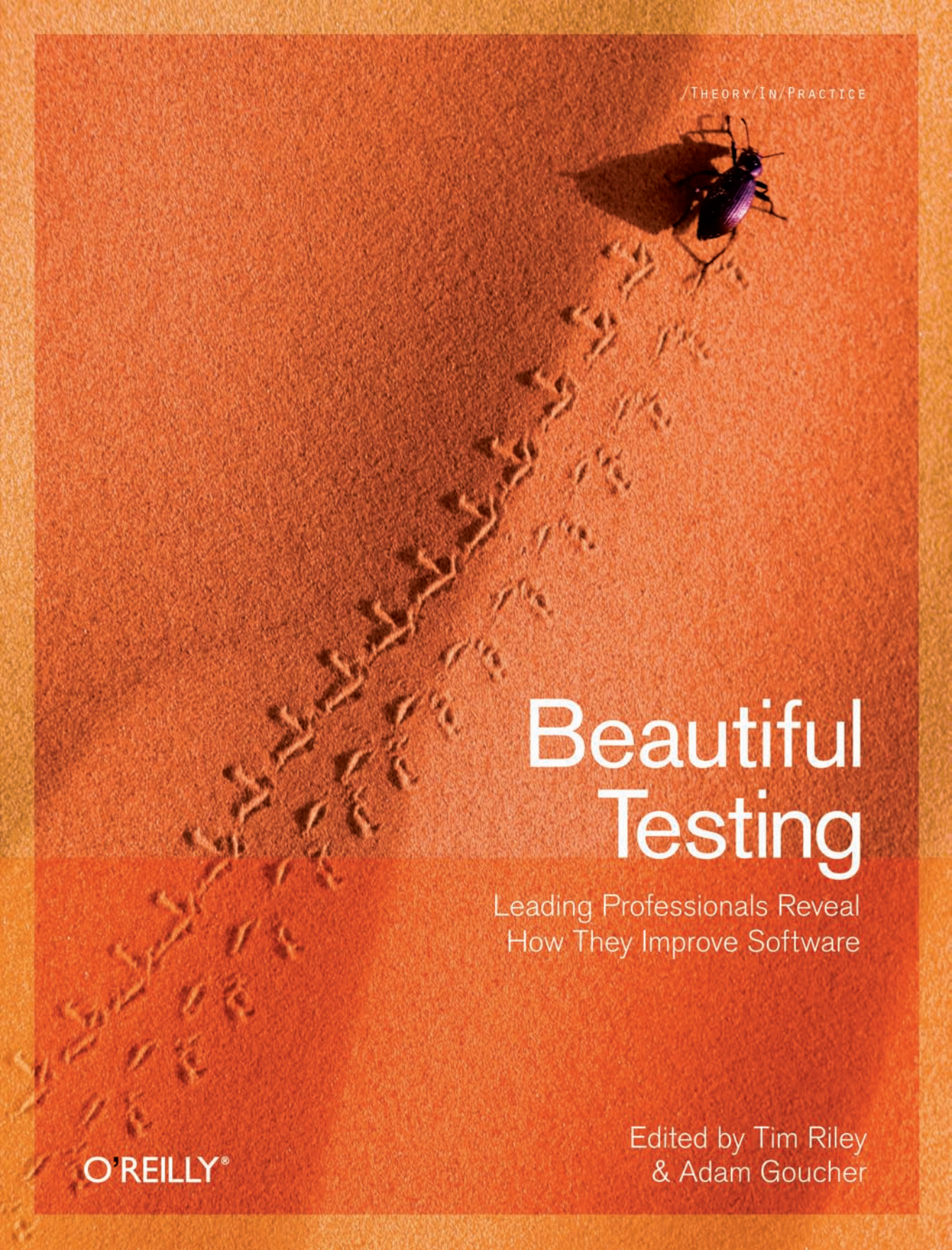
/THEORY/IN/PRACTICE



Beautiful Testing

Leading Professionals Reveal How They Improve Software

Edited by Tim Riley
& Adam Goucher

O'REILLY®

Software Engineering

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

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

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.

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

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

CHAPTER TWELVE

Software in Use

Karen N. Johnson

I stood in the ICU at my mom's bedside, thinking, "Don't cry, not now." I told myself I'd cry later on when I was alone. I didn't want to disrupt what was already chaos around me by becoming hysterical. I wanted to think clearly about what was going on. And I knew once I started crying, I would not be able to stop.

