

/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 downwards and to the left across the frame. The sand is a warm, orange-brown color.

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



Safari[®]
Books Online

Free online edition

for 45 days with purchase of
this book. Details on last page.

O'REILLY[®] 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.

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>		
Part One	BEAUTIFUL TESTERS	
<hr/>		
1	WAS IT GOOD FOR YOU?	3
	<i>by Linda Wilkinson</i>	
2	BEAUTIFUL TESTING SATISFIES STAKEHOLDERS	15
	<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
3	BUILDING OPEN SOURCE QA COMMUNITIES	27
	<i>by Martin Schröder and Clint Talbert</i>	
	Communication	27
	Volunteers	28
	Coordination	29
	Events	32
	Conclusions	35
4	COLLABORATION IS THE CORNERSTONE OF BEAUTIFUL PERFORMANCE TESTING	37
	<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
Part Two	BEAUTIFUL PROCESS	
<hr/>		
5	JUST PEACHY: MAKING OFFICE SOFTWARE MORE RELIABLE WITH FUZZ TESTING	55
	<i>by Kamran Khan</i>	
	User Expectations	55
	What Is Fuzzing?	57
	Why Fuzz Test?	57

	Fuzz Testing	60
	Future Considerations	65
6	BUG MANAGEMENT AND TEST CASE EFFECTIVENESS <i>by Emily Chen and Brian Nitz</i>	67
	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
7	BEAUTIFUL XMPP TESTING <i>by Remko Tronçon</i>	85
	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
8	BEAUTIFUL LARGE-SCALE TEST AUTOMATION <i>by Alan Page</i>	103
	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
9	BEAUTIFUL IS BETTER THAN UGLY <i>by Neal Norwitz, Michelle Levesque, and Jeffrey Yasskin</i>	119
	The Value of Stability	120
	Ensuring Correctness	121
	Conclusions	127
10	TESTING A RANDOM NUMBER GENERATOR <i>by John D. Cook</i>	129
	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
11	CHANGE-CENTRIC TESTING	143
	<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
12	SOFTWARE IN USE	155
	<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
13	SOFTWARE DEVELOPMENT IS A CREATIVE PROCESS	171
	<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
14	TEST-DRIVEN DEVELOPMENT: DRIVING NEW STANDARDS OF BEAUTY	181
	<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

15	BEAUTIFUL TESTING AS THE CORNERSTONE OF BUSINESS SUCCESS	195
	<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
16	PEELING THE GLASS ONION AT SOCIALTEXT	209
	<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
17	BEAUTIFUL TESTING IS EFFICIENT TESTING	235
	<i>by Adam Goucher</i>	
	SLIME	235
	Scripting	239
	Discovering Developer Notes	240
	Oracles and Test Data Generation	241
	Mindmaps	242
	Efficiency Achieved	244
Part Three BEAUTIFUL TOOLS		
18	SEEDING BUGS TO FIND BUGS: BEAUTIFUL MUTATION TESTING	247
	<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
19	REFERENCE TESTING AS BEAUTIFUL TESTING	257
	<i>by Clint Talbert</i>	
	Reference Test Structure	258

	Reference Test Extensibility	261
	Building Community	266
20	CLAM ANTI-VIRUS: TESTING OPEN SOURCE WITH OPEN TOOLS <i>by Tomasz Kojm</i>	269
	The Clam Anti-Virus Project	270
	Testing Methods	270
	Summary	283
	Credits	283
21	WEB APPLICATION TESTING WITH WINDMILL <i>by Adam Christian</i>	285
	Introduction	285
	Overview	286
	Writing Tests	286
	The Project	292
	Comparison	293
	Conclusions	293
	References	294
22	TESTING ONE MILLION WEB PAGES <i>by Tim Riley</i>	295
	In the Beginning...	296
	The Tools Merge and Evolve	297
	The Nitty-Gritty	299
	Summary	301
	Acknowledgments	301
23	TESTING NETWORK SERVICES IN MULTIMACHINE SCENARIOS <i>by Isaac Clerencia</i>	303
	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
A	CONTRIBUTORS	317
	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.

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

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

Peeling the Glass Onion at Socialtext

Matthew Heusser

I don't understand why we thought this was going to work in the first place.

—James Mathis, 2004

It's Not Business...It's Personal

I'VE SPENT MY ENTIRE ADULT LIFE DEVELOPING, TESTING, and managing software projects. In those years, I've learned a few things about our field:

- Software testing as it is practiced in the field bears very little resemblance to how it is taught in the classroom—or even described at some industry presentations.
- There are multiple perspectives on what good software testing is and how to do it well.
- The previous point means that there are no “best practices”—no single way to view or do testing that will allow you to be successful in all environments—but there are rules of thumb that can guide the learner.*

* I am a member of the context-driven school of software testing, a community of people who align around such ideas, including “there are no best practices.” See <http://www.context-driven-testing.com/>.

Beyond that, in business software development, I would add a few things more. First, there is a sharp difference between *checking*,[†] which is a sort of clerical, repeatable process to make sure things are fine, and *investigating*, which is a feedback-driven process.

Checking can be automated, or at least parts of it can. With small, discrete units, it is possible for a programmer to select inputs and compare them to outputs automatically. When we combine those units, we begin to see complexity.

Imagine, for example, a simple calculator program that has a very small memory leak every time we press the Clear button. It might behave fine if we test each operation independently, but when we try to use the calculator for half an hour, it seems to break down without reason.

Checking cannot find those types of bugs. Investigating might. Or, better yet, in this example, a static inspector looking for memory leaks might.

And that's the point. Software exposes us to a variety of risks. We will have to use a variety of techniques to limit those risks. Because there are no best practices, I can't tell you what to do, but I can tell you what we have done at Socialtext, and why we like it—what makes those practices beautiful to us.

Our approach positions testing as a form of risk management. The company invests a certain amount of time and money in testing in order to get information, which will decrease the chance of a bad release. There is an entire business discipline around risk management; insurance companies practice it every day. It turns out that *testing for its own sake* meets the exact definition of risk management. We'll revisit risk management when we talk about testing at Socialtext, but first, let's talk about beauty.

Tester Remains On-Stage; Enter Beauty, Stage Right

Are you skeptical yet? If you are, I can't say I blame you. To many people, the word "testing" brings up images of drop-dead simple pointing and clicking, or following a boring script written by someone else. They think it's a simple job, best done by simple people who, well...at least you don't have to pay them much. I think there's something wrong with that.

Again, the above isn't critical investigation; it's checking. And checking certainly isn't *beautiful*, by any stretch of the word. And beauty is important.

Let me explain.

In my formative years as a developer, I found that I had a conflict with my peers and superiors about the way we developed software. Sometimes I attributed this to growing up in the east coast versus the midwest, and sometimes to the fact that my degree was not in computer

[†] My colleague and friend Michael Bolton is the first person I am aware of to make this distinction, and I believe he deserves a fair amount of credit for it.

science but mathematics.[‡] So, being young and insecure, I went back to school at night and earned a Master’s degree in computer information systems to “catch up,” but still I had these cultural arguments about how to develop software. I wanted simple projects, whereas my teammates wanted projects done “right” or “extensible” or “complete.”

