and solutions, 279–280
evolution of, 282
implementing, 281
results, 296
scaling agile testing, 287–288
Dependencies
coordinating between teams, 285
customer involvement in enterprises, 294–296
distributed teams, 305
managing, 292
removing, 293
third-party partnerships, 292–294
Derby, Esther
adding feedback loops and, 16
Problem Solving Leadership (PSL) course, 74
Design patterns and principles
Default Data pattern (Morgan), 250

Factory pattern, 410–412
overview of, 240
Page Object pattern, 246–248, 407–410
testing through the API, 241–243
testing through the UI, 243–246
Desktop sharing, collaboration tools, 321
Detail, getting bogged down in, 159–160
Detail level in planning. *See* Levels of precision (detail), for planning
Development. *See also* DevOps
core practices, 394
environment for, 59–60
guiding with examples, 55–56, 148–149
liability in separating from IT or operations, 365
need for testers in, 26–27
planning iterative development, 88
tests guiding, 102
Development (delivery) teams
adopting agile values, 7
bonding with, 345
collaborating to meet regulatory needs, 344–345
defined, 416–417
Development, building the right thing
7 Product Dimensions (Gottesdiener and Gorman), 129–131
determining the purpose of new features, 121–123
example applying 7 Product Dimensions, 131–133
impact mapping, 123–126
investing and, 134–135
overview of, 119–120
story mapping, 126–129
tools for customer engagement, 123
tools for exploring early, 134
DevOps
adding infrastructure to testing scope, 365–367
in all Quadrants, 363–364
automating build verification testing, 369–371
automating provisioning of configuration base states, 374–376
branching issues and, 63
build pipeline and, 367–368
defined, 417
evolution of agile testing and, 5
interdependence of software development and infrastructure and operations, 26
monitoring and logging and, 59
overview of, 361–362
quality and, 363
testers adding DevOps value, 371–372
testing infrastructure, 372–373
DevOps for Developers (Hüttermann), 361
Dinwiddie, George, xxix, 4
Discover to Deliver (Gottesdiener and Gorman), 107, 191–192
"Discuss-Distill-Develop-Demo" cycle, in ATDD, 276
Dispersed teams. *See also* Distributed teams
communication and collaboration and, 310
overview of, 299–301
visualizing testing in, 386
Distributed teams
challenges of, 302
collaboration and, 309–312, 400
collaboration tools, 319–321
communication and, 309–311, 400
communication tools, 319
coping strategies, 308
cultural issues, 302–303
dependencies, 305
experience of working on, 306–307
experience working on offshore test team, 315–317
facilitating online communication, 306
integrating, 308–309
language issues, 303–304
management issues, 317–318
offshore testing, 312–317
overview of, 299–302
planning sessions, 305, 308
reasons for, 301–302
short feedback loops, 321–322
time zone issues, 304–305
tools for, 319–322
visualizing testing, 386
Documentation. *See also* Living documentation
of deliverables, 92–94
excessive, 341–342
"lack of documentation" myth, 339–340
linking to requirements, 345

Documentation (*continued*)
recording results of exploratory testing, 185
skill requirements, 141–142
training end users, 295
Domain knowledge, 46–47
Domain-specific language (DSL)
collaborating with distributed teams, 311
describing feature behavior, 147
example use and, 157–158
guiding development with customer-facing tests, 239
programmer buy-in and, 160
tests using, 56
in whole team approach, 259
Done
Feature Done, 286
Product Done, 287
Release Done, 342
Story Done, 286
Don't repeat yourself (DRY) principle, 35, 240
Driving development. *See* Examples, guiding development
Dropbox
collaboration tools, 312
experience working on offshore test team, 315–317
DRY (Don't repeat yourself) principle, 35, 240
DSL. *See* Domain-specific language (DSL)
DTSs (Defect-tracking systems), 321
DWs. *See* Data warehouses (DWs)

E

Elaboration, types of thinking skills, 49
Eliot, Seth, 200–201
Email, as communication tool, 319
Embedded systems/software
automation-assisted exploratory testing (Hagar), 334–335
critical nature of testing in, 328–329
defined, 325
learning agile testing for mobile software (Harrison), 336–337
lessons learned in applying agile testing (Hagar), 330–332
overview of, 325–326
similarities and differences of agile testing in, 326–328
test automation strategies (Morgan), 332–334
testing context and, xxvi
types of agile approaches used with, 329
Emery, Dale, 248
Emotions, learning and, 70–71
Empathy, giving/receiving feedback and, 45
End game
activities, 285
additional testing during, 90
addressing integration issues early, 291
defined, 417
taking release to Product Done, 287
End-to-end testing
Ruhland on, 193
susceptibility to change, 229–230
test automation pyramid, 225
Enterprise Solutions Group (ESG), at Dell
challenge of hardware compatibility matrix, 369, 371
challenges in transitioning to agile, 266–267, 279–280
results of adopting agile, 296
Enterprises (large organizations)
aligning target setting for test automation across, 277–278
Big Data as challenge to, 357
coordinating multiple teams, 283–284
coordinating tooling for, 289–290
customer involvement, 294–296
defined, 417
dependency management, 292
implementing test automation in, 254–258
organizational controls, 278–283
overview of, 275
product delivery and, 296–297
scaling automation for, 264–265, 267–268
scaling testing for, 276–277
system test team and environment and, 284–289
test coverage, 291
third-party partnerships, 292–294
version control, 290
what they are, 275–276
Environments
in 7 Product Dimensions, 130

in DevOps, 376
development environments, 59–60
feature testing by, 286
system test team and, 284–289
test environments. *See* Test environments
ESG. *See* Enterprise Solutions Group (ESG), at Dell
ETL. *See* Extract, transform, and load (ETL)
Evangelisti, Augusto, xxix
about the contributors section, xxxvi
on branching strategy, 63
on conversation value, 150
on managing testers, 19
on quality guilds, 14
on reducing defect debt, 213–216
Evans, David
about the contributors section, xxxvi
on automated tests as living documentation, 339
on getting to know distributed team members, 310
"Pillars of Testing" model, 402–405
on spending time effectively, 9
Everyday Scripting with Ruby (Marick), 57
Examples, guiding development
ATDD, 149–150
BDD, 152–153
benefits of, 157–158
capturing examples and turning into automated tests, 147
"checked examples" in Q1 and Q2 tests, 103
collaborating with distributed teams, 311
conversation as means of delivering value, 150–152
as core practice, 394–395
in defect prevention, 158
evolution of agile testing and, 3
overview of, 55–56, 145
pitfalls of, 159–162
planning Q2 tests and, 105
power of using, 145, 147–148
resource materials for learning mechanics of, 162
SBE, 153–154
solving bad test data problem, 354
use in real life, 146–147
various approaches to, 148–149
where to get, 155–157
Executives, educating, 17–19
Experiments
learning and, xxviii
small experiments in continual learning, 80
Exploratory testing
in agile testing context, 188–190
automation-assisted, 334–335
collaboration with development team and, 345
as core practice, 395–396
creating test charters, 94, 168–171
defined, 417
in DW/BI, 350
in evolution of agile testing, 5
generating test charter ideas, 171
groups in, 183–185
helping others and, 80
Hendrickson on, 66–67
journeys in, 175–176
managing test charters, 176
offshore test teams and, 315
overview of, 165–167
personas in, 171–174
Q3 tests (business-facing tests) and, 66, 103, 106–107
recording results, 185–188
in SBTM, 176–178
skill development for, 167
skills requirements in square-shaped teams, 30–32
stories in, 175
in TBMT, 178–183
in test automation pyramids 223, 227–228
tours in, 174–175
Explore It (Hendrickson), 50, 66, 106, 168
Extract, transform, and load (ETL)
defined, 417
learning to test BI and, 351
managing test data and, 355–356
speeding up/automating, 349
testing data rules, 353–354
Extreme Manufacturing, 331
Extreme Programming (XP)
agile testing quadrants and, 101
evolution of agile testing and, 3–4
testing needs and, 192

