

/THEORY/IN/PRACTICE

A photograph of a beetle on a sandy surface. The beetle is in the upper right corner, and its tracks lead away from it, curving towards the bottom left. The sand is a warm, orange-brown color. The beetle is dark, possibly black or dark brown, with a metallic sheen on its back.

# Beautiful Testing

Leading Professionals Reveal  
How They Improve Software

O'REILLY®

Edited by Tim Riley  
& Adam Goucher

## Beautiful Testing

*“Any one of the insights or practical suggestions from these testing gurus would be worth the price of the book. The ideas are elegant and possibly challenging, yet are presented clearly and enthusiastically. This comprehensive, ambitious, engaging, and entertaining collection belongs on the bookshelf of every testing professional.”*

—Ken Doran, QA Lead, Stanford University; Chair, Silicon Valley Software Quality Association

Successful software depends as much on scrupulous testing as it does on solid architecture or elegant code. But testing is not a routine process; it’s a constant exploration of methods and an evolution of good ideas.

*Beautiful Testing* offers 23 essays—from 27 leading testers and developers—that illustrate the qualities and techniques that make testing an art. Through personal anecdotes, you’ll learn how each of these professionals developed beautiful ways of testing a wide range of products—valuable knowledge that you can apply to your own projects.

Here’s a sample of what you’ll find inside:

- Microsoft’s Alan Page knows a lot about large-scale test automation, and shares some of his secrets on how to make it beautiful
- Scott Barber explains why performance testing needs to be a collaborative process, rather than simply an exercise in measuring speed
- Karen N. Johnson describes how her professional experience intersected her personal life while testing medical software
- Rex Black reveals how satisfying stakeholders for 25 years is a beautiful thing
- Mathematician John D. Cook applies a classic definition of beauty, based on complexity and unity, to testing random number generators

**This book includes contributions from:**

Adam Goucher  
Linda Wilkinson  
Rex Black  
Martin Schröder  
Clint Talbert  
Scott Barber  
Kamran Khan

Emily Chen  
and Brian Nitz  
Remko Tronçon  
Alan Page  
Neal Norwitz,  
Michelle Levesque,  
and Jeffrey Yasskin

John D. Cook  
Murali Nandigama  
Karen N. Johnson  
Chris McMahon  
Jennitta Andrea  
Lisa Crispin  
Matthew Heusser

Andreas Zeller and  
David Schuler  
Tomasz Kojm  
Adam Christian  
Tim Riley  
Isaac Clerencia

All author royalties will be donated to the Nothing But Nets campaign to prevent malaria.

US \$49.99

CAN \$62.99

ISBN: 978-0-596-15981-8



9

**Safari**<sup>®</sup>  
Books Online

**Free online edition**  
for 45 days with purchase of  
this book. Details on last page.

**O'REILLY**<sup>®</sup> oreilly.com

# Beautiful Testing

Edited by Tim Riley and Adam Goucher

**O'REILLY®**

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

## **Beautiful Testing**

Edited by Tim Riley and Adam Goucher

Copyright © 2010 O'Reilly Media, Inc.. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://my.safaribooksonline.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Editor:** Mary E. Treseler

**Production Editor:** Sarah Schneider

**Copyeditor:** Genevieve d'Entremont

**Proofreader:** Sarah Schneider

**Indexer:** John Bickelhaupt

**Cover Designer:** Mark Paglietti

**Interior Designer:** David Futato

**Illustrator:** Robert Romano

### **Printing History:**

October 2009: First Edition.

O'Reilly and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Beautiful Testing*, the image of a beetle, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-0-596-15981-8

[V]

1255122093

*All royalties from this book will be donated to the UN Foundation's Nothing But Nets campaign to save lives by preventing malaria, a disease that kills millions of children in Africa each year.*

# CONTENTS

PREFACE		xiii
<i>by Adam Goucher</i>		
<b>Part One</b>	<b>BEAUTIFUL TESTERS</b>	
<hr/>		
<b>1</b>	<b>WAS IT GOOD FOR YOU?</b>	<b>3</b>
	<i>by Linda Wilkinson</i>	
<b>2</b>	<b>BEAUTIFUL TESTING SATISFIES STAKEHOLDERS</b>	<b>15</b>
	<i>by Rex Black</i>	
	For Whom Do We Test?	16
	What Satisfies?	18
	What Beauty Is External?	20
	What Beauty Is Internal?	23
	Conclusions	25
<b>3</b>	<b>BUILDING OPEN SOURCE QA COMMUNITIES</b>	<b>27</b>
	<i>by Martin Schröder and Clint Talbert</i>	
	Communication	27
	Volunteers	28
	Coordination	29
	Events	32
	Conclusions	35
<b>4</b>	<b>COLLABORATION IS THE CORNERSTONE OF BEAUTIFUL PERFORMANCE TESTING</b>	<b>37</b>
	<i>by Scott Barber</i>	
	Setting the Stage	38
	100%?!? Fail	38
	The Memory Leak That Wasn't	45
	Can't Handle the Load? Change the UI	46
	It Can't Be the Network	48
	Wrap-Up	51
<b>Part Two</b>	<b>BEAUTIFUL PROCESS</b>	
<hr/>		
<b>5</b>	<b>JUST PEACHY: MAKING OFFICE SOFTWARE MORE RELIABLE WITH FUZZ TESTING</b>	<b>55</b>
	<i>by Kamran Khan</i>	
	User Expectations	55
	What Is Fuzzing?	57
	Why Fuzz Test?	57

	Fuzz Testing	60
	Future Considerations	65
<b>6</b>	<b>BUG MANAGEMENT AND TEST CASE EFFECTIVENESS</b> <i>by Emily Chen and Brian Nitz</i>	<b>67</b>
	Bug Management	68
	The First Step in Managing a Defect Is Defining It	70
	Test Case Effectiveness	77
	Case Study of the OpenSolaris Desktop Team	79
	Conclusions	83
	Acknowledgments	83
	References	84
<b>7</b>	<b>BEAUTIFUL XMPP TESTING</b> <i>by Remko Tronçon</i>	<b>85</b>
	Introduction	85
	XMPP 101	86
	Testing XMPP Protocols	88
	Unit Testing Simple Request-Response Protocols	89
	Unit Testing Multistage Protocols	94
	Testing Session Initialization	97
	Automated Interoperability Testing	99
	Diamond in the Rough: Testing XML Validity	101
	Conclusions	101
	References	102
<b>8</b>	<b>BEAUTIFUL LARGE-SCALE TEST AUTOMATION</b> <i>by Alan Page</i>	<b>103</b>
	Before We Start	104
	What Is Large-Scale Test Automation?	104
	The First Steps	106
	Automated Tests and Test Case Management	107
	The Automated Test Lab	111
	Test Distribution	112
	Failure Analysis	114
	Reporting	114
	Putting It All Together	116
<b>9</b>	<b>BEAUTIFUL IS BETTER THAN UGLY</b> <i>by Neal Norwitz, Michelle Levesque, and Jeffrey Yasskin</i>	<b>119</b>
	The Value of Stability	120
	Ensuring Correctness	121
	Conclusions	127
<b>10</b>	<b>TESTING A RANDOM NUMBER GENERATOR</b> <i>by John D. Cook</i>	<b>129</b>
	What Makes Random Number Generators Subtle to Test?	130
	Uniform Random Number Generators	131