Then one day I realized: they had never been taught about beauty, nor that beauty was inherently good. Although I had missed a class or two in my *concentration* in computer science, they also missed something I had learned in mathematics: an appreciation of *aesthetics*. Some time later I read *Things a Computer Scientist Rarely Talks About* (Center for the Study of Language and Information) by Dr. Donald Knuth, and found words to articulate this idea. Knuth said that mathematicians and computer scientists need similar basic skills: they need to be able to keep many variables in their head, and they need to be able to jump up and down a chain of abstraction very quickly to solve complex programs. According to Knuth, the mathematician is searching for truth—ideas that are consistently and universally correct—whereas the computer scientists can simply hack a conditional[§] in and move on.

But mathematics is more than that. To solve any problem in math, you *simplify* it. Take the simple algebra problem:

$$2X - 6 = 0$$

So we add 6 to each side and get $2X = 6$, and we divide by 2 and get $X = 3$. At every step in the process, we make the equation simpler. *In fact, the simplest expression of any formula is the answer.* There may be times when you get something like $X = 2Y$; you haven’t solved for X or Y, but you’ve taken the problem down to its simplest possible form and you get full credit. And the best example of solving a problem of this nature I can think of is the *proof*.

I know, I know, please don’t fall asleep on me here or skip down. To a mathematician, a good proof is a work of art—the stuff of pure logic, distilled into symbols.^{||} Two of the highest division courses I took at Salisbury University were number theory and the history of mathematics from Dr. Homer Austin. They weren’t what you would think. Number theory was basically recreating the great proofs of history—taking a formula that seemed to make sense, proving it was true for $F(1)$, then proving if it was true for any $F(N)$, then it was also true for $F(N+1)$. That’s called proof by induction. Number theory was trying to understand how the elements

[‡] Strictly speaking, I have a bachelor’s degree in mathematics with a concentration in computer science.

[§] “Conditional” is a fancy word for an IF/THEN/ELSE statement block.

^{||} I am completely serious about the beauty of proofs. For years, I used to ask people I met with any kind of mathematics background what their favorite math proof was. Enough blank stares later and I stopped asking. As for mine, I’m stuck between two: the proof of the limit of the sum of $1/(2^N)$ for all positive integers, or Newton’s proof of integration—take your pick. (Rob Sabourin is one notable exception. I asked him his favorite, and he said he was stuck between two....)

of the universe were connected (such as the Fibonacci sequence, which appears in nature on a conch shell), or how to predict what the next prime number will be, or why pi shows up in so many places.

And, every now and again, Dr. Homer Austin would step back from the blackboard, look at the work, and just say, “Now...there’s a beautiful equation.” The assertion was simple: beauty and simplicity were inherently good.

You could tell this in your work because the simplest answer was correct. When you got the wrong answer, your professor could look at your work and show you the ugly line, the hacky line, the one line that looked more complex than the one above it. He might say, “Right there, Matt. That’s where you went off the rails.”#

By the end of the semester, we could see it too. For that, I am, quite honestly, in his debt.*

Of course, you can learn to appreciate beauty from any discipline that deals in abstraction and multiple variables. You could learn it from chess, or chemistry, or aerospace engineering, or music and the arts.† My experience was that it was largely missing from computer science, at least in the 1990s. Instead of simplicity, we celebrated *complexity*. Instead of focusing on value to customers, more senior programmers were writing the complex frameworks and architectures, leaving the junior developers to be *mere implementers*. The goal was not to deliver value quickly but instead to develop a castle in the sky. We even invented a term, “gold plating,” for when a developer found a business problem too simple and had to add his own bells and whistles to the system, or perhaps, instead of solving one problem and solving it well, he created an *extensible framework* to solve a much larger number of generic business problems.

Joel Spolsky would call this person an “architecture astronaut,” in that they get so abstract, they actually “cut off the air supply” of the business.‡ In the back of my mind I could hear the voice of Dr. Austin saying, “Right there—there—is where your project went off the rails.”

Ten years later, we’ve learned a great deal. We have a growing body of knowledge of how to apply beauty to development; O’Reilly even has a book on the subject. But testing...testing is inherently ugly, right? Aside from developer-facing testing, like TDD, testing is no fun at best and rather-have-a-tooth-pulled-with-no-anesthetic at worst, right?

No, I don’t think so. In math we have this idea of *prima facie* evidence, that an argument can be true on its face and not require proof. For example, there is no proof that you can add one to both sides of an equation or double both sides and the equation remains true. We accept

No pun on Ruby intended. I am a Perl hacker.

* Him, and Dr. Kathleen Shannon, and Dr. Mohammad Mouzzam, and Professor Dean Defino, and Professor Maureen Malone.

† My coworker and occasional writing partner, Chris McMahon, has a good bit to say about testing as a performing art. You should check out...oh, wait, he has his own chapter. All right, then.

‡ <http://www.joelonsoftware.com/articles/fog0000000018.html>

this at face value—*prima facie*—because it’s obvious. All of our efforts in math build on top of these basic *prima facie* (or “axiomatic”) arguments.[§]

So here’s one for you: boring, brain-dead, gag-me-with-a-spoon testing is *bad* testing. It’s merely checking. And it is not beautiful. One thing we know about ugly solutions is that they are wrong; they’ve gone off the rails.

We can do better.

Come Walk with Me, The Best Is Yet to Be

This phrase is very common and I am unsure of its origins. I believe I first read it in the collected poetry of my grandmother, Lynette Isham. My favorite poem of hers included this line: “Once there was music in my heart. Then I met you...and I heard the words.”

This was about her son, my father, Roger Heusser. I don’t know about testing, but that’s some of the most beautiful prose I have ever read. I had to put it in print.

For our purposes, let’s look at software risk management as an investment of time and resources to find problems before those problems become angry customers. James Whittaker, the author of *How To Break Software* (Addison-Wesley), took that idea one step further to say that customers don’t want to pay a tester a salary; they want testing to be performed, and are willing to pay for it.^{||} This makes testing, at least in theory, a naturally outsourceable function.

No, I’m not suggesting that your team outsource testing.[#] At Socialtext, we develop and test software in parallel (more about that later) and do informal collaboration instead of handoffs. Outsourcing, as an alternative, usually involves “passing off” software to a test team and waiting for results, only to get a hundred bug reports dropped in your lap. Generally speaking, that is neither effective nor efficient—and certainly not beautiful.

I am suggesting that *management wants testing to have happened*, and wants the results of that testing to be presented in a way they understand. How we do that is up to us. Let’s start with that ugliest of false dichotomies: manual or automated testing.