F

Facilitation skills, thinking skills, 42–43
Factory pattern, design patterns, 410–412
Fail fast, learning and, 12
Farley, David, 361
Fast feedback, agile principles
  API level tests and, 223
  applying to DW/BI, 351–352
  automated tests and, 97, 153, 161, 227
  automation levels and, 115
  A/B tests and, 204
  build pipeline and, 416
  DevOps and, 362
  in distributed teams, 301
  learning from, 12, 87, 394
  as objective of test execution, 256
  prioritizing information from, 347
  for product improvement and visibility, 17
  providing information to developers, 331
  rapid release cycles and, 200
  on regression failures, 234, 289
  for risk reduction, 291
  test environments and, 60
  in testing and coding approach, 354
  "Trinity Testing" (Harty) and, 184
  from users, 327
Fazal, Kareem
  about the contributors section, xxxvi
  on automating build verification testing, 369–371
Fearless Change (Manns and Rising), 19
Feature Done, 286
Feature injection, BDD and, 153
Feature testing
  as core practice, 396–397
  in different environments, 286
  iterative testing and fast feedback and, 5
  release planning and, 92
  remember the big picture, 396–397
  story maps in, 126
Features
  exploratory testing at feature level, 189
  levels of precision in planning, 92–94, 96
  overview of, 88
  planning Q2 tests and, 105
  planning Q3 tests and, 106–107
  prioritizing, 283
  the "why" of new features, 121–123
Federated data, 348, 417
"Feedback Wrap" workout (Appelo), 45
Feedback/feedback loops
  continuous integration and, 262
  fail fast and, 12
  fast feedback. *See* Fast feedback, agile principles
  impact mapping and, 125
  key success factors, 393
  mobile and embedded systems and, 326
  offshore test teams and, 316–317
  organizational culture and, 15–17
  planning for, 87
  from prototyping, 205–206
  regression testing and, 161
  short feedback loops in agile testing, 321–322
  skill in giving/receiving, 45–46
  test automation pyramid and, 223
Fishbone diagrams. S*ee* Ishikawa diagrams, 44
FIT (Framework for Integrated Test), 3
FitNesse tool, 254–255, 257, 265
Flow diagrams
  benefits of flow-based systems, 383
  example use in, 145
  kanban and, 88
  story level and, 96
Focus groups, 140
Framework for Integrated Test (FIT), 3
Freeman, Steve, 126
Frempong, Benjamin
  about the contributors section, xxxvii
  on automating provisioning of configuration base states, 374–376
Functional decomposition, types of thinking skills, 49
Functional tests
  ATDD and, 150
  in expanded test automation pyramid, 233–234
  in test automation pyramid, 115
  tools for, 237
  turning examples into automated tests, 147
Functionality
  key components of business value of software, 122
  testing through the UI, 243
FURPS model, for planning tests, 113–114

G

g11n. *See* Globalization
Gärtner, Markus, xxix
  on ATDD, 56
  on debriefing following tours, 175
  Pomodoro Testing, 182–183
  on starting testing process right, 239
  on technical debt, 211
Gawande, Atul, 9
Generalizing specialists, 33–36
George, Chris
  about the contributors section, xxxvii
  on reducing technical debt, 218–220
Gilb, Kai, 107
Gilb, Tom, 107, 277
Git tool, 365–366
Globalization
  challenges of global markets, 195
  defined, 415
  guiding development with examples, 55–56, 148–150
  language and character set support, 198
  planning Q4 tests and, 107
  programmer buy-in and, 160
Goals
  in collaboration process, 52
  metrics in setting, 277
  SMART, 49, 386
  stakeholders in goal setting process, 124
Gorman, Mary, xxx
  7 Product Dimensions, 129, 131–133
  about the contributors section, xxxvii
  given_when_then template for getting examples, 156
  on Planguage, 107
  on representation of quality attributes, 191
GoToMeeting, collaboration tool, 312
Gottesdiener, Ellen, xxx
  7 Product Dimensions, 129, 131–133
  about the contributors section, xxxvii
  given_when_then template for getting examples, 156
  on Planguage, 107
  on representation of quality attributes, 191
Graphs, for recording test results, 187
Group chats, collaboration tool, 321
Group hugs
  concurrency testing and, 194–195
  exploratory testing and, 183–184
Groups, in exploratory testing, 183–185
Guest, David, 29
Guiding development. *See* Examples, guiding development

H

Hagar, Jon
  about the contributors section, xxxviii
  on automation-assisted exploratory testing, 334–335
  defining embedded software and mobile apps, 325
  on lessons learned applying agile testing to mobile and embedded systems, 330–332
Hagberg, Jan Petter, 303, 322
Hardware compatibility matrix, at Dell, 369
Hariprasad, Parimala
  about the contributors section, xxxviii
  on offshore testing teams, 315–318
Harrison, JeanAnn, xxix
  about the contributors section, xxxviii
  on collaboration in meeting regulatory needs, 344–346
  on learning agile testing for mobile software, 336–337
Harty, Julian, 330
  on critical nature of testing for mobile apps, 328
  on “Trinity Testing,” 184
Hassa, Christian, 276–277
Heinrich, Mike
  about the contributors section, xxxviii
  example of data warehouse setup—Figure 22-1, 348
  on learning to test BI, 351–352
Heinze, Sherry, xxix
  about the contributors section, xxxvix
  on example use in everyday life, 146–147
Hellesøy, Aslak, 56
Help Desk persona, 172–173
Hendrickson, Elisabeth, xvii, 109, 214
  “Agile Acid Test,” 79
  ATDD and, 151
  challenging use of Quadrants, 112–113

Hendrickson, Elisabeth (*continued*)
- "Discuss-Distill-Develop-Demo" cycle, 276
- evolution of agile testing and, 3
- on exploratory testing, 66–67
- on planning Q3 tests, 106–107
- on resources for good charters, 168
- "Test Heuristics Cheat Sheet" as source for ideas in exploratory testing, 167
- on tools for structured and focused thinking, 50

Heuristics
- defined, 418
- in exploratory testing, 166
- planning models and, 114