	Nonuniform Random Number Generators	132
	A Progression of Tests	134
	Conclusions	141
<b>11</b>	<b>CHANGE-CENTRIC TESTING</b>	<b>143</b>
	<i>by Murali Nandigama</i>	
	How to Set Up the Document-Driven, Change-Centric Testing Framework?	145
	Change-Centric Testing for Complex Code Development Models	146
	What Have We Learned So Far?	152
	Conclusions	154
<b>12</b>	<b>SOFTWARE IN USE</b>	<b>155</b>
	<i>by Karen N. Johnson</i>	
	A Connection to My Work	156
	From the Inside	157
	Adding Different Perspectives	159
	Exploratory, Ad-Hoc, and Scripted Testing	161
	Multuser Testing	163
	The Science Lab	165
	Simulating Real Use	166
	Testing in the Regulated World	168
	At the End	169
<b>13</b>	<b>SOFTWARE DEVELOPMENT IS A CREATIVE PROCESS</b>	<b>171</b>
	<i>by Chris McMahon</i>	
	Agile Development As Performance	172
	Practice, Rehearse, Perform	173
	Evaluating the Ineffable	174
	Two Critical Tools	174
	Software Testing Movements	176
	The Beauty of Agile Testing	177
	QA Is Not Evil	178
	Beauty Is the Nature of This Work	179
	References	179
<b>14</b>	<b>TEST-DRIVEN DEVELOPMENT: DRIVING NEW STANDARDS OF BEAUTY</b>	<b>181</b>
	<i>by Jennitta Andrea</i>	
	Beauty As Proportion and Balance	181
	Agile: A New Proportion and Balance	182
	Test-Driven Development	182
	Examples Versus Tests	184
	Readable Examples	185
	Permanent Requirement Artifacts	186
	Testable Designs	187
	Tool Support	189
	Team Collaboration	192
	Experience the Beauty of TDD	193
	References	194

<b>15</b>	<b>BEAUTIFUL TESTING AS THE CORNERSTONE OF BUSINESS SUCCESS</b>	<b>195</b>
	<i>by Lisa Crispin</i>	
	The Whole-Team Approach	197
	Automating Tests	199
	Driving Development with Tests	202
	Delivering Value	206
	A Success Story	208
	Post Script	208
<b>16</b>	<b>PEELING THE GLASS ONION AT SOCIALTEXT</b>	<b>209</b>
	<i>by Matthew Heusser</i>	
	It's Not Business... It's Personal	209
	Tester Remains On-Stage; Enter Beauty, Stage Right	210
	Come Walk with Me, The Best Is Yet to Be	213
	Automated Testing Isn't	214
	Into Socialtext	215
	A Balanced Breakfast Approach	227
	Regression and Process Improvement	231
	The Last Pieces of the Puzzle	231
	Acknowledgments	233
<b>17</b>	<b>BEAUTIFUL TESTING IS EFFICIENT TESTING</b>	<b>235</b>
	<i>by Adam Goucher</i>	
	SLIME	235
	Scripting	239
	Discovering Developer Notes	240
	Oracles and Test Data Generation	241
	Mindmaps	242
	Efficiency Achieved	244
<hr/>		
<b>Part Three</b>	<b>BEAUTIFUL TOOLS</b>	
<b>18</b>	<b>SEEDING BUGS TO FIND BUGS: BEAUTIFUL MUTATION TESTING</b>	<b>247</b>
	<i>by Andreas Zeller and David Schuler</i>	
	Assessing Test Suite Quality	247
	Watching the Watchmen	249
	An AspectJ Example	252
	Equivalent Mutants	253
	Focusing on Impact	254
	The Javalanche Framework	255
	Odds and Ends	255
	Acknowledgments	256
	References	256
<b>19</b>	<b>REFERENCE TESTING AS BEAUTIFUL TESTING</b>	<b>257</b>
	<i>by Clint Talbert</i>	
	Reference Test Structure	258

	Reference Test Extensibility	261
	Building Community	266
<b>20</b>	<b>CLAM ANTI-VIRUS: TESTING OPEN SOURCE WITH OPEN TOOLS</b>	<b>269</b>
	<i>by Tomasz Kojm</i>	
	The Clam Anti-Virus Project	270
	Testing Methods	270
	Summary	283
	Credits	283
<b>21</b>	<b>WEB APPLICATION TESTING WITH WINDMILL</b>	<b>285</b>
	<i>by Adam Christian</i>	
	Introduction	285
	Overview	286
	Writing Tests	286
	The Project	292
	Comparison	293
	Conclusions	293
	References	294
<b>22</b>	<b>TESTING ONE MILLION WEB PAGES</b>	<b>295</b>
	<i>by Tim Riley</i>	
	In the Beginning...	296
	The Tools Merge and Evolve	297
	The Nitty-Gritty	299
	Summary	301
	Acknowledgments	301
<b>23</b>	<b>TESTING NETWORK SERVICES IN MULTIMACHINE SCENARIOS</b>	<b>303</b>
	<i>by Isaac Clerencia</i>	
	The Need for an Advanced Testing Tool in eBox	303
	Development of ANSTE to Improve the eBox QA Process	304
	How eBox Uses ANSTE	307
	How Other Projects Can Benefit from ANSTE	315
<b>A</b>	<b>CONTRIBUTORS</b>	<b>317</b>
	INDEX	323

# Preface

**I DON'T THINK BEAUTIFUL TESTING COULD HAVE BEEN PROPOSED**, much less published, when I started my career a decade ago. Testing departments were unglamorous places, only slightly higher on the corporate hierarchy than front-line support, and filled with unhappy drones doing rote executions of canned tests.

There were glimmers of beauty out there, though.

Once you start seeing the glimmers, you can't help but seek out more of them. Follow the trail long enough and you will find yourself doing testing that is:

- Fun
- Challenging
- Engaging
- Experiential
- Thoughtful
- Valuable

Or, put another way, beautiful.

Testing as a recognized practice has, I think, become a lot more beautiful as well. This is partly due to the influence of ideas such as test-driven development (TDD), agile, and craftsmanship, but also the types of applications being developed now. As the products we develop and the

ways in which we develop them become more social and less robotic, there is a realization that testing them doesn't have to be robotic, or ugly.

Of course, beauty is in the eye of the beholder. So how did we choose content for *Beautiful Testing* if everyone has a different idea of beauty?

Early on we decided that we didn't want to create just another book of dry case studies. We wanted the chapters to provide a peek into the contributors' views of beauty and testing. *Beautiful Testing* is a collection of chapter-length essays by over 20 people: some testers, some developers, some who do both. Each contributor understands and approaches the idea of beautiful testing differently, as their ideas are evolving based on the inputs of their previous and current environments.

Each contributor also waived any royalties for their work. Instead, all profits from *Beautiful Testing* will be donated to the UN Foundation's Nothing But Nets campaign. For every \$10 in donations, a mosquito net is purchased to protect people in Africa against the scourge of malaria. Helping to prevent the almost one million deaths attributed to the disease, the large majority of whom are children under 5, is in itself a Beautiful Act. Tim and I are both very grateful for the time and effort everyone put into their chapters in order to make this happen.

## How This Book Is Organized

While waiting for chapters to trickle in, we were afraid we would end up with different versions of "this is how you test" or "keep the bar green." Much to our relief, we ended up with a diverse mixture. Manifestos, detailed case studies, touching experience reports, and war stories from the trenches—*Beautiful Testing* has a bit of each.

The chapters themselves almost seemed to organize themselves naturally into sections.

### Part I, Beautiful Testers

Testing is an inherently human activity; someone needs to think of the test cases to be automated, and even those tests can't think, feel, or get frustrated. *Beautiful Testing* therefore starts with the human aspects of testing, whether it is the testers themselves or the interactions of testers with the wider world.

#### Chapter 1, *Was It Good for You?*

Linda Wilkinson brings her unique perspective on the tester's psyche.

#### Chapter 2, *Beautiful Testing Satisfies Stakeholders*

Rex Black has been satisfying stakeholders for 25 years. He explains how that is beautiful.

#### Chapter 3, *Building Open Source QA Communities*

Open source projects live and die by their supporting communities. Clint Talbert and Martin Schröder share their experiences building a beautiful community of testers.

#### *Chapter 4, Collaboration Is the Cornerstone of Beautiful Performance Testing*

Think performance testing is all about measuring speed? Scott Barber explains why, above everything else, beautiful performance testing needs to be collaborative.

## **Part II, Beautiful Process**

We then progress to the largest section, which is about the testing process. Chapters here give a peek at what the test group is doing and, more importantly, why.

#### *Chapter 5, Just Peachy: Making Office Software More Reliable with Fuzz Testing*

To Kamran Khan, beauty in office suites is in hiding the complexity. Fuzzing is a test technique that follows that same pattern.