§ In fact, most of geometry is built on top of the idea that parallel lines never intersect. The proof of this basic rule? You won’t find it. Anywhere. It’s *prima facie*. If you can figure it out, give me a call; we could probably win at least a million dollars. I am completely serious.

|| http://blogs.msdn.com/james_whittaker/archive/2008/08/20/the-future-of-software-testing-part-1.aspx

If your organization simply does not view development or testing as a core competence—perhaps if you are not a software company—it might a good idea for you to find a partner that does have that competence and wants to develop a symbiotic relationship with you. (See my M.S. thesis at http://www.xndev.com/CS/CS692/TheOutsourcingEquation_ABIT.doc.)

Automated Testing Isn't

It's very tempting to speak of "automated testing" as if it were "automated manufacturing"—where we have the robot doing the exact same thing as the thinking human. So we take an application like the one shown in [Figure 16-1](#) with a simple test script like this:

1. Enter 4 in the first box.
2. Enter 4 in the second box.
3. Select the Multiply option from the Operations drop-down.
4. Press Submit.
5. Expect "16" in the answer box.

Magical Mystery Calculator

First Number:

Second Number:

Choose an operation ▾

Choose an operation
Addition
Subtraction
Multiplication
Division
Exponent

Answer:

FIGURE 16-1. A very simple application

We get a computer to do all of those steps, and call it automation. The problem is that there is a hidden second expectation at the end of every test case documented this way: "And nothing else odd happened."

The simplest way to deal with this "nothing else odd" is to capture the entire screen and compare runs, but then any time a developer moves a button, or you change screen resolution, color scheme, or anything else, the software will throw a false error.

These days it's more common to check only for the exact assertion. Which means you miss things like the following:

- An icon's background color is not transparent.
- After the submit, the Operations drop-down changed back to the default of Plus, so it reads like "4 + 4 = 16".
- After the second value is entered, the cancel button becomes disabled.
- The Answer box is editable when it should be disabled (grayed out).

- The operation took eight seconds to complete.
- The new page that is generated has the right answer, but the first value entered is zeroed out. In other words, it now reads $0 + 4 = 8$.

A thinking, human tester would notice these things in an instant. A computer will not. So an “automated test” is simply not the same thing as having a human being running the test. Moreover, say the requirements allow 10 seconds for the result. A human can notice that the simple multiplication is just barely within tolerance, and might experiment with large numbers or complex operations in order to find other errors. A computer cannot do that.

So we find several different root causes of bugs, which could be found by different methods.

Into Socialtext

Before I could introduce Socialtext, I needed to introduce our work environment and lay out the fundamental issues involved when we test. At Socialtext, we have a number of different defects, with different root causes, that need to be found (or prevented) using different technical strategies. Without those precepts, we’d be making assumptions about the mission of our team (find bugs? or report status?), our goals (100% automation of all tests?), and what metrics are appropriate (number of tests? statement coverage?).

I did a simple check in the Socialtext Bug database and found the following rough categories of bugs ([Table 16-1](#)).

TABLE 16-1. Defect categories and examples

Category of defect		Examples
Untestable or rendering	9	Exported PDF has incorrect font; widget headers do not preserve padding; tag lookahead does not keep up with typing
Browser compatibility untestable or rendering	9	Workspace pulldown has no scrollbar on IE6
Catchable by computer-based browser execution and evaluation	19	Delete tag; tag does not go away
Catchable by reasonably light slideshow	7	All files list is incorrectly sorted for a complex sort
Special characters, exception-handling, internationalization	13	Special characters (@\$=, copyright or trademark, etc.) in fields do not render correctly on save
Appliance installs or upgrades	8	Appliance upgrade fails
Backup, restore, import, export	3	Import of old (<3 releases back) workspace fails
Usability issues	9	Tab order, spelling, too many clicks to use, “make this UI like other dialogs”

Category of defect		Examples
Any kind of test	8	Icons render as “X” on latest build; open or save fail on latest build; home page of app fails on latest build
Background/batch processes	4	Automatic emails are going off too often/not often enough/incorrect
Complex interactions of software	7	Two specific widgets, used together, can corrupt each other (out of dozens); cannot perform a specific operation twice in a row
Performance	4	Twitter-like signals take > 1 minute to display
Total	100	

We could argue the importance of the different categories of defects; for example, you could argue that test automation should include special cases, or complex interactions, or “any kind of test,” or that appliance upgrades should be automated and count as “automatable defects.” You could argue that some of the rendering could be caught by automation, and that we could have software that opens a socket, downloads a PDF, and then does some sort of compare against a PDF generated last week.

In any event, I submit that this defect breakdown shows that an application will fail in many different ways, and test strategy needs to address those different ways through a variety of approaches, or a balanced breakfast. Now I would like to tell you about how we find those bugs at Socialtext.

But...What Do You Make?

Oh, excuse me. At Socialtext we make software that allows people to have conversations instead of transactions. Think Facebook, or blogging, or Twitter—but inside your business and secure. Our initial product offering was a wiki, which enables a sort of corporate intranet that can be edited by anyone at any time, using a simple markup language or even an MS Word–like editor.

Figure 16-2 shows a simple wiki page: a user story for a new feature.

To change the page, the user clicks the Edit button, which brings up the page shown in Figure 16-3.

Notice that this editor is very similar to Microsoft Word, with its buttons for bold, italics, headers, add a link or image, and so on. The text also appears formatted on-screen. A literal-text inspection could find the right words on the page, but they could be missing formatting or have other font problems. Simply testing the editor itself is a nontrivial testing problem.