Heusser, Matthew
- about the contributors section, xxxvix
- on charters and session-based testing, 169–170
- on managing regression tests, 177–178

Hiring
- finding the right people, 36–37
- onboarding process for testers, 37–38

Hiring Geeks That Fit (Rothman), 37

How Google Tests Software (Whittaker), 109

"How?" questions
- in creation of roadmaps, 123–124
- iPhone example, 140

Humble, Jez, 361

"Hump of pain," in test automation, 237

Hunt, Andy, 49–50

Hussman, David, 71
- on journeys as means of creating charters, 175–176
- persona use by, 171

Hüttermann, Michael, 372
- about the contributors section, xxxvix
- on adding infrastructure to testing scope, 365–367
- on DevOps, 361
- on "outside-in, barrier-free collaborative" in Q tests, 103–104

Hypotheses, A/B tests for checking, 134

I

i18n. *See* Internationalization

IaaS (Infrastructure as a service), 418

IDEs. *See* Integrated development environments (IDEs)

Impact mapping
- overview of, 123–126
- for product vision, 89
- tools for visualization of thinking process, 43–44

Infrastructure
- adding infrastructure to testing scope, 365–367
- "Levels of Service" provided by infrastructure testers, 371–372
- support for distributed teams, 322
- testing, 372–373
- testing requirements and, 192

Infrastructure as a service (IaaS), 418

Initiative, testers taking, 142–143

Integrated development environments (IDEs)
- collaboration and, 56
- manage automation with, 262
- tester expertise with IDE tools, 60
- T-shaped skills and, 29
- unit tests in, 4

Integration
- addressing integration issues early, 291
- of distributed teams, 308–309
- product release level and, 90
- system integration team in offshore testing, 313

Integration tests, offshore testing and, 312–314

Integrity checks, data and relational, 59

Interfaces
- in 7 Product Dimensions, 129–130
- technical skills and, 58
- user interfaces. *See* User interface (UI)

Internationalization
- planning Q4 tests and, 107
- testing needs and, 195–200

Interoperability, test planning and, 91

Investigative testing
- A/B tests. *See* A/B tests
- concurrency testing, 194–195
- exploratory testing. *See* Exploratory testing
- internationalization and localization, 195–200
- overview of, 163
- regression testing. *See* Regression tests
- types of, 164
- user acceptance testing. *See* User acceptance testing (UAT)
- user experience testing. *See* User experience (UX)

Ishikawa diagrams
types of thinking skills, 49
for visualization of thinking process, 44
IT departments, liability in separating from development, 365
Iterations
iteration demos as learning opportunity, 77
managing Big Data, 358
planning, 88, 93, 96–97, 130, 293, 310
planning meetings, 173–174, 186
regression testing and, 98
stories and, 28
visual cues of technical debt, 217
Ivarsson, Anders, 297

J
JavaScript, test automation pyramid and, 226
JBehave, for automating acceptance tests, 214
Jdefault library, 250
Jenkins tools
in DevOps example, 366
testing infrastructure with, 373
Job responsibilities. *See* Roles/competencies; Testers, determining job responsibilities
Johnson, Karen N., 330
Jones, Griffin
about the contributors section, xxxvix
on possibility of agile in regulated environment, 343
Journeys, in creating test charters, 175–176
JUnit tests, 265

K
Kahneman, Daniel, 50
Kämper, Stephan
about the contributors section, xl
on testing infrastructure, 372–373
Kanban
for capacity utilization (Zheglov), 10–12
as communication tool, 319
for continuous improvement, 386–389
course on, 75
defined, 418
distributed teams and, 311
as flow-based method, 88
in reducing defect debt, 214
use in testing (Rogalsky), 382–385
for visualizing testing, 382–385
Kaner, Cem, 399
Karten, Naomi, 49
Keeping it real
continual improvement and adaptation, 6
as core practice, 401
cutting corners and, 220
getting buy-in, 160
maintaining automation tests, 248
making regulatory requirements part of work, 344
managing source code, 64
options in dealing with uncertainty, 18
planning and, 87
regression testing and, 200
tests, skills, and resources and, 104–105
trust building within teams, 45
unsustainable pace and, 8
value of whole team understanding operating environment, 376
"what" vs. "why" in feature development, 122
working with third-parties and, 293
Kelln, Nancy, 25
Keogh, Liz, xxix, 18
on BDD, 152
on getting needed capabilities in place before testing, 239
on having too many unknowns, 161–162
on reviewing workflow for simplicity, 385
on tests for monitoring system performance, 107–108
on value of conversations, 147
Kerievsky, Joshua, 3
Khoo, Trish
about the contributors section, xl
on need for testers in development, 26–27
Kniberg, Henrik
on use of agile by Spotify, 16
using agile principles without being too rigorous, 297
Knight, Adam, xxix
about the contributors section, xl
on branching strategy, 64–65
on development environments, 59
DW/BI tests and, 350

Knight, Adam (*continued*)
experience using TBTM, 182
on managing testers, 19
on square-shaped team, 30–32
on testing performance and scale, 357–359
on tools for extending test harnesses, 237–238
Kohl, Jonathan, 326, 330

L

L10n. *See* Localization
Lambert, Rob, 29
Langr, Jeff
Arrange-Act-Assert pattern, 240
automating acceptance tests, 214
mnemonic for nonfunctional requirements—FURPS, 113
Large organizations. *See* Enterprises (large organizations)
Leadership skills, 74
Lean Coffee sessions, 14
Lean principles
implementing, 276
just in time and, 85
in manufacturing, 10
startup, 108–110, 134, 203
Learning. *See also* Continual learning
change and, 9
conferences, courses, meet-ups, coach camps and collaborating, 72–75
domain knowledge, 46–47
evolution of agile testing and, 5
experiments and, xxviii
fostering a learning culture, 13–15, 77
helping others, 79–80
importance of a learning culture, 12–13
making time for, 8–12, 77–79
mentors, learning from, 71
overview of, 69
publications, podcasts, and online communities, 75–77
resources, 72–74
roles and competencies and. *See* Roles/competencies
social networking, 74
styles, 69–72
surprise learning, 80–83
technical skills. *See* Technical awareness (technical skills)
to test business intelligence, 351–352
thinking skills. *See* Thinking skills
Legacy systems
defined, 418
internationalization requirements, 198
reducing technical debt, 218
session-based testing and, 169
test strategy for, 177
Levels of precision (detail), for planning
applying to enterprise example, 284
different points of view and, 87–89
feature level, 92–94, 96
overview of, 87
product delivery cycle, 91–92
product release level, 89–90
regression testing and, 97–98
story level, 96
task level, 96–97
visualizing what you are testing, 98–100
Listening
learning by, 69
thinking skills, 48–49
Living documentation. *See also* Documentation
continuity provided by domain-specific language, 239
SBE and, 153–154, 420
TDD and, 56
test automation and, 38, 209, 268, 339, 342
testing through the API and, 241–243
testing through the UI and, 243
L-mode (linear and slow), thinking skills and, 49
Load testing
applying Q4 tests, 103
test environments, 61
Localization
planning Q4 tests and, 107
testing needs and, 195–200
Log files, reading, 59
Logic Bubbles, thinking skills and, 50
Lyndsay, James, xxix, 178
on breadth and specificity of test charters, 169
resources for session-based testing, 178
on scripted testing vs. exploratory testing, 166