#### *Chapter 6, Bug Management and Test Case Effectiveness*

Brian Nitz and Emily Chen believe that how you track your test cases and bugs can be beautiful. They use their experience with OpenSolaris to illustrate this.

#### *Chapter 7, Beautiful XMPP Testing*

Remko Tronçon is deeply involved in the XMPP community. In this chapter, he explains how the XMPP protocols are tested and describes their evolution from ugly to beautiful.

#### *Chapter 8, Beautiful Large-Scale Test Automation*

Working at Microsoft, Alan Page knows a thing or two about large-scale test automation. He shares some of his secrets to making it beautiful.

#### *Chapter 9, Beautiful Is Better Than Ugly*

Beauty has always been central to the development of Python. Neal Noritz, Michelle Levesque, and Jeffrey Yasskin point out that one aspect of beauty for a programming language is stability, and that achieving it requires some beautiful testing.

#### *Chapter 10, Testing a Random Number Generator*

John D. Cook is a mathematician and applies a classic definition of beauty, one based on complexity and unity, to testing random number generators.

#### *Chapter 11, Change-Centric Testing*

Testing code that has not changed is neither efficient nor beautiful, says Murali Nandigama; however, change-centric testing is.

#### *Chapter 12, Software in Use*

Karen N. Johnson shares how she tested a piece of medical software that has had a direct impact on her nonwork life.

#### *Chapter 13, Software Development Is a Creative Process*

Chris McMahon was a professional musician before coming to testing. It is not surprising, then, that he thinks beautiful testing has more to do with jazz bands than manufacturing organizations.

#### *Chapter 14, Test-Driven Development: Driving New Standards of Beauty*

Jennitta Andrea shows how TDD can act as a catalyst for beauty in software projects.

### *Chapter 15, Beautiful Testing As the Cornerstone of Business Success*

Lisa Crispin discusses how a team's commitment to testing is beautiful, and how that can be a key driver of business success.

### *Chapter 16, Peeling the Glass Onion at Socialtext*

Matthew Heusser has worked at a number of different companies in his career, but in this chapter we see why he thinks his current employer's process is not just good, but beautiful.

### *Chapter 17, Beautiful Testing Is Efficient Testing*

Beautiful testing has minimal retesting effort, says Adam Goucher. He shares three techniques for how to reduce it.

## **Part III, Beautiful Tools**

*Beautiful Testing* concludes with a final section on the tools that help testers do their jobs more effectively.

### *Chapter 18, Seeding Bugs to Find Bugs: Beautiful Mutation Testing*

Trust is a facet of beauty. The implication is that if you can't trust your test suite, then your testing can't be beautiful. Andreas Zeller and David Schuler explain how you can seed artificial bugs into your product to gain trust in your testing.

### *Chapter 19, Reference Testing As Beautiful Testing*

Clint Talbert shows how Mozilla is rethinking its automated regression suite as a tool for anticipatory and forward-looking testing rather than just regression.

### *Chapter 20, Clam Anti-Virus: Testing Open Source with Open Tools*

Tomasz Kojm discusses how the ClamAV team chooses and uses different testing tools, and how the embodiment of the KISS principle is beautiful when it comes to testing.

### *Chapter 21, Web Application Testing with Windmill*

Adam Christian gives readers an introduction to the Windmill project and explains how even though individual aspects of web automation are not beautiful, their combination is.

### *Chapter 22, Testing One Million Web Pages*

Tim Riley sees beauty in the evolution and growth of a test tool that started as something simple and is now anything but.

### *Chapter 23, Testing Network Services in Multimachine Scenarios*

When trying for 100% test automation, the involvement of multiple machines for a single scenario can add complexity and non-beauty. Isaac Clerencia showcases ANSTE and explains how it can increase beauty in this type of testing.

Beautiful Testers following a Beautiful Process, assisted by Beautiful Tools, makes for Beautiful Testing. Or at least we think so. We hope you do as well.

## Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Beautiful Testing*, edited by Tim Riley and Adam Goucher. Copyright 2010 O'Reilly Media, Inc., 978-0-596-15981-8."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at [permissions@oreilly.com](mailto:permissions@oreilly.com).

## Safari® Books Online



Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at <http://my.safaribooksonline.com>.

## How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.  
1005 Gravenstein Highway North  
Sebastopol, CA 95472  
800-998-9938 (in the United States or Canada)  
707-829-0515 (international or local)  
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

<http://oreilly.com/catalog/9780596159818>

To comment or ask technical questions about this book, send email to:

[bookquestions@oreilly.com](mailto:bookquestions@oreilly.com)

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our website at:

<http://oreilly.com>

## Acknowledgments

We would like to thank the following people for helping make *Beautiful Testing* happen:

- Dr. Greg Wilson. If he had not written *Beautiful Code*, we would never have had the idea nor a publisher for *Beautiful Testing*.
- All the contributors who spent many hours writing, rewriting, and sometimes rewriting again their chapters, knowing that they will get nothing in return but the satisfaction of helping prevent the spread of malaria.
- Our technical reviewers: Kent Beck, Michael Feathers, Paul Carvalho, and Gary Pollice. Giving useful feedback is sometimes as hard as receiving it, but what we got from them certainly made this book more beautiful.
- And, of course, our wives and children, who put up with us doing “book stuff” over the last year.

—Adam Goucher

## Beautiful Testing As the Cornerstone of Business Success

*Lisa Crispin*

**IN MARCH OF 2009, DURING HIS “TRAVELS IN SOFTWARE” TOUR,** Brian Marick visited our team. He interviewed several of us about our experiences working together as an agile team for more than five years. I’ve been a tester on this team since it adopted Scrum in 2003. When Brian’s interview with me was almost over, he remarked, “We’ve talked for an hour and you haven’t mentioned testing.” That made me laugh, but it was true. I’d spent all the time talking about how we work together to deliver more business value in better ways. When I first thought about the term “beautiful testing,” this synergy is what came to my mind.

People try to quantify testing in various ways. What about its qualities? I imagine that most people don’t associate the words “beautiful” and “testing” in the same sentence. For me, the beauty of testing lies in the team’s absolute commitment to quality, to doing the right thing, and to doing things right. My team at ePlan Services Inc. focuses on testing to help the business achieve healthy growth. To paraphrase Elisabeth Hendrickson, we don’t see testing as a phase. It’s an integral part of software development, equal in value to coding. Everyone on the team drives themselves to deliver the highest quality software possible, which means everyone on the team does a beautiful job of testing.