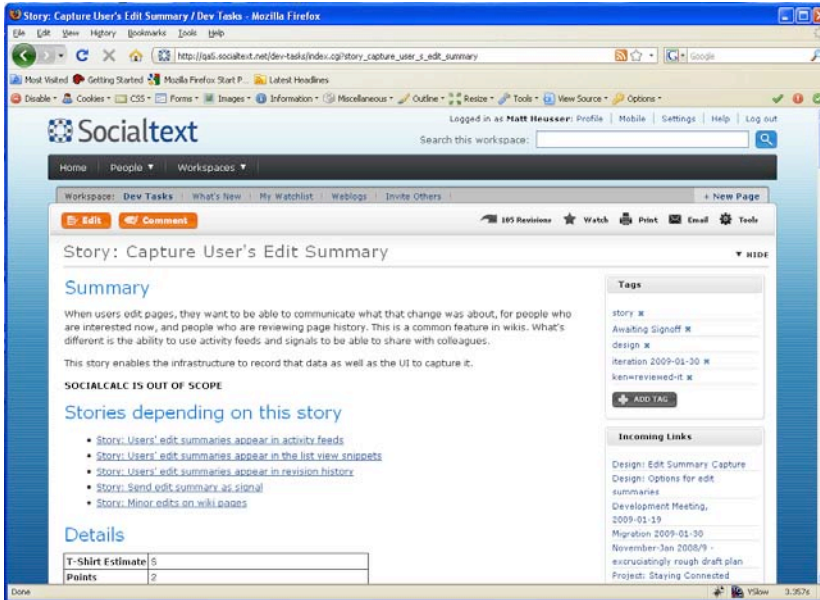


FIGURE 16-2. Viewing a wiki page

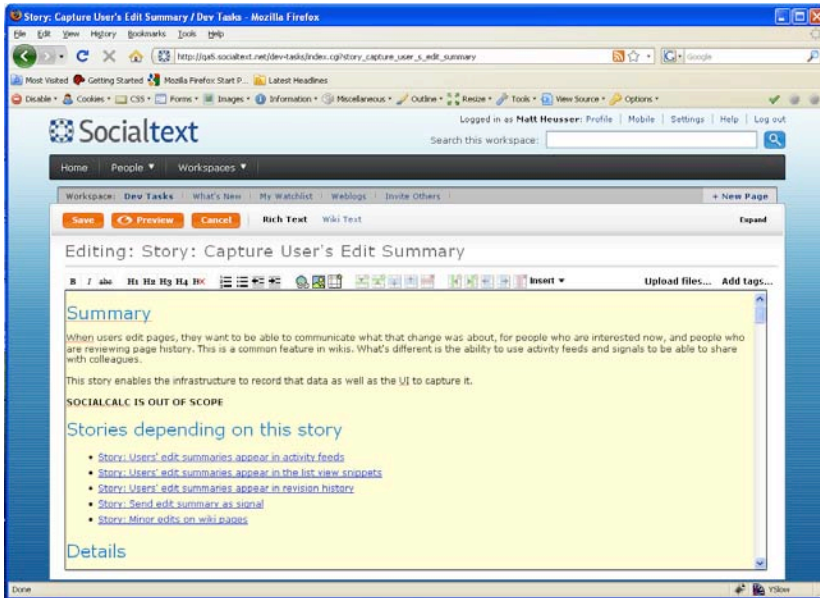


FIGURE 16-3. Editing a wiki page

Socialtext’s wiki software is built out of open source components, with a Linux (Ubuntu) operating system, Apache web server, Postgres database, and Perl programming language—essentially a LAMP stack.*

Software Process at Socialtext

Socialtext follows a development process inspired by Extreme Programming. The basic unit of work is a “story,” which is an extremely lightweight requirements document. A story contains a brief description of a feature along with examples of what needs to happen to consider the story completed; we call these examples “acceptance tests” and describe them in plain English. The purpose of the story is to create a shared mental model of what work will be done and to have a single point of reference and expectations. Of course, each story is embodied in a Socialtext wiki page; we tag stories with “Needs Review,” “In Dev,” “In QA,” or “Awaiting Signoff” to create an informal workflow and indicate progress of the work.

We do not expect our stories to be complete, as requirements change over time. Instead, we try to make our stories *good enough* for the developers to begin work, and declare them “good enough” when the story review process reaches the point of diminishing returns.

We do not expect our stories to be correct—but we do have a review process to make them *better*.

We do not expect our stories to be unambiguous, as English is a vague and ambiguous language itself—but having concrete examples certainly helps.

A story describes a complete feature, but that feature may not itself be marketable or saleable. So we collect a series of stories in an iteration, and timebox those iterations at two weeks in length. We are not dogmatic about iteration length, and allow a three-week iteration around the Christmas holiday. Figure 16-4 illustrates some of the activities that happen during a typical iteration and the order in which they happen.

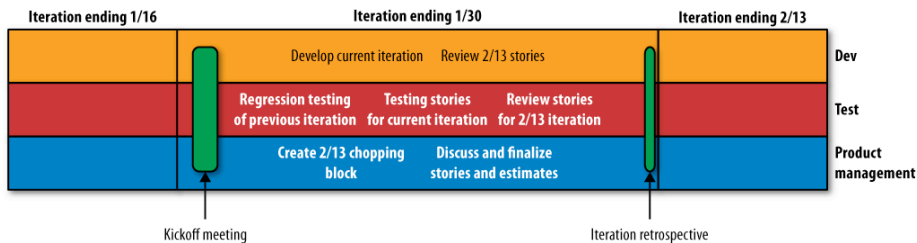


FIGURE 16-4. Iteration progression overview

* Linux, Apache, MySQL, Perl: LAMP.






Ideally, developers finish stories and get to “code close” on the Wednesday of the second week of the iteration that begins on a Monday. Regression testing takes about two days, and on Monday, everyone starts on the next iteration. This assumes that the testers find no bugs in regression testing and that the developers are always on time. And, I am sure, that theory might sell a lot of books. Our experience has been that it’s common for regression testing of the previous iteration to run into the next. That’s OK because the devs haven’t handed off any stories yet. (When that happens, one of the biggest problems we face is straddling the iterations, which is when developers and testers are fixing and finding bugs in the previous iteration while attempting to kick off and develop the current one.)

When you consider that, at this point, product management is working on stories for the next iteration, you realize that the team is working on three iterations at once. You may ask how this is beautiful (more about that later), but one explanation is that it is a pipeline, not a waterfall.

What We Actually Do

Let’s follow a story—specifically, Edit Summaries. First, a product manager gets an idea. He notices that customers are using our “Recent Changes” feature heavily, but want to know if the change is important (“Matt modified the vacation policy”) or minor (“Matt made spelling and grammar changes to the contracting policy”). Edit Summaries would allow the user to create a summary that shows up when people are looking at revision history (what’s new) or a user’s stream of actions. It’s a killer feature, and product management says we have to have it in order to sell to Acme Corporation, so they draft a story. The story looks something like [Figure 16-5](#).

Story Overview

T-Shirt Estimate	S
Points	2
Estimator(s)	 Shawn Devlin,  Jeremy Stashewsky
Dependencies	Tech Story: NLW logs events to database
Lead Dev	 Jeremy Stashewsky
Lead QA	 Matt Heusser
Priority	2
Customer	 Adina Levin
Comments	(1 for UI changes and 1 for backend changes)
Notes	

The user-interface will include a check box and text box for the user to enter an optional, brief edit summary. The summary is intended to convey the changes made (eg “fixed spelling” or “increased vacation policy by one week.” That summary is recorded on the save event, and will be displayed along with the event where events appear, such as the user’s profile or activities widget. Summaries will also appear as an new column when user’s click revisions (page history.) Future work includes having the summaries appear any time a list of pages is show, such as search results, pages tag list, what’s new, etc. This is described in [Story: Users’ edit summaries appear in the list view snippets](#)

FIGURE 16-5. Details on a story

The story starts out with a customer, who is the business sponsor (probably a product manager), and a description of a feature. At this point the story is only a paragraph of text; no one has been assigned. The product manager, developers, architect/coach, and occasionally QA meet to discuss the overall size of the story, implementation details, and story tests. We record our estimates in terms of points, which are ideal engineering half-days for a developer to implement. Due to pairing, vacation, interruptions, and so on, the ideal is never realized; we track the actual number of points accomplished and use that to predict future performance, a notion known as “iteration velocity.”

During this process, the product management team might create mockups. For example, the Edit Summary dialog should like [Figure 16-6](#).