M

Maassen, Olav, 123
Maintainability, in expanded test automation pyramid, 233–234
Maintenance
 making test maintenance visible, 213
 of tests, 248–251
 visual cues of technical debt, 217
Maksymchuk, Cory
 about the contributors section, xl
 on using examples to prevent defects, 158
Management 3.0 (Appelo), 16
Management issues, distributed teams and, 317–318
Manns, Mary Lynn, 19
Mapping
 impact mapping. *See* Impact mapping
 mind mapping. *See* Mind mapping
 relationship mapping, 49
 story mapping. *See* Story mapping
 tools for structured and focused thinking, 50
Marick, Brian, 103
 on automating acceptance tests, 214
 evolution of agile testing and, 3
 on example-driven development, 145
 on learning to code, 57
 on Quadrants, 101
Martin, Robert C., (Bob)
 on building trust, 17
 on prioritization of high-value work, 401
 on rules of clean code, 35
Matrices
 adding visibility to tests and test results, 390
 Attribute Component Capability (ACC) matrix, 113–114
 monitoring risks and assumptions, 92
 release-level test matrix, 99
Matts, Chris, 153
 on Real Options, 18
 on determining the most valuable options, 123
McDonald, Mark P., 276, 292
McKee, Lynn, 25, 55
McKinney, Drew
 about the contributors section, xli
 on user research, 205–206
McMahon, Chris, 310
Meetings
 GoToMeeting collaboration tool, 312
 for prioritization of work, 364
 getting the right people, 365
 standup meetings, 15
Melnick, Grigori, 214
Metrics, in goal setting, 277
Meyer, Geoff, xxix, 287–288, 296–297
 about the contributors section, xli
 on agile journey at Dell, 279–282
 on suitable approach to test automation, 266–267
Mind mapping
 adding visibility to tests and test results, 390–391
 as collaboration tool, 321
 converting from SBTM to TBTM, 180–181
 for documentation of deliverables, 94
 impact mapping developed from, 123
 mobile apps and, 327
 for recording results of exploratory testing, 185
 types of thinking skills, 49
 for visualization of thinking process, 43
 for visualizing what you are testing, 98–100
Minimum viable product (MVP)
 defined, 418
 visualizing, 126
Ministry of Testing, 327
Mission statement, as source for charters, 169
Mobile apps/mobile devices
 automation-assisted exploratory testing (Hagar), 334–335
 concurrency testing and, 194
 critical nature of testing in, 328–329
 integration of hardware and operating systems in, 344–345
 learning agile testing for (Harrison), 336–337
 lessons learned in applying agile testing (Hagar), 330–332
 overview of, 325–326
 similarities and differences in agile testing in, 326–328
 test automation, 244
 test automation strategies (Morgan), 332–334
 test planning and, 91
 types of agile approaches used with, 329

Models, data, 347
Models, planning
agile testing quadrants and, 101–105
alternative to Quadrants (Hendrickson), 112–113
challenges to Quadrants, 108–111
FURPS model and ACC matrix applied to, 113–114
heuristics applied to, 114
overview of, 101
Q1 tests, 105
Q2 tests, 105–106
Q3 tests, 106–107
Q4 tests, 107–108
test automation and, 115–116
Monitoring
general technical skills, 59
managing Big Data, 359
memory, CPU, logging and, 59
risks and assumptions, 92
testing requirements and, 192
Morgan, Jeff (“Cheezy”)
about the contributors section, xli
on managing test data, 249–250, 355–356
on test automation strategies for mobile and embedded systems, 332–334
on testing browser-based applications, 244–245
Morville, Peter, 328
Moss, Claire
about the contributors section, xli
on learning, 80–83
MVP. *See* Minimum viable product (MVP)

N

Naming conventions, test standards, 265
Natural language, use in BDD, 152
North, Dan
on BDD, 152
evolution of agile testing and, 3
Notes, recording results of exploratory testing, 185–186

O

OAT (operational acceptance testing)
cycle time from demo to, 387–389
defined, 419
Observation skills, 48
O’Dell, Chris
about the contributors section, xliii
on flipping an inverted test automation pyramid, 229–230
Offshore testing. *See also* Distributed teams
cultural issues and, 303
experience working on test team, 315–317
language usage, 312
overview of, 301
pros/cons, 312–315
One-page test plans, 94
Online boards, in visualizing testing for distributed or dispersed teams, 386
Online communities, learning from, 75–77
Online courses, learning from, 74
Online games, for integration of distributed teams, 309
Online mind mapping, visualizing what you are testing, 98–100
Online tracking tools, managing dependencies, 305
Open source projects, as learning opportunity, 76
Operational acceptance testing (OAT)
cycle time from demo to, 387–389
defined, 419
Operations
development and. *See* DevOps
liability in separating from development, 365
Options Board, for displaying 7 Product Dimensions, 132
OPV (Other people’s views), thinking skills and, 50
Oracles, in exploratory testing, 166
Organization, of regression tests, 97
Organizational culture
educating stakeholders, 17–19
fostering a learning culture, 13–15, 77
importance of a learning culture, 12–13
investing time in learning, 8–12
managing testers, 19–20
overview of, 7–8
transparency and feedback loops, 15–17
Organizations
dealing with organizational controls, 278–283
enterprise-level. *See* Enterprises (large organizations)
Other people’s views (OPV), thinking skills and, 50

Ottinger, Tim
- Arrange-Act-Assert pattern, 240
- on automating acceptance tests, 214
- mnemonic for nonfunctional requirements—FURPS, 113

Ownership, offshore test team and, 316

P

Page Object pattern
- creating maintainable UI tests, 245
- creating with `PageFactory` class, 410–412
- example written with Selenium 2 (WebDriver), 407–410
- implementing, 247–248
- test automation strategies for mobile and embedded systems, 332–334
- testing through the UI and, 243–246
- understanding, 246–247

Paper prototypes. *See* Prototyping

Parallelization, in managing Big Data, 358–359

Patton, Jeff
- persona use by, 171
- on story mapping, 126–127

Performance tests
- DW/BI and, 357–359
- in expanded test automation pyramid, 233–234
- test environments, 61

Personas
- creating test charters, 171–174
- playing role of "bad customer" in exploratory testing, 166–167
- in story mapping, 128
- user experience designers creating, 396

"Pillars of Testing" model (Evans), 402–405

Pilot projects, for working out kinks, 280

Planguage (Gilb and Gilb), 107, 277

Planning
- applying to enterprise example, 284
- distributed teams and, 305, 308
- example of lightweight test plan (BMI calculator), 95–96
- features, 92–94
- feedback/feedback loops, 87
- identifying product needs, 129–131
- iteration planning, 93, 96–97, 130, 293, 310
- iteration planning meetings, 173–174, 186
- "Levels of Service" provided by infrastructure testers, 371
- models. *See* Models, planning
- points of view in, 87–89
- at product level, 286
- question asker and, 25
- regression testing and, 97–98
- release planning, 89–92
- story readiness, pre-planning, 96, 293
- task level, 96–97
- visual (whiteboard) approach to, 81–82
- visualizing what you are testing, 98–100