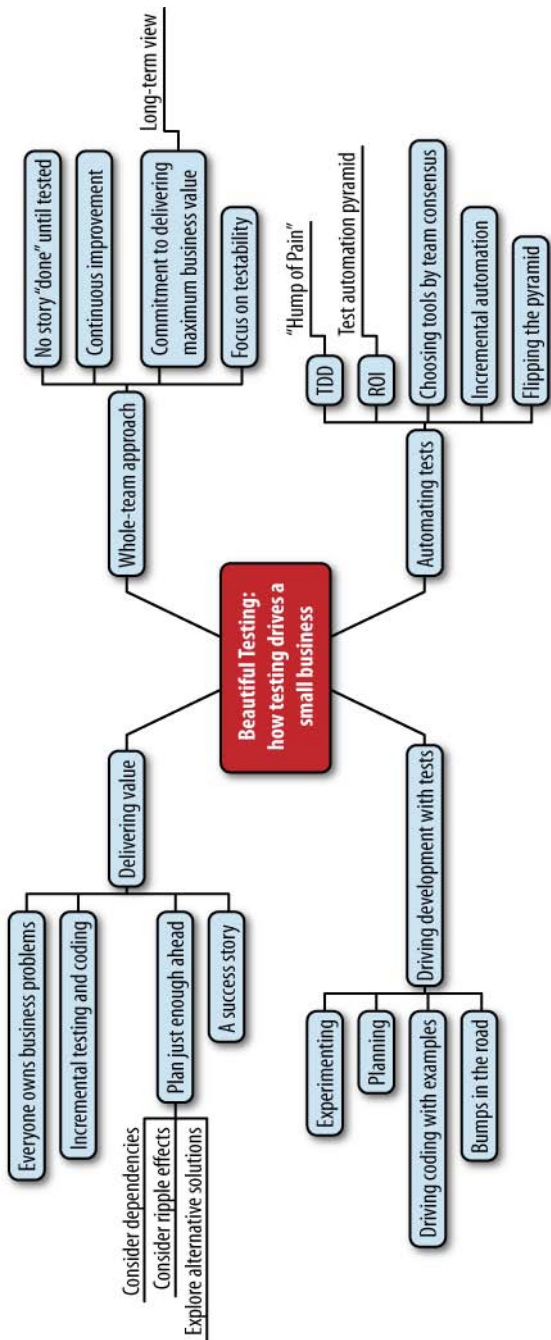


FIGURE 15-1. Mindmap of Chapter 15

My team's commitment to driving development with tests, not only with test-driven development (TDD) at the unit level, but also at the business-facing level, helped make our company a profitable and fast-growing business. Our software includes processes and features that outshine our competitors. We release new business value to production every two weeks. We work side-by-side with our business experts and customers, contributing our own ideas for business solutions. We have automated 100% of our regression testing, and these tests allow us to refactor and implement new functionality fearlessly. If you've never experienced this situation, I can tell you that it's incredibly freeing and empowering. Our team is confident that we can come up with a good solution every time.

Beautiful testing is undramatic. It's life on an even keel. It's confidence that we can deliver every sprint and not mess up. Nobody has to die.

Here is a success story that I hope will inspire you to think differently about testing. It isn't an afterthought or a safety net. It's a solid foundation for beautiful code, happy development teams, and happy customers.

## **The Whole-Team Approach**

From the first Scrum sprint, our mission was to produce the best code we could. We were fortunate to have management who believed a focus on quality, rather than speed, would pay off in the long run. We committed to driving development with tests, not only at the unit level with TDD, but at higher levels too. Inspired by ideas from Brian Marick, Robert "Uncle Bob" Martin, and others, we asked our customers for examples of desired behavior for each theme (aka epic or feature set) and story, and turned these into executable business-facing tests that also guided coding. We worked to automate 100% of our regression tests so that we could devote plenty of time to end-to-end, exploratory testing.

## **No Story Is "Done" Until It's Tested**

In our early sprints, I was daunted to hear programmers say during the scrums, "I'm done with Story XYZ," when I knew that code hadn't been tested, had no automated tests, and was not "done" in any true sense. I raised this issue at the first sprint retrospective. We wrote on our storyboard, which at the time was a big whiteboard, "No story is done until it's tested," and then we lived by this rule.

Since our team was new to TDD, we wrote task cards for writing unit tests until they became an ingrained habit and part of every coding task. We also had a column on our storyboard where we indicated whether high-level tests had been written for the story. Coding task cards weren't moved into "Work in Progress" until the high-level acceptance tests were available on the wiki.

## Continuous Improvement

Our most important tool was the retrospective we had at the beginning of every sprint. This was a great chance to solve testing issues. Like many (if not most) teams, we felt the pain of testing being squeezed to the end of the iteration. Testing tasks, especially the test automation tasks, were being carried over to the next iteration.

In our retrospectives, we came up with some rules for ourselves to help address these issues:

- All high-level acceptance tests must be written up on the wiki, usually in a list or bullet-point form, by day four of the two-week iteration.
- The team must focus on completing one story at a time.
- A story must be ready for the testers to test by day four of the iteration.
- No new functionality can be checked in on the last day of the iteration.

These rules have become a natural part of our process. Anytime we find testing tasks lagging behind coding, we check to see whether we've neglected our guidelines or need new ones. We challenge ourselves to set the bar higher. In our early days doing continuous integration, our goal was a stable build by day 12 of the iteration. Now we have a stable build every day. Reaching such a goal was a thrill, and testing continuously along with coding got us there.

Every six months we set our team goals. For example, when we decided to measure unit test coverage, we set a goal to select a tool and get a baseline of coverage. For the next six months, our goal was to achieve 70% coverage. When we met that, we set a new goal for the next six months to improve it to 73%. Every sprint, we evaluate our progress and experiment with new ideas or approaches as needed. Is our test coverage good enough? Are there more production support requests than usual? Is there a performance problem in production? At one time, the ability to release on any day, not just the end of the sprint, seemed like an unachievable dream. Now we can respond quickly to any business emergencies, or help the business take advantage of opportunities, by releasing mid-sprint if desired. We can take the time we need to think through every business problem and come up with a good solution. That's a lot prettier than hacking quick fixes into a scary code base.

## Delivering Maximum Business Value

Our company's business model depended on implementing the right software in a timely manner. We were a small startup that wasn't yet profitable. Our team had to find ways to optimize the company's return on the software investment.

Some organizations turn to agile development because they think it will let them "go faster." Agile values and practices will allow you to deliver business value more frequently, but only if you take the long view. For example, learning TDD is a big investment, but having good automated test coverage and well-designed code allows the team to deliver features more quickly.

## Focusing on Testability

Our legacy application was unstable and buggy. We were determined to start producing high-quality software. Since we had no automated regression tests in late 2003, I wrote manual test scripts for all the critical functionality of the web application. The entire team—including programmers, the DBA, the system administrator, and the Scrum master—spent the last day or two of each two-week sprint executing these manual tests. It’s impressive how this activity motivates team members to design code for testability, and to investigate good ways to automate regression testing at different levels.

We were committed to using test-driven development at the unit level, as well as using higher-level, business-facing tests to guide coding. We intended to automate 100% of our regression tests in order to keep technical debt to a minimum, get immediate feedback about whether code changes break something, and allow continual refactoring. Most importantly, automating all regression tests means more time for critical exploratory testing. There’s a special beauty in catching an unexpected ripple effect from a new feature in time to correct it well before releasing.

## Automating Tests

After a few weeks of research and discussion, the team decided that going forward we’d write all new code in a layered architecture, designed with automated testing in mind. As with many (if not all) teams new to TDD, our programmers found it hard to write unit tests for the legacy code, where business and presentation logic was mixed up with database access. In fact, it was just hard to do TDD, period. Brian Marick refers to this phenomenon as the “Hump of Pain” (see [Figure 15-2](#)). As the team became proficient in TDD, the programmers started writing unit tests whenever they changed legacy code, as well as when they coded in the new architecture.

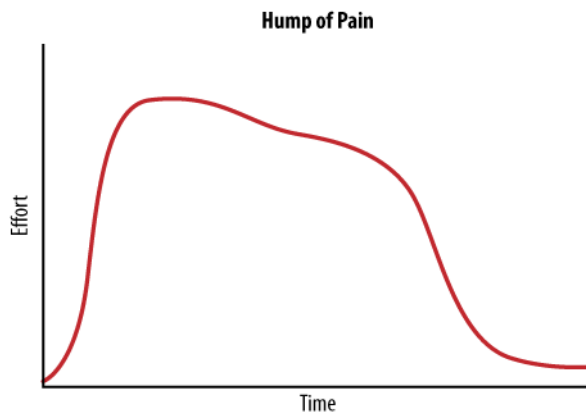


FIGURE 15-2. The Hump of Pain

My team liked Mike Cohn’s “test automation pyramid” idea (Figure 15-3). We knew our best return on investment (ROI) would be from automating tests at the unit level. We were keen to do the bulk of our functional test automation “behind” the GUI, and had chosen FitNesse as the tool to accomplish this.

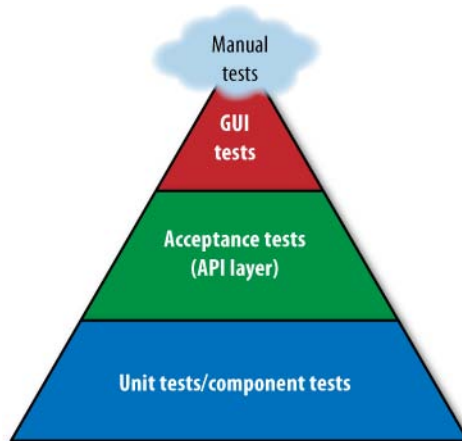


FIGURE 15-3. Test automation pyramid

Our immediate problem was a huge, buggy legacy system where the presentation, business logic, and database layers were intertwined. The fastest way to get some automated regression test coverage was through the GUI. But GUI tests are notoriously brittle and hard to maintain. We had to find good tools to help us achieve a reasonable ROI.