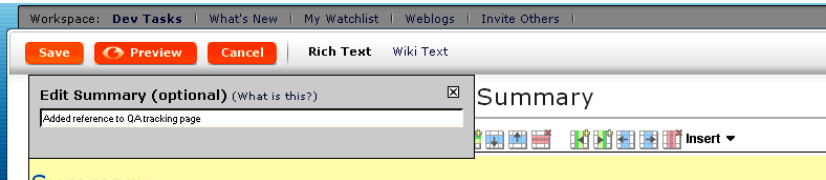


FIGURE 16-6. An Edit Summary mockup

One more thing about the actual requirements for Edit Summaries before we move on. You’ll notice that once the Edit Summaries go in, it is not clear how the user will get them out. There are other stories that specify that Edit Summaries can appear in revision history, in Recent Changes, in a user’s activity stream as updates, and as micro-blog posts. This story is to *capture* the Edit Summary, and is concerned only with the single dialog box.

The story includes the header, a plain-text description, some sample screen captures, and the acceptance tests.

During the initial cut of the story, the product owner makes a good-faith first attempt to create acceptance tests, which are augmented by developers and testers before any developer writes a line of code. The goal of the story-creation and review process is not to create an exhaustive list; this is not big-testing-up-front (BTUF). Instead, the team is striving to really understand the problem domain, so that when we find a problem later on, the entire team will shrug in agreement that “yup, that’s a bug” and be committed to fixing it as a whole team—instead of whining that someone changed their minds or that the requirements weren’t right.

[Table 16-2](#) lists some of the acceptance tests for the Edit Summaries dialog. “lando” is Orlando Vazquez, and MRH is Matthew R. Heusser. The tests are stored on a wiki page, and at the end of a working time-block, the worker updates the document with what is completed—but we’re getting ahead of ourselves. Notice that these are specifications at the behavioral level of the feature.

TABLE 16-2. Acceptance tests for the Edit Summaries dialog

Test	Dev sign off	QA sign off
User mouses over Save button in either edit mode and a summary UI appears.	lando	MRH
User quickly passes over Save button; summary UI does not appear.	lando	MRH
Move focus (click), either into edit area or entirely away; summary UI disappears.	lando	MRH
Move focus (click) entirely away, mouse over Save again, summary UI appears; click in edit area, summary UI disappears.	lando	MRH
Mouse over Save, UI appears. Move mouse directly to the right, click Cancel. Click Edit. UI should not appear.	lando	MRH
Summary UI maintains content across disappearing and reappearing during editing.	lando	MRH
User enters a summary and clicks Cancel. User clicks edit, and goes to Edit Summary, old summary is gone.	lando	MRH
Superfluous whitespace should be minimized from the Edit Summary; “ dog ” becomes “dog”, etc.	lando	MRH
Entering “:” or “;” in the summary should save and retrieve correctly.	lando	MRH
Summaries persist on st-admin export/reimport workspace.	lando	MRH
Commenting on a page shouldn’t change the current Edit Summary.	lando	MRH
User cannot enter > 250 characters in the UI.	lando	MRH
Backend truncates > 250 characters.	lando	MRH
Documentation is updated to include new screen captures.	lando	

This table lists only 15 tests; I’m sure you can think of more. The actual software had 35, including requirements for localization, for an API so programmers can write an Edit Summary without the web browser, and for creating some amount of test automation, which acts as a requirement and is fed into the story estimate.

You’ll note that I do not view this as a list of *everything* to test. It is not a list of test cases; it is a list of acceptance tests. These tests have value for communication, to create a shared model of the work to be done, and as examples. As a list of test cases, it is actually pretty weak. It does give the developers a good idea of the expected behavior for Edit Summaries.

At this point the developers had everything they needed to write the code. Orlando called Stash (Jeremy Stashewsky) on the phone and, sharing a shared-editor screen, created the story as a pair. They coded the backend in Perl and, along the way, wrote unit tests in TDD-style to aid in design and testing. A lengthy discussion of TDD is outside the scope of this chapter, but I have asked Lando to say a few words about what he did:

My process while TDD'ing a new story is to first plan some simple unit tests that act almost like a spec of how I would like to see the story work. I write these simple failing tests, and then write the code to make them pass. Then, having implemented the “normal” use-case scenarios, I go on to flesh out the rest of the story tests and edge cases in the same way.

```
# initialize some globals we'll use
setup_tests();

signal_edit_summary: {
  my $hub = setup_page(
    revision_id => $save_revision_id,
    edit_summary => 'Dancing is forbidden!',
    signal_edit_summary => 1,
  );

  $hub->edit->edit_content;
  my $page = load_page_by_id(hub => $hub, page_id => $page_id);
  is $page->edit_summary, 'Dancing is forbidden!', 'proxy method
works';

  # check that the signal was created
  signal_ok (
    viewer => $user,
    signaler => $user,
    body => "Dancing is forbidden!" (edited Save Page in Admin
Wiki)',
    topic => {
      page_id => $page_id,
      workspace_id => $current_workspace->workspace_id,
    },
    msg => 'normal length edit summary with signal'
  );

  # check that the appropriate events were created
  is_event_count(2);
  event_ok (
    event_class => 'signal',
    action => 'page_edit',
  );
  event_ok (
    event_class => 'page',
    action => 'edit_save',
  );
}
```

After Lando finishes his work, he deletes the page tag “In Dev” and adds a tag called “In QA.” The next time a developer looks at the page that tracks this iteration’s work, the story has magically moved into the QA queue, and the QA lead (that’s me) will pick it up.

Now I check out the latest version of the code and rip through the story tests in all supported browsers: the newest and previous versions of Internet Explorer, Firefox, and Safari. I also try odd combinations of characters, special characters, and, of course, straight text with no spaces.

After story tests, I conduct exploratory testing of the story and related changes.[†] I find that in Internet Explorer, the box appears in the wrong position (in the far right), return the story to “In Dev,” and email the developer. Lando fixes the issue, marks it in QA, I retest, and I move the story to customer acceptance.

Occasionally, the problem is big enough to mark, hard enough to fix, and not critical to the success of the story. In that case, the tester will create a Bugzilla (bug tracking) ticket and add it to the story as a note. The product manager can read the bug and determine whether we can ship with the defect or not. The follow-up story, for example, was that summaries are added to revision history. I found that I could add 250 characters of text with no spaces and cause serious scrollbar and margin issues in the revision history page. After a quick huddle, we decided that a) you don’t see this issue using the largest words in the English dictionary, b) it should probably be fixed, but perhaps not today, and c) I would create a bug report.

That tests the story once, but we release out to our Software-as-a-Service production server every two weeks, and a minor change to something else can easily ripple. If we retested every acceptance test on every browser every two weeks, the burden would eventually cripple the team. So we do some test automation using a framework we developed called *wikitests*.