Platform as a service (PaaS), 419

Play, as learning technique, 73–74

Plunkett, Brian, 280

POC (proof of concept), 64

Podcasts, learning from, 75–77

Points of view, in planning, 87–89

Pomodoro Testing (Gärtner), 182–183

"Power of Three" (Dinwiddie), 4

Practice, learning and, 74

Pragmatic Thinking and Learning (Hunt), 49

Prioritization
- of backlog, 173–174
- of defect backlog, 81
- of defect/bug fixes, 289
- of features, 46, 159, 283
- of outcomes, 17
- of stories, 127–128, 293
- of test charters, 396
- as a thinking skill for testing, 41, 49

Privacy, data-related issues, 353

Problem Solving Leadership (PSL) course, 43, 74

Problem solving skills, thinking skills, 43–44

Product Done, taking release to, 287

Product release
- defined, 419
- enterprises and, 296–297
- integrating exploratory testing into agile testing, 189
- overview of, 88
- planning, 89–92
- regulatory compliance as part of Release Done, 342
- single and multiple-team product delivery cycles, 91–92

Products
  roadmap, 90
  test planning at product level, 286
Profiling, general technical skills, 59
Programmers
  buy-in and, 160–161
  on system test team, 284
Programming, 57. *See also* Code/coding
Proof of concept (POC), 64
Prototyping
  example use in, 145
  use in testing, 205–206
  UX designers and testers both using, 140
“Provocation Starters” (Vaage), 71–72, 413–414
Pryce, Nat, 126
Pseudo code, automating tests and, 57
PSL (Problem Solving Leadership) course, 43, 74
Publications, learning from, 75–77
Puppet tools, in DevOps example, 365–367
Putting it all together
  confidence-building practices, 394
  context sensitivity, 399–400
  continual learning, 397–398
  creating shared vision, 402
  example use, 394–395
  exploratory testing, 395–396
  feature testing, 396–397
  keeping it real, 401
  “Pillars of Testing” model (Evans), 402–405
  seven key success factors, 393
Pyramid, test automation
  alternative versions, 224–227
  dangers of delaying test automation, 227–229
  evolution of agile testing at Dell, 282
  example of expanded pyramid, 231–234
  example of flipping an inverted pyramid, 229–231
  original version, 223–224
  overview of, 223
  planning for test automation, 115–116
  showing different dimensions, 231, 234

Q

qTrace tool, for recording, 187–188
Quadrants, agile testing
  alternative to (Hendrickson), 112–113
  A/B tests in Q2. *See* A/B tests
  challenges to, 108–111
  DevOps working across all, 363–364
  evolution of agile testing at Dell, 282
  order of applying, 103–105, 191, 239
  overview of, 101–103
  Q1 tests, 105, 145
  Q2 tests, 105–106, 145
  Q3 tests, 106–107
  Q4 tests, 107–108, 268
  technical debt due to skipping Q3 and Q4 tests, 211
  testing quality attributes, 65–67
  UAT in Q3, 201
Quality
  balancing many aspects of, 135
  DevOps and, 363
  guilds and, 14
Quality attributes
  in 7 Product Dimensions, 130
  in acceptance tests, 150
  representation in operations and development dimensions, 191
  testing, 65–67
Query languages, skill requirements, 59
Questioning (tools)
  tools for visualization
   of thinking process, 44
  “Why?,” “Who?”, “How?”, and “What?”, 123–124
Queueing theory, capacity utilization and, 10–12

R

Rainsberger, J. B., 147
Rall, Aldo, xxix
  about the contributors section, xli–xlii
  on helping others, 79
Rasmussen, Jonathan, xxi
Real Options (Matts and Maassen), 18
Reality checks. *See* Keeping it real
Recording
  refactoring ideas, on whiteboards, 230
  results of exploratory testing, 185–188
Refactoring
  defined, 419
  scaling automation for large organizations, 268
  whiteboard for recording ideas, 230

Reflect-and-adapt (Hendrickson), xviii
Regression tests
  challenges of, 200–201
  continuous integration with automated tests, 289
  defined, 419
  DevOps role building/deploying, 376
  DW/BI and, 349
  in expanded test automation pyramid, 234
  failures, 62
  key success factors, 393
  managing with SBTM charters, 177–178
  manual, 213, 223
  planning for, 97–98
  scaling for large organizations, 267–268
  team approach to biggest problem, 217–218
  technical debt in, 216–217
  testing through the UI and, 243, 245
  too many, 161
Regulatory environments
  agile use in regulated environments, 343
  collaboration in meeting regulatory needs, 344–346
  compliance and, 340–341
  excessive documentation and, 341–342
  including auditors in the solution, 342
  "lack of documentation" myth, 339–340
  overview of, 339
Relationship mapping, types of thinking skills, 49
Relationship skills, 74
Release candidates
  creating continuously, 365
  defined, 419
  product release level and, 189
Release Done, regulatory compliance as part of, 342
Release planning sessions, 92
Reliability, in expanded test automation pyramid, 233–234
Requirements gathering, 159
Resources, for learning
  conferences, courses, meet-ups, and collaborating, 72–75
  publications, podcasts, and online communities, 75–77
Retrospectives
  in addressing technical debt, 217
  collaboration tools, 322
  power of, 13
  self-managed teams and, 24
Return on investment (ROI)
  defined, 419
  key components of business value of software, 122
  meeting automation challenges and, 258
  test automation pyramid and, 115
  test planning and, 91
  testing through the UI and, 243
Ries, Eric, 134
Rising, Linda
  Fearless Change (Manns and Rising), 19
  on power of retrospectives, 13
Risk-based testing, 51
Risks, testing, 125
R-mode (nonlinear, fast, "rich"), thinking skills and, 49
Roadmaps, questions in creation of, 123–124
Robson, Sharon, xxix
  about the contributors section, xlii
  adding dimensions to test automation pyramid, 231–232
  on application of thinking skills to testing, 49–51
  on effective collaboration, 52–53
Rogalsky, Steve, xxix
  about the contributors section, xlii
  on learning, 79
  on podcast as learning resource, 75–76
  on story mapping and testing, 127–129
  on use of kanban in testing, 382–385
ROI. *See* Return on investment (ROI)
Roles/competencies
  advantages of some blurred roles on teams, 142
  competencies vs. roles, 24–28
  generalizing specialists, 33–36
  hiring the right people, 36–37
  onboarding process for testers, 37–38
  overview of, 23
  square-shaped team and, 30–32
  T-shaped skill set and, 28–30
  importance of titles (Walen), 25