## Choosing Tools by Team Consensus

Since our entire development team takes responsibility for quality and for making sure all testing activities are successfully completed for each release, we choose tools as a team. This applies to both development and test frameworks.

Here’s an example of one of our team tool decisions. In early 2004, commercial test automation frameworks were not very programmer-friendly. Most used a proprietary scripting language. We’re a Java shop, and the programmers don’t want to work with another programming language. JUnit was the obvious choice for TDD. We had decided to use CruiseControl for our continuous integration build process, and CruiseControl uses Ant.

I had heard good things about Canoo WebTest, a test framework in which one specifies tests as Ant scripts. WebTest drives tests through a simulated browser, not an actual one, but our thin client layer doesn’t experience many browser incompatibilities. We liked the way it reported results with our build output. Everyone on our team was comfortable with this

approach, so we went with WebTest for our GUI test automation. It's one of several test automation frameworks we use now.

Amazing new tools come out every week. If practitioners we respect recommend a new tool, and it potentially offers valuable new features, we'll budget time to try it. We always challenge the tools we have and see whether there's something better. If a tool's ROI isn't high enough, we shouldn't keep using it. If a new tool offers more value for the amount of work and expense to get it up to speed, we should switch. We wouldn't abandon existing automated regression tests, but we might start creating new ones in a different framework. Each team member is free to question the tool we're using or propose a new one. It's a team decision whether to try out or adopt a new test tool. The freedom to experiment and share opinions means a better testing infrastructure, and more beautiful testing.

Team tool decisions apply to more than test frameworks. The defect tracking system, the wiki, the voice response technology, all of these choices are a team effort. As we used the pyramid to guide our automation choices, we use agile principles and values to guide our other tool selections. Simplicity, speed and clarity of feedback, and supporting collaboration are among the most important considerations when we evaluate a new addition to our infrastructure.

## **Incremental Automation**

When you start with no automated tests, achieving a goal of 100% regression test automation means beginning with baby steps. While the programmers were busy mastering TDD, I started building an automated GUI smoke test suite to replace our laborious manual regression testing.

I asked our customers for a prioritized list of the critical areas of our application that must always work. Each sprint, I added a few WebTest scripts. A teammate helped me configure our build process to automatically run the tests. With each iteration, we spent less time executing manual regression scripts. After about eight months, we no longer had to do any manual regression testing on the old code, and had automated tests at both the unit and GUI level for each new story. That was a beautiful feeling.

At first, the WebTest scripts ran in the continuous build process along with the few JUnits we had. As we got more WebTest scripts and more JUnit tests, we moved the WebTest scripts, which are much slower than unit tests, to a "nightly" build.

## **Flipping the Pyramid**

Like many new teams, our test automation triangle was upside down. To borrow a term from [Patrick Wilson-Welsh](#), we needed to flip our test triangle right side up.

As soon as the programmers had traction on TDD and some bandwidth to devote to a new layer of tests, it seemed like time to start using FitNesse for functional testing. I went to a programmer and asked, "Will you help me write FitNesse tests for this story you're working on?" He did, found it easy to do and useful, and told the rest of the team. After a few more

iterations, each programmer had tried FitNesse and found it worthwhile. We started writing task cards for FitNesse tests for every story that was “FitNesse-able.”

As we built up suites of FitNesse regression tests, our test automation pyramid started to change shape. For a period of time it became more of a diamond than a triangle, bulging out in the middle FitNesse layer. As more and more code was implemented with TDD in the new architecture, the unit level base of the pyramid outgrew the upper layers. We kept writing WebTest smoke test scripts as well, but they were an ever-smaller percentage of our regression tests.

As of April 2009, we have 3,864 unit tests, 474 FitNesse tests, and 99 WebTest scripts. We’ve flipped our test automation pyramid the right way around. Each “test” contains many assertions, so this multiplies out to many thousands of automated regression tests.

These tests are run in multiple continuous builds. The build that runs the JUnit tests takes about eight minutes. This build may go several days without failing, or it might fail multiple times a day; it depends on the stories underway, and whether someone’s doing a big refactoring or working on code used in multiple areas. The person who “broke the build” jumps right into fixing it, and if he needs help, the team is ready.

Complete feedback from the functional and GUI tests, which we call the “full build,” takes a couple of hours. Generally, regression tests will be caught several times during the sprint. We’re happy those bugs aren’t making it out to production, and we fix them immediately. Every two hours, we know whether or not we have a stable, releasable build. This is invaluable.

No automated test suite is foolproof, but this safety net affords us the ability to refactor as needed and implement new code without fear of unexpected consequences. It keeps our technical debt to a manageable level. We have time to do manual exploratory testing and learn about possible ripple effects.

We also have a few dozen Watir test scripts, which add to our regression safety net, but more importantly, they help us do more exploring than we could with only manual keystrokes.

To set up a scenario for manually testing in the UI, we can run a script or two that sets up the appropriate test data, gets us to the right screen, and saves us many hours of tedium and time. We can get through many more scenarios than we could with 100% manual testing.

## **Driving Development with Tests**

Mastering TDD at the unit level seemed more straightforward than learning how to use business examples and business-facing tests to guide development. FitNesse was clearly a good tool for this purpose, but figuring out the right approach involved lots of trial and error.

## Experimenting

We found that FitNesse tests were really easy to write. In our zeal to turn business examples into executable tests, the product owner (PO) and I got carried away. Our team had a difficult, high-risk theme coming up that involved complex algorithms and had a hard deadline. The PO and I spent days writing executable FitNesse tests for different parts of this theme, well ahead of the team starting on the stories for those features.

When the first iteration of working on the theme began, the programmers' reaction to our detailed FitNesse tests was: "Huh?" They couldn't see the forest for the trees. The complex, highly detailed test scenarios didn't give them a "big picture" of what they needed to code. In addition, the test design wasn't compatible with the code's architecture.

We had to back up, provide some "big picture" examples, and rewrite all the FitNesse tests, but our team learned a good lesson. We now write only high-level acceptance test cases in advance of coding. We don't start writing detailed, executable tests until someone picks up one of the coding task cards.

There's no prescription you can follow for knowing when to write tests, how many tests to write, and how detailed they should be. Every team has to experiment together and find out what works.

## Planning

My team sees testing and coding as two parts of one software development process. It's impossible to separate beautiful testing and beautiful coding. When we plan at the theme or epic level, we consider testing as well as coding. Can we write FitNesse tests for these stories? Do we need new data to test with? Is our current test environment adequate? Do we need to work with any third parties? Theme and story estimates include time for testing activities. During iteration planning, we write test task cards alongside development task cards. We use the Agile Testing Quadrants (Figure 15-4) to help us think of all the types of testing, when we need to do them, and what tools we'll need.

When we start a new project or theme, we usually start by working through examples, mocking up prototypes, and thinking about high-level tests that will guide development. These are among the tests in Q2. The Q1 tests are a given, since we'll use TDD, but it's easy to forget about Q3 and Q4 when we're planning. It might take advance planning to implement the right tools for activities such as performance testing, and we have to be sure to budget enough time for adequate exploratory testing.