Wikitests[‡]

A wikipitest is a keyword-driven test that drives the browser. With wikipitests, each test is expressed as a series of commands in a table, one command per table row. These commands are Selenese, which is near English. They can be read by anyone with a modest programming background, and because they are stored in wiki pages, anyone can view, revise, and save them. In addition, we can build our own commands—for example, `st-login`—which combine common operations. For example, the following tables show the test for Edit Summaries. You’ll notice that the first column is the command, and the second and third are parameters. For example, in the first command, `Pt` is the name of the pause variable, and `5000` is the amount

[†] On rereading this chapter, I’m a little disappointed in myself that the second half covers very little description of the time and effort we spend on exploratory testing, but I wanted to cover the areas that were most unique, special, and beautiful about Socialtext. Given the list of authors in this book, I expect plenty of coverage of exploratory methods in other chapters.

[‡] A few years ago, Bret Pettichord was running a class on homebrew test automation where he suggested a style of test automation framework. I would argue that wikipitests are among the most advanced known frameworks developed in that style. The framework itself strings together open source and home-development components to create an overall effect. The browser-driver component of the framework is Selenium RC, mostly written by Jason Huggins. `test:www:selenium`, developed by my coworker, Luke Closs, wraps Selenium RC so that it can be driven through the Perl programming language. Other components take Socialtext wiki pages and convert them into Perl commands, which are executed by `test:www:selenium`. Ken Pier manages the ongoing project to create, extend, and maintain wikipitests. I have made some modest contributions, developing plug-ins, wikipitests commands, and lots and lots of test cases, but the actual credit goes to Ken and Luke. Other contributors include Kevin Jones, Lyssa Kaelher, Shahed Hossain, the aforementioned Chris McMahan, our intern David Ahnnger-Pier, and, recently, Audrey Tang.

to substitute. In the second, `st-admin` is the backend command to take an action, `update-page` is the parameter to create a new page, and the third column is the expected result to compare. (If the output said “Unable to create a new page” or something like that, the test would log an error.) The number after `wait_for_element_present_ok` is the number of thousandths of a second to wait, `%%variable_name%%` means substitute a variable for the data, and so on. An abbreviated version of the Edit Summaries test follows.

Comment	Test Case: Edit Summaries		
----------------	---------------------------	--	--

Comment	Test Case: Edit Summaries—create a page from file		
<code>st-admin</code>	<code>updatepage workspace %%workspace%% email %%email%% page "Edit Summaries %%start_time%%" < %%wikitest_client_files%%wikitest_toc.txt</code>		The “Edit Summaries % %%start_time% %” page has been created

Comment	Test Case: Edit Summaries—create one Edit Summary		
<code>open_ok</code>	<code>/%%workspace%%/index.cgi?Edit Summaries %%start_time%%</code>		
<code>wait_for_element_visible_ok</code>	<code>steditbuttonlink</code>		30000
<code>click_ok</code>	<code>steditbuttonlink</code>		
<code>wait_for_element_visible_ok</code>	<code>link=Wiki Text</code>		30000
<code>click_ok</code>	<code>link=Wiki Text</code>		
<code>wait_for_element_visible_ok</code>	<code>wikiwg_wikitext_textarea</code>		30000
<code>wait_for_element_visible_ok</code>	<code>stsavebuttonlink</code>		30000

Comment	Test Case: Edit Summaries—type the summary		
<code>wait_for_element_present_ok</code>	<code>steditsummarytextarea</code>		30000
<code>click_ok</code>	<code>steditsummarytextarea</code>		
<code>type_ok</code>	<code>steditsummarytextarea</code>		Quick Summary for my friends % %%start_time% %
<code>click_and_wait</code>	<code>stsavebuttonlink</code>		

Comment	Test Case: Edit Summaries—create a second Edit Summary	
open_ok	/%workspace%/index.cgi?Edit Summaries %%start_time%%	
wait_for_element_visible_ok	steditbuttonlink	30000
click_ok	steditbuttonlink	
wait_for_element_visible_ok	stsavebuttonlink	30000
wait_for_element_visible_ok	link=Wiki Text	30000
click_ok	link=Wiki Text	
wait_for_element_visible_ok	wikiwg_wikitext_textarea	30000

Comment	Test Case: Edit Summaries—type the second summary	
wait_for_element_present_ok	steditsummarytextarea	30000
click_ok	steditsummarytextarea	
type_ok	steditsummarytextarea	A second summary for a wikitest % %%start_time%% %
click_and_wait	stsavebuttonlink	
wait_for_element_visible_ok	steditbuttonlink	30000

Comment	Test Case: Edit Summaries—revision history	
click_and_wait	link=3 Revisions	
wait_for_element_visible_ok	link=Back To Current Revision	30000
text_like	qr/A second summary for a wikitest %%start_time%%.+Quick Summary for my friends %%start_time%%/	

Comment	Test Case: Edit Summaries—teardown	
st-admin	purgepage workspace %workspace% page edit_summaries_%%start_time%%	page was purged

Comment	Test case: Edit Summaries COMPLETED		
----------------	-------------------------------------	--	--

A few points about this testing style: all HTML elements have to have an ID name, such as `st-save-button-link`, and the software runs around, clicking links and looking at values. To find the name of elements, I had to use a tool[§] and “inspect” the user interface—or else collaborate with the developers on names before the code was produced.

Thus the overall process is to create a test case (which is a wiki page), run the test, watch it fail, change something, and try again. In this example, we see several dead stops and retries, as I abandon one test strategy and move to a different one, or perhaps peer into the code to debug it. All in all, it takes me half a day to write automation for a feature that can be tested manually, in all browsers, in an hour. I did not pick this as some sort of extreme example; this problem is typical. Moreover, the wikipage is not sapient; it cannot tell if the Edit Summary dialogue would appear to be too wide, move down suddenly, or be the wrong color. What it does do is allow some amount of basic sanity coverage in a short period of time, which we desire in order to enable many quick iterations.

Wikitest output looks something like this:

```
# st-login: wikipester@ken.socialtext.net, d3vnu11l, test-data -
/nlw/login.html?redirect_to=%2Ftest-data%2Findex.cgi
ok 1 - open, /nlw/login.html?redirect_to=%2Ftest-data%2Findex.cgi
ok 2 - type, username, wikipester@ken.socialtext.net
ok 3 - type, password, d3vnu11l
ok 4 - click, id=login_btn, log in
ok 5 - wait_for_page_to_load, 60000
#
# comment: Test Case: Edit Summaries
# Set 'pt' to '5000'
#
# comment: Test Case: Edit Summaries - create a page from file, because we can't type
# newlines with type_ok st-admin update-page --workspace test-data --email
# wikipester@ken.socialtext.net --page "Edit Summaries 1234802223"
# < /opt/wikipester_files/wikipester toc.txt
ok 6 - st-admin update-page --workspace test-data --email wikipester@ken.socialtext.net
--page "Edit Summaries 1234802223" </opt/wikipester_files/wikipester toc.txt
ok 7 - open, /test-data/index.cgi?Edit Summaries 1234802223
ok 8 - click, st-edit-button-link
ok 9 - wait_for_condition, try { selenium.isTextPresent('Editing: Edit Summaries
1234802223') ? true : false } catch(e) { false }, 55000
ok 10 - wait_for_condition, try { selenium.isVisible('st-save-button-link') ? true :
false } catch(e) { false }, 30000
ok 11 - wait_for_condition, try { selenium.isElementPresent('st-edit-summary-text-area')
? true : false } catch(e) { false }, 5000
ok 12 - click, st-edit-summary-text-area
ok 13 - type, st-edit-summary-text-area, Quick Summary for my friends
ok 14 - click, st-save-button-link
ok 15 - wait_for_condition, try { selenium.isElementPresent('st-edit-button-link') ?
true : false } catch(e) { false }, 30000
ok 16 - click, st-edit-button-link
```