Root-cause analysis, tools for visualization of thinking process, 44
Rothman, Johanna, xix
on feedback loops and transparency, 16
on hiring right people, 37
RSpec approach, to test-driven development, 259
Rubin, Ken, xxix
Ruby Faker library, 250
Ruhland, Bernice Niel, xxix, 77
about the contributors section, xlii
on adapting to change, 15
on group exploration process, 183
on maintaining list of testing types for her team, 193
on one-page test plans, 94
on recording results of exploratory testing, 187
on SBTM use to train testers, 176
on self-managed teams, 23–24
on testers and business analysts, 138

S

SaaS (software as a service)
defined, 420
product development and, 202
Sandboxes, test environments, 60
Satir Global Network, 74
SBE. *See* Specification by example (SBE)
SBMT. *See* Session-based test management (SBTM)
Scale, in agile testing
automation and, 358–359
considerations regarding, 275
Dell example, 287–288
for enterprises, 276–277
speed and, 357–358
Scenarios
creating test charters, 174
use of examples and, 159
Schoots, Huib
about the contributors section, xliii
on documentation, 94
experience working on distributed teams, 306–307
Scope
adding infrastructure to testing scope, 365–367
deriving using SBE, 153
Scott, Alister
alternative test automation pyramid, 227
on testing through the UI, 243
Scripted tests, 166
Scrum/scrum teams
Agile evolution at Dell and, 282
in automated build verification, 370
Extreme Manufacturing and, 331
first agile experiences compared with, 382
scaling agile testing, 287–288
sprint cycles, 266–267
time-boxing, 281
Security
data-related issues, 353
security worried persona, 172
applying Q4 tests, 103
quality attributes and, 65–66
skill requirements for DW/BI testers, 348
Selenium 2 (WebDriver)
Page Object example, 407–410
`PageFactory` class, 410–412
Selenium test library, 244, 254–255, 265
Self-managed teams, Ruhland on, 23–24
Semantic Studios, 329
Session sheets, recording results of exploratory testing, 185
Session-based test management (SBTM)
converting to TBTM, 179–181
managing regression tests with SBTM charters, 177–178
managing test charters, 176, 178
charters and session-based testing, 169–170
recording results of exploratory testing, 186
Set-based development, 245, 259, 420
Shannon, Paul
about the contributors section, xliii
on flipping an inverted test automation pyramid, 229–230
"Sharing the pain," 213
Shore, James, 214
Simplicity principle
applying to DW/BI, 351
applying to planning, 88
automated tests and, 256
mind mapping for detail capture, 94
reviewing workflow for simplicity, 385

Sinclair, Jennifer, xxix, xliii
Sinclair, Toby
about the contributors section, xliii
on A/B tests, 203–204
Single, responsibility, Open/closed, Liskov substitution, Interface segregation, Dependency inversion (SOLID), 57
Sjödahl, Lars, 304
Skills
business analysis skills, 137–139
DevOps activities and, 363
documentation skills, 141–142
investing time in learning, 8
thinking skills. *See* Thinking skills
square-shaped team example, 30–32
team building and, 24, 27–28
technical skills, *See* Technical awareness (technical skills)
T-shaped skill set, 28–30
UX design skills, 140–141
Slices
feature testing and, 396
testing, 127
Small chunks principle
applying to DW/BI, 351
build-measure-lean (BML) and, 134
charters as, 170
distributed teams and, 313
identifying chunks for testing, 127
for learning and adapting, 87
for planning, 87
stories and, 88, 103
for training, 38
SMART (specific, measurable, achievable, relevant, and time-boxed)
continuous improvement and, 386
types of thinking skills, 49
Smoke tests, 263
Socratic questioning, types of thinking skills, 49
Soft skills. *See* Thinking skills
Software as a service (SaaS)
defined, 420
product development and, 202
Software delivery cycle, feedback loops in, 15
Software Test Attacks to Break Mobile and Embedded Devices (Hagar), 330
SOLID: Single, responsibility, Open/closed, Liskov substitution, Interface segregation, Dependency inversion (SOLID), 57
Solving problems, 43–44
SonarQube, checking business code with, 366
Source code control system, 62–65, 420
tester expertise, 60
version control, 61
SOX compliance, regulatory environments, 340
Specialists, generalizing specialists, 33–36
Specification by Example (Adzic), 56
Specification by example (SBE)
agile approaches used with mobile and embedded systems, 329
confidence-building practices, 394
defined, 420
functional test tools, 237
guiding development with examples, 55–56, 148, 153–154
use in reducing defect debt, 214
in whole-team approach to meeting new challenges, 259
Specification workshops, facilitating, 42
Spikes
deferring test planning until after spike solutions, 92
planning and, 90
testing potential automation tool with, 261
Split testing. *See* A/B tests
Spotify, 16, 297
Spott, Dave, 280
Spreadsheets
example use in, 145, 157
recording results of exploratory testing, 186
struggling with failing CI builds, 263
SQL (Structured Query Language), 59
SQL injection, security tests, 65–66
Square-shaped team, 30–32
Staging environments, for testing, 60
Stakeholders
accountability to, 343
auditors as, 342
buy-in, 160–161
educating, 17–19
getting examples from, 147–148
giving/receiving feedback, 46

Stakeholders (*continued*)
  specification workshops and, 42
  tools for engagement with, 123–127
Standards, naming conventions, 265
Standup meetings, for face-to-face communication, 15
State transition diagrams, 67
Steel threads, 160, 313
Stories
  blocked, 292
  breaking features up into, 92, 94
  in coaching, 48
  for code refactoring, 213
  creating test charters from, 175
  determining purpose of, 121
  feature-testing, 94, 96
  getting examples for, 155
  guiding development with examples, 55
  iteration testing and, 28
  in levels of precision, 89
  levels of precision in planning, 96
  overview of, 88–89
  planning Q2 tests and, 105
  planning Q3 tests and, 106–107
  prioritizing, 127–128, 293
  regression testing and, 97–98
  team approach to biggest problem, 218
Story boards
  communication tools, 319
  effective visualization of testing process, 384–386
  getting to know distributed team members, 311
Story Done, vs. Feature Done, 286
Story mapping
  getting examples and, 155
  impact mapping developed from, 123
  overview of, 126–127
  testing and, 127–129
Stress tests
  meeting regulatory needs and, 345
  mobile devices and, 345
Structured Query Language (SQL), 59
Subversion, 64
Success
  importance of celebrating, 17
  key factors in, 393
Sweets, Tony
  about the contributors section, xliii–xliv
  on use of Page Object pattern, 246–248
System integration team, approaches to offshore testing, 313
System test team
  coordinating dependencies between teams, 285–286
  creating, 284–285
  managing, 287
System under test (SUT)
  changes and updates and, 240
  how much automation is enough and, 263