We're fortunate that Steve, our product owner, gets what he calls "advance clarity" about each story from all the business stakeholders. He writes up their conditions of satisfaction and goes through his own checklist for each story, and considers things such as legal issues, impact on third parties, and whether any new reports are needed. He often writes a few high-level test cases. He understands that we use tests to drive coding, and helps us prepare for these tasks

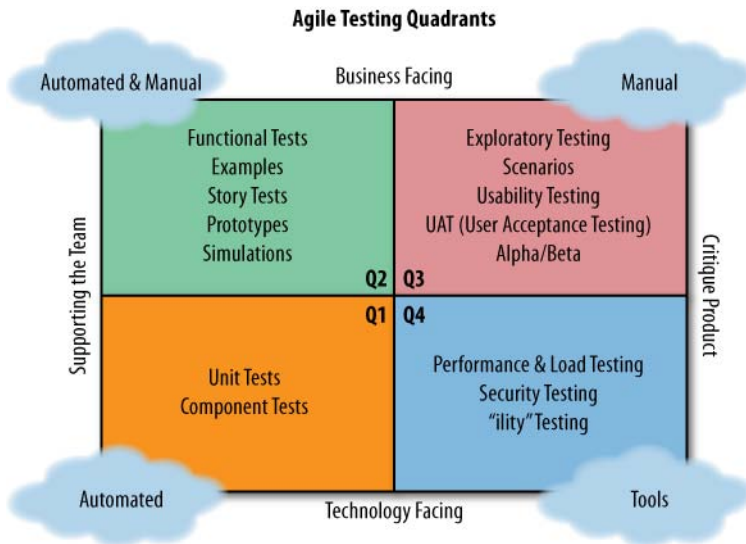


FIGURE 15-4. Agile Testing Quadrants

by giving us real-life examples that we can turn into executable tests. Such strong collaboration between the business and technical teams is part of beautiful testing.

When a complex theme is coming up, we hold hour-long brainstorming sessions where we and the customers work through example scenarios on the whiteboard and think about the best way to design the system. Everyone on the team feels free to ask questions and contribute ideas. It's sometimes a tester who comes up with a simpler solution than the product owner had proposed.

## Driving Coding with Examples

Regression tests at the GUI level were a key to our success, especially early on. Once we no longer had to devote time to manual regression testing, we were able to start using FitNesse to convert examples of desired system behavior into tests that drive development.

As mentioned earlier, our product owner supplied us with useful real-life examples that we could turn into executable tests. These were in the form of either whiteboard drawings or Excel spreadsheets. We turned these into executable FitNesse tests, such as the one seen in [Figure 15-5](#).

Also as mentioned earlier, we consider testing when we plan big new themes. Some new features might be straightforward to code but not so easy to test. During sprint planning, we write task cards for all testing tasks. Here's a typical set of task cards for a story to process an incoming loan payment:

Test two payments in the same day.

Loan Processing Fixture								
take loan in the amount of	1000	with interest rate	6.0	frequency	Monthly	and term	1 year with loan origination date	10-01-2005
check	periodic payment is	86.07						
post payment	1	of	86.07	on	10-31-2005			
post payment	2	of	86.07	on	10-31-2005			
receive payment	1	of	86.07	on	11-01-2005			
receive payment	2	of	86.07	on	11-01-2005			
settle and confirm payment	1							
settle and confirm payment	2							
check	interest applied for	1	is	5.10				
check	principal applied for	1	is	80.97				
check	principal applied for	2	is	86.07				
check	loan balance is	832.96						

FIGURE 15-5. Example FitNesse test

- Write high-level test cases for applying loan payment interest and principal
- Write FitNesse test cases for applying loan payment interest and principal
- Write FitNesse fixtures to automate tests for applying loan payment
- Write Canoo WebTest tests for processing loan payment
- Manually test processing loan payments

Have I mentioned that anyone on the team can sign up for any testing task? Because our application is so testing-intensive, even though we have two testers for five programmers, the programmers often pick up testing tasks, such as writing FitNesse tasks. And it's not unusual for them to use FitNesse tests for their own test-driven development.

## Bumps in the Road

Of course, there are bumps in our automation road. Even after five years, a hard-to-solve problem can trip up our team. Here's a recent example. We had fallen behind almost two years in our version of FitNesse, and finally upgraded to the latest version. The new version no longer produced HTML result files, and the parser used by our continuous build to produce test results was no longer of any use.

I spent days trying to get the new version integrated with our continuous build, but couldn't produce the right result files. At the same time, a teammate was converting our continuous build processes from CruiseControl to Hudson. In the end, we worked together to get the FitNesse suites running as part of the Hudson build processes. My main contribution was to get help from the testing community in the form of a Hudson plug-in for FitNesse and a stylesheet to get the results into the necessary format. My teammate, a system administrator and programmer, did the heavy lifting. Neither of us could have succeeded as quickly working independently.

Our team gives the same attention to automated test design as we do to production code design. Good coding and design practices are as essential to test scripts as to any production code. We

continually refactor our tests to minimize our maintenance costs. If there's a change in functionality, good test design means that updating the tests is a simple task.

Our investment in test design pays off multiple ways. Our automated tests provide incontrovertible documentation of the system behavior. Often, one of our business people will come over with a question such as, "I can't remember how the interest portion of a loan payment is calculated if a loan payment was skipped." I can run a FitNesse test to show conclusively how the functionality is implemented. That saves hours of discussion about "I thought we designed it to work this other way."

## **Delivering Value**

When Brian Marick interviewed me for "Travels in Software," he asked me an interesting question: "What would you go back and tell your 2003 self?" The most important thing I wish I had known back when our team first implemented Scrum was that we really needed to understand the business inside and out. Not only the parts of the business automated in our software, but all the manual operations, accounting, marketing, and sales—everything that makes the business succeed.

## **Everyone Owns Business Problems**

A year into agile development, our team effectively used TDD to produce robust, stable code. We were automating 100% of our regression tests, and devoting significant time to exploratory testing. Too often, though, we failed to deliver the exact functionality the customers wanted. We missed or misunderstood requirements, especially in areas outside the main application, such as accounting. We sometimes missed ripple effects that changes in one part of the system had on other, seemingly unrelated parts.

Problems not caused by software defects also troubled us. Application users made mistakes that had to be corrected with manual data changes. How could we make the system more "foolproof," and could we provide software to allow users to unravel their own messes? How could we maximize the ROI of the software we delivered every two weeks?

We decided to budget time to study the business more closely. Sitting with our customers as they performed their daily processing, we found simple software solutions that could save them hours of time every week and reduce the possibility of manual errors. For example, adding some information to an existing screen was easy to do and saved significant research time. Software to perform some complex calculations, which up to then had been done manually, took a couple of iterations to deliver. Because the business people are able to make more accurate operations, they don't come to us as often needing manual corrections in the database. Understanding the entire accounting system, not just the parts done by the application, was critical to ensuring that accounts balance correctly.

Here's another example of what I learned when I dedicated extra time and effort to learning the business. I worked at ePlan for more than five years without understanding that money moved among five different accounts. It's critical that the cash balances of these accounts balance every day. Learning this allowed me to "get" why our accountant might get a bit nervous when one account is out of balance by half a million dollars. Understanding the money movement, we could write the correct code to make sure money was withdrawn from and deposited to the correct accounts every day. It might seem obvious now, but it took us several years to achieve this level of understanding. Now we can help the business take stock accurately every day.

## **Incremental Testing and Coding**

We've always tried to break work into small chunks, and focus on completing one small piece of functionality at a time. Another thing I'd tell my 2003 self is to commit to delivering less. We all want to satisfy our customers, but we have to be realistic. Working in small increments has made us much more efficient. We break complex stories into what we call "steel threads." The first "thread" is an end-to-end slice of the most basic functionality. We build each subsequent "thread" on top of the completed ones. Each thread is testable. Often the functionality is usable even before all the threads are finished.

Here's an example. We recently had a theme that involved uploading, parsing, and validating a file, creating data and persisting it in the database, and then processing the data. Testers, programmers, and customers mocked up the system on a whiteboard. The UI consisted of four screens. Our first UI "thread" was to display each screen, with some hardcoded data and buttons to navigate from one screen to another. There wasn't any business value yet, but we could show it to the customers to make sure they liked the flow, and write an automated end-to-end GUI test. We added functionality to each screen incrementally, completing testing and test automation for each small "thread" and building on that for the next increment.