§ Firebug and the Web Developer plug-in are free for Mozilla Firefox. The IE developer toolbar is a good alternative for Internet Explorer.

```

ok 17 - wait_for_condition, try { selenium.isVisible('st-save-button-link') ? true :
      false } catch(e) { false }, 55000
ok 18 - wait_for_condition, try { selenium.isElementPresent('st-edit-summary-text-area')
      ? true : false } catch(e) { false }, 30000
ok 19 - click, st-edit-summary-text-area
ok 20 - type, st-edit-summary-text-area, A second summary for a wikipage
ok 21 - click, st-save-button-link
ok 22 - wait_for_condition, try { selenium.isVisible('st-edit-button-link') ? true :
      false } catch(e) { false }, 60000
ok 23 - click, //a[@id='st-watchlist-indicator'], clicking watch button
ok 24 - wait_for_condition, try { selenium.isVisible('link=3 Revisions') ? true :
      false } catch (e) { false }, 60000
#
# comment: Test Case: Edit Summaries Revision History
ok 25 - click, link=3 Revisions
# comment: Test Case: Edit Summaries teardown
# st-admin purge-page --workspace test-data --page edit_summaries_1234802223
ok 26 - st-admin purge-page --workspace test-data --page edit_summaries_1234802223
#
# comment: Test case: Edit Summaries COMPLETED
1..26

```

If failures occurred, the software would say something like, “It looks like you failed X of Y tests” or “Failed after X tests run.” We use a tool, called `tap2html`, that summarizes the results of tests suites so that we can see success and failure of a 10,000 test-step suite at a glance and drill down into details when needed.

Although we do have some canned test data (for search, watch pages, and so on), each test case is designed to run independently. Wikitests can also be grouped into test sets, so it is possible to make a wikipage that is a series of links to other wikipages. To do regression testing with wikipages, we run a test suite, redirect the output to a file, and search through the file for “not ok”, “error”, “warning”, and other messages. (Yes, we have some scripts that can process the errors and tell us what test cases failed, and a visualizer that takes ASCII text output and converts it into a graphical representation viewed in a browser.)

A Balanced Breakfast Approach

So far, we have acceptance tests, unit tests, and wikipages. We are painfully aware of the bugs that wikipages fail to find, and yet every two weeks we need to do regression testing of all of our features, with a goal of moving code to internal staging within 48 hours of starting regression tests. To do that, we create a candidate test-tracking wiki page. Using the candidate page, we can assign testers to work on different pieces of the software, and report what bug reports have been filed. When the page says “ok”, “ok”, “ok” for all elements, testing for the iteration is done. (“bz” followed by a number means the tester found a bug.) Let’s look at a candidate page, already in progress, and discuss it:

Testing Iteration Ending 2009-01-30:

- test-release status on iteration page: Green

wikitests:

- FF2: **PASS. Widgets tests have been fixed in master but not in 01-30**
- FF3 **in progress** chris. (Thought this was mine. **PASS.**) Ken (It was, I figured I'd take it off your hands. It's snowing here. -C)
 - TC: Hidden Email Address for Public wiki must have a race condition.
- IE7: **PASS mostly** mcchris at step 5941 in TC: Calc Watchlist the database is corrupted and apache-perl crashed. Can not reproduce. Otherwise no errors at all. **update:** Stash suggests that the nlw-error.log record indicates a race condition when saving spreadsheets such that an expected db record does not exist upon a subsequent INSERT.
- IE6: mcchris **PASS**
 - TC: REST Workspace passes on re-run
 - original run encountered a single 502 error. this makes me nervous
 - TC: Hidden Email Address for Public wiki passes run manually. fails b/c of a race condition w/ search results maybe?
 - **All widgets tests failed** Cursorsory manual examination widgets seem ok but slow as usual.
 - **All Reports tests failed** The reports tables were not in place and we're sending raw sql to the browser when the env is fubar. Following up with Stash/someone. Note: I think I have seen this failure to put reports tables in place before. Will try to figure out why that happens
- Safari: matt the cheshire cast button in test_case_revisions doesn't work, and causes a failure. mcchris says this is a known issue. test_case_preview seems to switch from simple to rich text mode, which isn't possible in safair. Widget tests fail but will be tested manually.

Run Test Case: Gadgets Galore to get one of every Gallery gadget on your Dashboard

Visual Inspection:

link checklist here if desired. [template here](#)

- FF2 Pass - Added Slideshow tests for Edit Summaries and changed the Miki Tagging order
- FF3 **PASS (provisional)** Chris
 - multipage export to PDF and to Word failed on my laptop but I believe those are local problems and would like others to confirm.
 - *worksforme* on ff3/linux. --rs
- IE7 **PASS** -chris
- IE6 scotty - **PASS** all fixed

- Safari matt - The automated tests showed I saw some strangeness for Export and upload when editing. So I ran it manually without a problem - also upload while editing isn't supported on safari, so I'm not worried.

Socialcalc - matt - I noticed that you can click first background color, then text color, to create an awkward effect, but It's just an awkwardness; I did not file a bug. Also {bz: 2046}

- IE6 - ok
- IE7 - ok
- Safari - ok
- FF2 - ok
- FF3 - ok

People - matt

- IE6 - ok but {bz: 2049}
- IE7 - ok but {bz: 2049}
- Safari - ok but {bz: 2049}
- FF2 - ok but {bz: 2049}
- FF3 - ok but {bz: 2049}

Test My Conversations widgets by creating entries in all three tabs and clicking on all links in entries. Items should open in new pages.

Signals - scotty

- IE6 - ok
- IE7 - ok
- Safari - ok

And so on.

TestRunner is a suite of developer-facing tests. We list it for completeness; if TestRunner is red, something is wrong. Wikitests are the suite of all browser-driving, unattended tests.

“Slideshow” blends wikitests and visual inspection, and was invented by Socialtext’s product quality manager, Ken Pier. A typical Slideshow run takes about half an hour, and can catch visual bugs that a wikipitest would miss. Due to limitations of our test tools, other tests are run manually, such as HTTPS-based access. Because Selenium cannot access HTTPS pages by driving the browser, we sometimes run HTTPS tests manually to make sure the Secure Sockets Layer works correctly. I did not list feature retests, which we do to check the features of the new stories in the new “iteration” branch, to make sure any new changes do not “step on” the code introduced earlier in the cycle, bug reverification in the new branch, or the heavy exploratory methods we do. Most of our exploratory work is tied to a story, but we also conduct

exploratory tests on the candidate, and sometimes those charters are tracked on the candidate page.