T

Tabaka, Jean, 71
Tables, recording results of exploratory testing, 187
Talks, Mike, xxix
  about the contributors section, xliv
  on constant learning, 78–79
  on downside of kanban boards, 384
  example of need for business analysis in testing, 138–139
  on learning domain knowledge, 46–47
  on onboarding process for new testers, 38
Tasks
  addressing both product and regulatory needs, 342
  creating testing tasks, 96–97
  exploratory testing with, 189
Taxonomy, Quadrants as, 104
TBMT. *See* Thread-based test management (TBMT)
TDD. *See* Test-driven development (TDD)
Teams
  adopting agile values, 7
  automation solutions for teams in transition, 253–254, 258
  bonding with, 345
  competencies included on, 27
  coordinating multiple, 283–284
  creating system test team, 284–289
  dependencies between, 92
  dispersed. *See* Dispersed teams
  distributed. *See* Distributed teams

finding/hiring the right people, 36–37
fostering a learning culture, 13–15
generalizing specialists and, 36
including business analysts on, 129
multidisciplinary, 256
product delivery cycles and, 91
self-managed (Ruhland), 23–24
square-shaped team, 30–32
technical skills and, 56
understanding team issues (Gregory), 18
whole-team approach. *See* Whole team
Technical awareness (technical skills)
automation and coding skills, 56–58
automation through the API, 240–243
automation through the UI, 243
continuous integration and source code control systems, 62–65
development environments and, 59–60
DW/BI testing, 348
general technical skills, 59
guiding development with examples, 55–56
learning resources, 74
overview of, 55
quality attribute testing, 65–67
test design techniques, 67
test environments, 60–62
Technical debt
defined, 420
making it visible, 212–213, 216–217
overview of, 211–212
quantifying cost of, 17
reducing defect debt, 213–216
reducing through collaboration, 218–220
team approach to, 217–218, 220
test automation volcano and, 229
unrealistic deadlines creating, 7–8
Technical skills. *See* Technical awareness (technical skills)
Technical writers, 141–142
Telecommuting, creating policy for, 310
Test automation
acceptance tests, 214
aligning target setting for test automation across enterprise, 277–278
automation-assisted exploratory testing (Hagar), 334–335
collaborative solutions in tool selection, 260–261, 264
continuous integration with automated regression testing, 289
data management, 249–250
design patterns and principles, 240
development environments and, 59–60
examples turned into automated tests, 147
exploratory testing used in conjunction with, 167
how much automation is enough, 262–263
implementing in large organizations, 254–258
maintaining tests, 248–251
for mobile and embedded systems, 332–334
overview of, 209–210
Page Object pattern and, 246–248
Planning, 115–116
pyramid approaches. *See* Pyramid, test automation
Q4 tests (technology-facing tests), 268
regression testing, 97
rules and reasons for, 241
scaling for large organizations, 264–265, 267–268
scaling for Big Data, 357–359
selecting test automation solutions, 253
solutions for teams in transition, 253–254, 258
starting off right, 239–241
technical debt and. *See* Technical debt
technical skills for, 56–58
test environments and, 61
testing through the API (service level), 241–243
testing through the UI, 243–246
third-party partnerships and, 293
tools and processes in, 237–238
transition challenges and, 266–267
whole team approach to, 238–239, 258–260
Test charters. *See* Charters
Test coverage
in enterprises, 291
medical device example, 344
Test design. *See also* Design patterns and principles
techniques, 67
use of diagrams for, 67
UX design skills, 140–141
UX designers creating personas, 396

Test environments. *See also* Environments
- build pipelines and, 367
- building and maintaining, 60–62
- DevOps in maintenance of, 362
- making visible, 291
- overview of, 60–62

Test failure
- debugging, 250–251
- managing, 289–290
- timing issues, 237

"Test Heuristics Cheat Sheet" (Hendrickson), 114

Test matrix. *See* Matrices

Test-driven design. *See* Test-driven development (TDD)

Test-Driven Development (Beck), 9

Test-driven development (TDD)
- BDD as response to, 153
- collaborating with distributed teams, 312
- core development practices, 394
- defined, 421
- DSL examples and, 158
- example-driven development compared with, 148
- integrating coding and testing, 105
- in mobile and embedded systems, 326, 329
- programmer buy-in and, 160
- reducing defect debt, 214
- scaling, 276–277
- at the task level, 96, 98

Testers. *See also* Roles/competencies
- adding DevOps value, 371–372
- agile principles for, 28–29, 386
- business analysis skills, 137–140
- as community of practice (CoP), 80
- database skills, 348
- determining job responsibilities, 137
- documentation skills, 141–142
- generalizing specialists and, 36
- giving/receiving feedback and, 46
- "Levels of Service" provided by infrastructure testers, 371–372
- managing, 19–20
- need for in development (Khoo), 26–27
- onboarding process for, 37–38
- quality guilds (Evangelisti), 14
- SBTM use in training, 176
- square-shaped team example, 30–32
- on system test team, 284
- taking initiative, 142–143
- technical skills and collaboration, 56
- Trio (developer, tester, BA) in exploratory testing, 184
- UX design skills, 140–141

Testing
- acceptance tests. *See* Acceptance tests
- A/B tests. *See* A/B tests
- BI tests. *See* Business intelligence (BI) tests
- business analysis overlapping with, 137
- component tests, 224–225, 233–234
- context-driven, 399
- creating tasks for, 96–97
- creating tests before coding, 96
- early, 134
- end-to-end tests. *See* End-to-end testing
- evolution of agile testing, 3–6
- example of lightweight test plan (BMI calculator), 95–96
- exploratory. *See* Exploratory testing
- features. *See* Feature testing
- functional tests. *See* Functional tests
- integrating coding with, 105
- investigative. *See* Investigative testing
- learning, 72–76
- load testing, 61, 103
- offshore. *See* Offshore testing
- performance tests. *See* Performance tests
- Pomodoro testing (Gärtner), 182–183
- prototyping in, 205–206
- Quadrants. *See* Quadrants, agile testing
- quality attributes, 65–67
- regression tests. *See* Regression tests
- requirements and purpose of (Walen), 139–140
- risk-based, 51
- risks, 125
- scaling tests, 276–277, 287–288
- scripted tests, 166
- session-based, 169–170
- slices, 127
- smoke tests, 263
- as social science (Walen), 25
- staging environments for, 60
- starting process right (Gärtner), 239

story mapping and, 127–129
stress tests, 345
testing-as-a-service, 314
thinking skills applied to (Robson), 49–51
through the API (service level), 241–243
through the UI, 243–246
types of, 192–194
unit tests. *See* Unit tests
user acceptance testing (UAT), 149–150
visualizing what you are testing, 98–100
workflow tests. *See* Workflow tests

Testing, in enterprise
aligning target setting for test automation, 277–278
implementing test automation in, 254–258
scaling testing, 276–277
system test team and environment and, 284–289
test coverage, 291

Testing in production (TiP) (Eliot), 200–201

Testing-as-a-service, 314

Thinking skills
coaching and listening, 48–49
collaboration skills, 52–53
diagrams for structured and focused thinking (Robson), 50
domain knowledge and, 46–47
exploratory testing and, 165
facilitation, 42–43
giving/receiving feedback, 45–46
learning resources, 75
organizational skills, 51
overview of, 41–42
problem solving, 43–44
thinking differently, 49–51