Another "thread" was to parse and validate the uploaded file. We wrote FitNesse test cases for the parsing and validation, then wrote and tested the code without needing the UI at all. Then, we specified tests for creating the data and inserting it in the database and wrote that code. We tackled the code to process the data the same way. Those slices of functionality could be completed independently of the UI slices. By the time all the code was in place, we had time for end-to-end exploratory testing, with scripts to help set up the scenarios for manual testing. The exact functionality the customers needed was delivered quickly and defect-free. This incremental approach leads to the most elegant solution.

## **Plan Just Enough Ahead**

If I tell you, "Don't plan your stories and themes too far in advance of when you will actually start developing them; priorities may change, and you want the information fresh in your minds when you start working on the stories," it doesn't sound like I'm talking about testing.

That's because I can't separate testing out from coding. Practices and principles that improve testing usually improve coding, and vice versa.

Our development team's depth of business knowledge makes planning more efficient. We can provide useful input when the business starts to plan a new feature. We can ask good questions that help them flesh out their requirements. We often suggest alternative solutions that might cost less to develop. We can help business experts decide whether a particular functionality will have a good ROI, so that we don't waste time planning features that don't deliver enough value.

## **A Success Story**

When our team first implemented Scrum, the business had about 500 customers, wasn't profitable, and was in danger of going under because we couldn't deliver what our customers needed. As of this writing, we have well over 3,000 customers, we've been profitable for a couple of years, and in spite of the current unhealthy economy, we're growing steadily. Guiding development with tests, combined with our domain knowledge, helps ensure that we produce the right software. Our tests were designed to ensure that we code the right features the right way, and as a bonus, they also provide a super safety net. Our team's long-term commitment to designing software the right way helps us deliver value quickly and frequently. Our business stakeholders can count on our team's ability to change and grow our product, delighting existing customers and attracting new ones. Our beautiful team approach to testing and coding means joy and satisfaction for our team, our business, and our customers.

## **Post Script**

I wrote the first draft of this chapter as a tester for ePlan Services Inc. I subsequently accepted an opportunity at Ultimate Software Group. I'm pleased to report that my experiences thus far with my new team echo my experiences at ePlan.

At the time I joined them, Ultimate had been doing Scrum for four years, and was delivering four production releases per year with 28 (!) Scrum teams. My new team is committed to driving development with tests at the unit and acceptance level. We work hard with the product folks to understand how our clients will use our software. Seeing the process work so similarly to what my team at ePlan did is a huge affirmation.

The common threads are the teams' focus on the customers' needs and on delivering what they want at the right time. Driving the coding with tests and examples helps ensure that our code base will accommodate all future changes. The business has confidence that new software features are delivered on time and continue to produce value.

## Contributors

**JENNITTA ANDREA** has been a multifaceted, hands-on practitioner (analyst, tester, developer, manager), and coach on over a dozen different types of agile projects since 2000. Naturally a keen observer of teams and processes, Jennitta has published many experience-based papers for conferences and software journals, and delivers practical, simulation-based tutorials and in-house training covering agile requirements, process adaptation, automated examples, and project retrospectives. Jennitta's ongoing work has culminated in international recognition as a thought leader in the area of agile requirements and automated examples. She is very active in the agile community, serving a third term on the Agile Alliance Board of Directors, director of the Agile Alliance Functional Test Tool Program to advance the state of the art of automated functional test tools, member of the Advisory Board of IEEE Software, and member of many conference committees. Jennitta founded The Andrea Group in 2007 where she remains actively engaged on agile projects as a hands-on practitioner and coach, and continues to bridge theory and practice in her writing and teaching.

**SCOTT BARBER** is the chief technologist of PerfTestPlus, executive director of the Association for Software Testing, cofounder of the Workshop on Performance and Reliability, and coauthor of *Performance Testing Guidance for Web Applications* (Microsoft Press). He is widely recognized as a thought leader in software performance testing and is an international keynote speaker. A trainer of software testers, Mr. Barber is an AST-certified On-Line Lead Instructor who has authored over 100 educational articles on software testing. He is a member of ACM, IEEE, American Mensa, and the Context-Driven School of Software Testing, and is a signatory to the Manifesto for Agile Software Development. See <http://www.perftestplus.com/ScottBarber> for more information.

**REX BLACK**, who has a quarter-century of software and systems engineering experience, is president of **RBCS**, a leader in software, hardware, and systems testing. For over 15 years, RBCS has delivered services in consulting, outsourcing, and training for software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS conducts product testing, builds and improves testing groups, and hires testing staff for hundreds of clients worldwide. Ranging from Fortune 20 companies to startups, RBCS clients save time and money through improved product development, decreased tech support calls, improved corporate reputation, and more. As the leader of RBCS, Rex is the most prolific author practicing in the field of software testing today. His popular first book, *Managing the Testing Process* (Wiley), has sold over 35,000 copies around the world, including Japanese, Chinese, and Indian releases, and is now in its third edition. His five other books on testing, *Advanced Software Testing: Volume I*, *Advanced Software Testing: Volume II* (Rocky Nook), *Critical Testing Processes* (Addison-Wesley Professional), *Foundations of Software Testing* (Cengage), and *Pragmatic Software Testing* (Wiley), have also sold tens of thousands of copies, including Hebrew, Indian, Chinese, Japanese, and Russian editions. He has written over 30 articles, presented hundreds of papers, workshops, and seminars, and given about 50 keynotes and other speeches at conferences and events around the world. Rex has also served as the president of the International Software Testing Qualifications Board and of the American Software Testing Qualifications Board.

**EMILY CHEN** is a software engineer working on OpenSolaris desktop. Now she is responsible for the quality of Mozilla products such as Firefox and Thunderbird on OpenSolaris. She is passionate about open source. She is a core contributor of the OpenSolaris community, and she worked on the Google Summer of Code program as a mentor in 2006 and 2007. She organized the first-ever GNOME.Asia Summit 2008 in Beijing and founded the Beijing GNOME Users Group. She graduated from the Beijing Institute of Technology with a master's degree in computer science. In her spare time, she likes snowboarding, hiking, and swimming.

**ADAM CHRISTIAN** is a JavaScript developer doing test automation and AJAX UI development. He is the cocreator of the Windmill Testing Framework, Mozmill, and various other open source projects. He grew up in the northwest as an avid hiker, skier, and sailer and attended Washington State University studying computer science and business. His personal blog is at <http://www.adamchristian.com>. He is currently employed by Slide, Inc.

**ISAAC CLERENCIA** is a software developer at eBox Technologies. Since 2001 he has been involved in several free software projects, including Debian and Battle for Wesnoth. He, along with other partners, founded Warp Networks in 2004. Warp Networks is the open source-oriented software company from which eBox Technologies was later spun off. Other interests of his are artificial intelligence and natural language processing.

**JOHN D. COOK** is a very applied mathematician. After receiving a Ph.D. in from the University of Texas, he taught mathematics at Vanderbilt University. He then left academia to work as a software developer and consultant. He currently works as a research statistician at M. D. Anderson Cancer Center. His career has been a blend of research, software development,

consulting, and management. His areas of application have ranged from the search for oil deposits to the search for a cure for cancer. He lives in Houston with his wife and four daughters. He writes a blog at <http://www.johndcook.com/blog>.

**LISA CRISPIN** is an agile testing coach and practitioner. She is the coauthor, with Janet Gregory, of *Agile Testing: A Practical Guide for Testers and Agile Teams* (Addison-Wesley). She works as the director of agile software development at Ultimate Software. Lisa specializes in showing testers and agile teams how testers can add value and how to guide development with business-facing tests. Her mission is to bring agile joy to the software testing world and testing joy to the agile development world. Lisa joined her first agile team in 2000, having enjoyed many years working as a programmer, analyst, tester, and QA director. From 2003 until 2009, she was a tester on a Scrum/XP team at ePlan Services, Inc. She frequently leads tutorials and workshops on agile testing at conferences in North America and Europe. Lisa regularly contributes articles about agile testing to publications such as *Better Software* magazine, *IEEE Software*, and *Methods and Tools*. Lisa also coauthored *Testing Extreme Programming* (Addison-Wesley) with Tip House. For more about Lisa's work, visit <http://www.lisacrispin.com>.