During candidate testing we also test the upgrade of our software from one version to the next. Because we offer both a hosted version and an appliance, and because we allow customers to upgrade at any time, we have a set of outstanding versions in the field. For some time we had a matrix and tested every combination of the software upgrade to current—and yes, that could very nearly be an entire chapter. Suffice it to say, it needs to be done and done well, and we have tools and eventually developed architectural enhancements to make upgrade testing, if not fun, at least less painful.

SocialCalc, People, and Dashboard are new products. Because the code is so new and the graphics for those products are so complex, the user interface is constantly evolving, and it would be premature for us to invest heavily in wikttests.^{||} So we have some wikttests for them that work in some browsers, and we also have a documented test plan. Testing these products means running the wikttests, then also following up by hand with exploration, and finally consulting the test plan to see what you've missed, and perhaps going back to cover those areas. Figure 16-7 shows an excerpt of the Dashboard test plan.

Test	FF	IE6	IE7	Safari	Notes
Dragging tests - normal widget - displace on drop	OK, only at Top of the column, not in any other place.	OK	OK	OK	In IE6, IE7: Not getting displaced on drop. Always remain in same position. In Safari: Though, displace works on drop, but it's not as pleasant as it should be. Widgets take long time to finish the dragging options because it repopulates that existing data in the widgets. I believe it should have simple game to nav place without requesting asking the content from server.
Drag - Specifically, drag a widget that is not in the top row or left column to above the top/left	OK	OK	OK	OK	
Dragging Tests - normal widget - over, displace nothing	OK	OK	OK	OK	
Dragging Tests - oversize widget, place below another widget	OK	OK	OK	Maybe: Needs to be tested	It goes both in above and below of another widgets.
Dragging Tests - Oversize widget, displace on drop	OK	OK	OK	OK	
Click on links in any widget pops up new tab or window	OK	OK	OK	OK	Both in IE6 and IE7 received script error. For instance: Error 94, unspecified error. Also it opens up in same window.
Add a widget through the "Add Content" button	OK	OK	OK	OK	
Remove a widget by clicking the X	OK	OK	OK	OK	
Move a widget on the screen - by a few pixels - and drop. Expect a wiggle but no change.	OK	OK	OK	Maybe: Needs to be tested	
Add a widget with some customize	OK	OK	OK	OK	

FIGURE 16-7. Socialtext Dashboard test cases

^{||} Matthew wrote that in March of 2009. By June, we had much more wikttests coverage.

This is an older version; the current version is all white. I kept the red and yellow around to show how a manager can get status at a glance. At the same time, keeping these tests current can take a large amount of time. We strive to have the bare minimum of documented test cases.

Regression and Process Improvement

The stress on a software testing team comes from many directions. If we tried to rerun every test idea every two weeks, the weight would compound. Within a half-dozen iterations the team would be perennially behind. If we tried to automate every test idea, we'd have the same problem—and as I discussed earlier, trying to do automation can lead to blinders about entire categories of defects.

So as odd as it seems, the critical questions in this area are: What do we *not* test? What tests can we *skip*? What tests always seem to pass, we expect to always pass, and provide us only confirmatory value?

These are philosophical questions about truth, the classic problem of induction. As the saying goes, if we are trying to determine the truth of the statement “all swans are white,” a million white swans give us less information than a single black one.[#]

Over the past year, we have changed our tactics several times. It's impossible to know anything for sure, but if we spent eight hours per iteration testing features that always worked—and involved a well-isolated piece of the code—well, we had plenty of other techniques to use that could deliver results. We added reviews earlier in the process, changed the iteration schedule by a day or two here or there, and changed the format of our team standup meetings.

Perhaps it's time I told you about our staging tests.

The Last Pieces of the Puzzle

Once the candidate tests pass, the code leaves the test group and is placed in an internal staging environment. We use our own software—the wiki on staging—to essentially run our business.* We use it to create project plans, to track the iteration, to do QA tracking, and to create stories, for blogging and communication. That way, staging becomes the final proving ground for our software before it goes into production, and our employees occasionally do find defects or, more likely, usability issues or performance issues that unit and functional testing did not uncover.

[#] The best book about testing I've read this year is, by far, *The Black Swan* by Nassim Taleb (Random House), and this illustration comes entirely from him, with credit due. Of course, he took it from David Hume, who took it from some ancient Greeks....

* We do keep some element of sales and accounts payable, etc., on production, which is ever so slightly more stable, and has passed that “final test bed” of staging. Wouldn't you?

Our entire testing puzzle includes feature testing, wikttests, running test plans, exploratory testing, getting feedback from staging, usability, story review, beta programs, logging and log evaluation, and occasional performance research. If we tried to do everything and do it well, our team would swell in size and make the company unviable—or it would simply burn out the staff very quickly. Instead, we have to ask how much is enough and what can we trade out.

Our goal is not to know exactly what the software can do, but instead to provide a rapid assessment of the software that is fit for purpose. A few key questions I haven't mentioned yet are: "What should I be doing right now?" and "Am I done yet?"

Again, those are philosophical questions—questions of aesthetics. I would posit that combinations of the puzzle pieces that look the most beautiful are probably the correct ones.[†] The term for assembling these pieces is something I reluctantly call "test architecture."[‡] It turns out that Fred Brooks, the author of the all-time software classic *The Mythical Man-Month* (Addison-Wesley), appears to agree with me. At OOPSLA (the object-oriented software conference) in 2007, he defined the architect role in two halves:

1. First, to serve as an advocate for the customer.
2. In order to do that, to have an understanding of the entire product, end-to-end.

In the day and age of self-organizing, self-governing teams, I find a command-and-control big-designer-style architect to be archaic, but that work of advocating for the customer and understanding the product still needs to be done. Working at a higher level than developers, and with an express goal of finding and fixing flaws in the software, I honestly believe that the testing group has the capability of serving—in the best possible way—as architects of the product. In other words, we *are* testing software, *and* helping to put together the puzzle pieces to create a product that will delight and amaze our customer.

Delighting and amazing the customer through applying our minds and building our skills.
Hmm....

Somewhere, in the back of my mind, I see Dr. Homer Austin, my old math professor. And he is smiling.

[†] This is Occam's Razor restated.

[‡] Very reluctantly, as the analogy tends to separate the doers from the thinkers, which I believe is dangerous.

Acknowledgments

Of course, I am in debt to my professors at Salisbury University and also Grand Valley. Dr. Roger Ferguson and Dr. Paul Jorgensen both encouraged my interest in testing, and Dr. Paul Leidig encouraged me to write. Chris McMahon brought me into Socialtext, and Ken Pier, Luke Closs, and the rest of the Socialtext team built the testing process before I was even hired; it's been a privilege to work with them. (Ken also provided considerable, insightful peer review of this chapter.) Steve Poling, my friend and fellow technologist, has been encouraging me to write something like this for years.

My wife and children have been wonderful, encouraging, supportive, and understanding. I'd like to thank them most of all.

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.

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.