Thinking, Fast and Slow (Kahneman), 50

Thinking for Action (de Bono), 50

"The Thinking Tester" (Hendrickson), 112

Third-party
compatibility issues, 178
test libraries, 244

Third-party vendors
customer involvement in enterprises, 294–296
managing dependencies, 292–294
time frames and expectations of, 200
"why" of feature development and, 122

Thread-based test management (TBMT)
converting from SBMT to, 179–181
fractal representation of, 182
overview of, 178, 181–183

"Three Amigos" (Dinwiddie), 4

Thucydides, for automating acceptance tests, 214

Time constraints, organizational skills and, 51

Time management, making time for learning, 77–79

Time zones
communication and collaboration and, 309–310
coping strategies for distributed teams, 309
issues facing distributed teams, 304–305

Time-boxing
in collaboration process, 53
Scrum guidelines and, 281
story level and, 96
test planning and, 88

Titles, 25

Tools. *See also* by individual types
Agile Alliance Functional Test Tools committee (AA-FTT), 4, 153
analytic tools, 327
"bake-offs" for testing, 261–262
collaboration tools, 312, 322
collaborative approach to selecting, 264
consistency in selection of, 265, 289
coordination of tooling, 289–290
for DevOps, 366, 373
for exploring early, 134
facilitating online communication, 306–307
implementing agile at Dell and, 281
recording tools, 187–188
in test automation, 237–238
tester expertise with IDE tools, 60
thinking tools, 49–50
videoconferencing tools, 315–317
for visualization of thinking process, 43–44

Tools, for customer engagement
7 Product Dimensions (Gottesdiener and Gorman), 129–133
impact mapping, 123–126
overview of, 123
story mapping, 126–129

Tools, for distributed teams
collaboration tools, 319–321
communication tools, 319

Tours, creating test charters, 174–175
Transition, to agile development
  automation solutions for teams in transition, 253–254, 258
  challenges, 266–267
Transparency, in organizational culture, 15–17. *See also* Visibility
"Trinity Testing" (Harty), 184
Trust
  building, 17
  self-managed teams and, 24
T-shaped skills
  collaboration as means of developing, 344
  developing, 398
  generalizing specialists and, 33
  roles and competencies and, 28–30
Tung, Portia, 73–74

U

UAT. *See* User acceptance testing (UAT)
UIs. *See* User interface (UI)
Uncertainty, building trust and, 18
Unit tests
  consistency in tool selection and, 265
  defined, 421
  DW/BI and, 349
  JUnit tests, 265
  planning early and automating, 105
  in TDD, 98
  test automation pyramids and, 115, 224–225, 233–234
UNIX shell commands, skill requirements, 59
Usability feedback
  in expanded test automation pyramid, 233–234
  from prototyping, 205–206
User acceptance testing (UAT)
  acceptance tests contrasted with, 149–150
  in enterprises (Bligh), 294–295
  in expanded test automation pyramid, 233–234
  learning to test BI and, 351–352
  offshore testing and, 312–314
  overview of, 201–202
  product release level and, 90
  Q3 tests and, 107
  value of conversations and, 151–152
User experience (UX)
  A/B tests applied to, 203
  considering impact of stakeholders on business goals, 124
  design skill requirements, 140–141
  mobile devices and, 328–329
  testing, 205–207
  UX designers creating personas, 396
User interface (UI)
  achieving consensus for automation solutions, 260–261
  collaboration in meeting regulatory needs, 344
  consistency in tool selection and, 265
  example of benefit of including UX design in tests, 141
  mobile devices and, 328
  technical skills and, 58
  test automation and, 225–226, 287
  in test automation pyramid, 223–224
  testing mobile projects and, 344
  testing through the UI, 243–246
User stories. *See* Stories
User Story Mapping (Patton), 127
Users
  in 7 Product Dimensions, 129
  training end users, 295
Utilization, capacity utilization, 10–12
UX. *See* User experience (UX)

V

Vaage, Carol, 71–72, 413–414
Vagrant tool, used in DevOps example, 366
Vendors. *See* Third-party vendors
Version control
  in enterprises, 290
  Git and, 365
  source code control systems, 61–65, 420
Videoconferencing
  collaboration tools, 319–320
  working on offshore test team and, 315–317
Virtual machines, for running test suites, 262
Visibility
  communicating importance of testing, 381–382
  diagrams for visualization of thinking process, 44
  for continuous improvement, 386–390

giving/receiving feedback and, 46
kanban and, 382–385
online boards for distributed or dispersed teams, 386
overview of, 381
reducing need for control, 16
story boards and, 384–386
of technical debt, 212–213, 216–217
of test environments, 291
of tests and test results, 381, 390–392
working on offshore test team and, 316–317
Vision, creating shared, 402
Visual aids, 98–100
Visual learning, 69, 81
Visualization. *See* Visibility
Volcano, test automation, 228
Vuolli, Eveliina
about the contributors section, xliv
on introducing automation in enterprise system, 277–278

W

Walen, Pete
about the contributors section, xliv
on importance of titles, 25
on requirements and purpose of testing, 139–140
"Walking skeleton" (Freeman and Pryce), 126
Walshe, Mary
about the contributors section, xliv–xlv
on kanban for continuous improvement, 386–389
Web browsers. *See* Browsers
WebDriver tool, 261
WebEx, working on offshore test team and, 315–317
Weekend Testing, 74
"What?" questions
in creation of roadmaps, 123–124
iPhone example, 140
"what" vs. "why" in feature development, 122
Whelan, Declan, 153
Whittaker, James, 109
"Who?" questions
in creation of impact maps, 123–124
iPhone example impact map, 140
Whole team. *See also* DevOps
achieving consensus for automation solutions, 260–262
approach to agile testing, 362
approach to test automation, 238–239
key success factors, 393
meeting automation challenges, 228, 258–260
required for DW/BI testing, 348–349
understanding operating environment, 376
"Why?" questions
in building the right thing. *See* Development, building the right thing
iPhone example impact map, 140
questions in creation of impact maps, 123–124
Wiedemann, Christin
about the contributors section, xlv
on converting from SBTM to TBTM, 179–181
Wikis
adding visibility to tests and test results, 392
automation solutions in large organizations, 257
collaboration tools, 321
communication tools, 319
integration of distributed teams, 309
recording results of exploratory testing, 185
team wikis, 36
Winterboer, Lynn
about the contributors section, xlv
on solving the bad test data problem, 353–354
WIP (work-in-progress)
bandwidth for testing and, 401
kanban and, 382
Workflow tests
automating provisioning of configuration base states, 374–376
in expanded test automation pyramid, 233–234
levels for testing through the UI, 243
in test automation pyramid, 115–116
Work-in-progress *See* WIP
Workshops
getting participation in specification workshops, 155
as learning resource, 74
Wortel, Cirilo, 265
about the contributors section, xlv

Wortel, Cirilo (*continued*)
  on automation solutions in large organizations, 254–258
Wynne, Mat, 56

X

XP. *See* Extreme Programming (XP)

Z

"Zen, the Beginner's Mind" workshop (Hussman and Tabaka), 71
Zheglov, Alexei
  about the contributors section, xlvi
  on capacity utilization, 10