**ADAM GOUCHER** has been testing software professionally for over 10 years. In that time he has worked with startups, large multinationals, and those in between, in both traditional and agile testing environments. A believer in the communication of ideas big and small, he writes frequently at <http://adam.goucher.ca> and teaches testing skills at a Toronto-area technical college. In his off hours he can be found either playing or coaching box lacrosse—and then promptly applying lessons learned to testing. He is also an active member of the Association for Software Testing.

**MATTHEW HEUSSER** is a member of the technical staff (“QA lead”) at Socialtext and has spent his adult life developing, testing, and managing software projects. In addition to Socialtext, Matthew is a contributing editor for *Software Test and Performance Magazine* and an adjunct instructor in the computer science department at Calvin College. He is the lead organizer of both the Great Lakes Software Excellence Conference and the peer workshop on Technical Debt. Matthew's blog, [Creative Chaos](#), is consistently ranked in the top-100 blogs for developers and dev managers, and the top-10 for software test automation. Equally important, Matthew is a whole person with a lifetime of experience. As a cadet, and later officer, in the Civil Air Patrol, Matthew soloed in a Cessna 172 light aircraft before he had a driver's license. He currently resides in Allegan, Michigan with his family, and has even been known to coach soccer.

**KAREN N. JOHNSON** is an independent software test consultant based in Chicago, Illinois. She views software testing as an intellectual challenge and believes in [context-driven testing](#). She teaches and consults on a variety of topics in software testing and frequently speaks at software testing conferences. She's been published in *Better Software* and *Software Test and Performance* magazines and on [InformIT.com](#) and [StickyMinds.com](#). She is the cofounder of WREST, the [Workshop on Regulated Software Testing](#). Karen is also a hosted software testing expert on [Tech Target's website](#). For more information about Karen, visit <http://www.karennjohnson.com>.

**KAMRAN KHAN** contributes to a number of open source office projects, including AbiWord (a word processor), Gnumeric (a spreadsheet program), libwpd and libwpg (WordPerfect libraries), and libgoffice and libgsf (general office libraries). He has been testing office software for more than five years, focusing particularly on bugs that affect reliability and stability.

**TOMASZ KOJM** is the original author of Clam AntiVirus, an open source antivirus solution. ClamAV is freely available under the GNU General Public License, and as of 2009, has been installed on more than two million computer systems, primarily email gateways. Together with his team, Tomasz has been researching and deploying antivirus testing techniques since 2002 to make the software meet mission-critical requirements for reliability and availability.

**MICHELLE LEVESQUE** is the tech lead of Ads UI at Google, where she works to make useful, beautiful ads on the search results page. She also writes and directs internal educational videos, teaches Python classes, leads the readability team, helps coordinate the massive posting of Google restroom stalls with weekly flyers that promote testing, and interviews potential chefs and masseuses.

**CHRIS MCMAHON** is a dedicated agile tester and a dedicated telecommuter. He has amassed a remarkable amount of professional experience in more than a decade of testing, from telecom networks to social networking, from COBOL to Ruby. A three-time college dropout and former professional musician, librarian, and waiter, Chris got his start as a software tester a little later than most, but his unique and varied background gives his work a sense of maturity that few others have. He lives in rural southwest Colorado, but contributes to a couple of magazines, several mailing lists, and is even a character in a book about software testing.

**MURALI NANDIGAMA** is a quality consultant and has more than 15 years of experience in various organizations, including TCS, Sun, Oracle, and Mozilla. Murali is a Certified Software Quality Analyst, Six Sigma lead, and senior member of IEEE. He has been awarded with multiple software patents in advanced software testing methodologies and has published in international journals and presented at many conferences. Murali holds a doctorate from the University of Hyderabad, India.

**BRIAN NITZ** has been a software engineer since 1988. He has spent time working on all aspects of the software life cycle, from design and development to QA and support. His accomplishments include development of a dataflow-based visual compiler, support of radiology workstations, QA, performance, and service productivity tools, and the successful deployment of over 7,000 Linux desktops at a large bank. He lives in Ireland with his wife and two kids where he enjoys travel, sailing, and photography.

**NEAL NORWITZ** is a software developer at Google and a Python committer. He has been involved with most aspects of testing within Google and Python, including leading the Testing Grouplet at Google and setting up and maintaining much of the Python testing infrastructure. He got deeply involved with testing when he learned how much his code sucked.

**ALAN PAGE** began his career as a tester in 1993. He joined Microsoft in 1995, and is currently the director of test excellence, where he oversees the technical training program for testers and

various other activities focused on improving testers, testing, and test tools. Alan writes about testing on his [blog](#), and is the lead author on *How We Test Software at Microsoft* (Microsoft Press). You can contact him at [alan.page@microsoft.com](mailto:alan.page@microsoft.com).

**TIM RILEY** is the director of quality assurance at Mozilla. He has tested software for 18 years, including everything from spacecraft simulators, ground control systems, high-security operating systems, language platforms, application servers, hosted services, and open source web applications. He has managed software testing teams in companies from startups to large corporations, consisting of 3 to 120 people, in six countries. He has a software patent for a testing execution framework that matches test suites to available test systems. He enjoys being a breeder caretaker for [Canine Companions for Independence](#), as well as live and studio sound engineering.

**MARTIN SCHRÖDER** studied computer science at the University of Würzburg, Germany, from which he also received his master's degree in 2009. While studying, he started to volunteer in the community-driven Mozilla Calendar Project in 2006. Since mid-2007, he has been coordinating the QA volunteer team. His interests center on working in open source software projects involving development, quality assurance, and community building.

**DAVID SCHULER** is a research assistant at the software engineering chair at Saarland University, Germany. His research interests include mutation testing and dynamic program analysis, focusing on techniques that characterize program runs to detect equivalent mutants. For that purpose, he has developed the Javalanche mutation-testing framework, which allows efficient mutation testing and assessing the impact of mutations.

**CLINT TALBERT** has been working as a software engineer for over 10 years, bouncing between development and testing at established companies and startups. His accomplishments include working on a peer-to-peer database replication engine, designing a rational way for applications to get time zone data, and bringing people from all over the world to work on testing projects. These days, he leads the Mozilla Test Development team concentrating on QA for the Gecko platform, which is the substrate layer for Firefox and many other applications. He is also an aspiring fiction writer. When not testing or writing, he loves to rock climb and surf everywhere from Austin, Texas to Ocean Beach, California.

**REMKO TRONÇON** is a member of the XMPP Standards Foundation's council, coauthor of several XMPP protocol extensions, former lead developer of Psi, developer of the Swift Jabber/XMPP project, and a coauthor of the book *XMPP: The Definitive Guide* (O'Reilly). He holds a Ph.D. in engineering (computer science) from the Katholieke Universiteit Leuven. His blog can be found at <http://el-tramo.be>.

**LINDA WILKINSON** is a QA manager with more than 25 years of software testing experience. She has worked in the nonprofit, banking, insurance, telecom, retail, state and federal government, travel, and aviation fields. Linda's blog is available at <http://practicalqa.com>, and she has been known to drop in at the forums on <http://softwaretestingclub.com> to talk to her Cohorts in Crime (i.e., other testing professionals).

**JEFFREY YASSKIN** is a software developer at Google and a Python committer. He works on the Unladen Swallow project, which is trying to dramatically improve Python's performance by compiling hot functions to machine code and taking advantage of the last 30 years of virtual machine research. He got into testing when he noticed how much it reduced the knowledge needed to make safe changes.

**ANDREAS ZELLER** is a professor of software engineering at Saarland University, Germany. His research centers on programmer productivity—in particular, on finding and fixing problems in code and development processes. He is best known for GNU DDD (Data Display Debugger), a visual debugger for Linux and Unix; for Delta Debugging, a technique that automatically isolates failure causes for computer programs; and for his work on mining the software repositories of companies such as Microsoft, IBM, and SAP. His recent work focuses on assessing and improving test suite quality, in particular mutation testing.

## COLOPHON

The cover image is from Getty Images. The cover fonts are Akzidenz Grotesk and Orator. The text font is Adobe's Meridien; the heading font is ITC Bailey.