So I kept swallowing deeply, trying not to cry. My mom was lying in the hospital bed, hooked up to multiple devices. Her eyes were closed tight; she seemed to be in a deep sleep. I had the feeling that she was many miles away from me, but I was standing only inches from her side.

My mom had fallen and within 48 hours a brain injury had erupted deep inside her head that required emergency neurosurgery. Now post-surgery, she was in what the nurses and doctors refer to as a nonresponsive state. Her condition was unclear and her status considered unstable.

The neurosurgical ICU at Brigham and Women's Hospital in Boston is shaped in an arc, with each patient in a small alcove of her own. Each patient has her own nurse who stays within a few feet of the assigned patient at all times. That day an assortment of doctors, specialists, and family members streamed through the arched pathway, disappearing into the separate alcoves. A sense of urgency hung in the air. It occurred to me that each alcove had not only a patient but also a family and an event that had taken place. Everyone in the ICU has a story, a drama unfolding, and a family in panic and wait mode.

I was about as alone with my mom as one can be in a busy ICU. I looked up and around at all the equipment. There were tubes, wires, and unknown devices. All the equipment looked pretty scary. I started reading the labels on the devices for distraction. Labels, I thought, will

give me something to focus on. Reading labels will keep me from crying. The labels had company names printed on them, names such as Abbott Labs, Hospira, and Baxter. I knew each of these companies. I knew people who had worked at each of these companies. Gee, *I'd* worked at these companies. Memories of past projects and people began to trickle through my mind, a welcome distraction.

One tube in particular caught my eye. It was a feeding tube. It distressed me to see my mom's name on that tube. She'd always said that if she ever needed to be on a feeding tube, we should just…. I didn't want to think about that. I moved closer to the tube, my eyes following it from the fluid in the sterile bag hooked on an IV pole winding its way down to my mom. "Total parental nutrition" was the medical description of the fluid being pumped. I'd worked on that type of medical device and the computer software that directs it in building the right composition of fluids for the patient. If the dosage of the fluids or the mix of fluids is not correct, the patient can die.

I realized this was software I'd tested.

A Connection to My Work

Three weeks later, with my mom's situation stabilized as much as possible, it was time for me to go home to Chicago. My mom's condition had improved. She'd moved from the ICU to a regular hospital room, and then on to a rehabilitation center. Her progress had gone well, but she had a long way to go for a full recovery. I knew upon my return to Chicago I would once again have to seek work. Sometimes work finds me, and sometimes I find it. It's been that way since I left my full-time employment almost three years ago and became an independent consultant.

Once I was settled back into my home office, I contacted people I knew and in general looked for work. I got an unexpected phone call from a local company that I sometimes contracted jobs through. Was I available for some validation work? The contract company told me that the client had asked for me by name. The reason they'd asked for me was based on my previous experience with medical software. I had a good hunch which client it was, based on the contracting company. Who's looking for me? And what's the work?

The answer was, "The product is a total parental nutrition (TPN) product, and they need someone who can ensure the software testing has been robust enough. Are you interested?"

A few phone calls and a signed contract later, I was paired with a pharmacist to work on reviewing the software testing of a TPN product. A product just like what my mom was now dependent on for staying alive. When someone's in a hospital or rehab center, it's hard to know which device is being used on any given day, especially when the device isn't at the patient's bedside but is instead used at the medical lab offsite or down a hallway in a hospital lab where the IV bags of solutions are built before being dispensed to the patient.

So I don't know for certain whether the software I've tested in this area is the exact software that was used to build the solutions that kept my mom alive. But for me, the experience of seeing her hooked to a device that could easily and may possibly have been the same device with the same software that I tested had an effect on me.

If you've never tested medical software or worked with medical devices, you might want to believe that the testing and the overall development process is more rigorous than it is for any other type of hardware or software. But it isn't, necessarily.

The truth is there is not much difference between the software testing that takes place on medical software and other software. Yes, there are mountains of documentation, internal audits, and a final stamp of approval from the Food and Drug Administration (FDA). But when you peel back the formality and the perceived rigor, testing medical software relies on people and their abilities just as much as any other software or product. There is no special magic behind the testing of medical software; the quality of a product comes from the talents and ability of the team.

I think back to an earlier experience I had testing medical software. The personal conviction, common sense, and pure integrity of a couple of people in particular on the team made the difference. I was an individual contributor on that team, and I am fortunate to have seen integrity in action, which is clearly one of the more beautiful sights to see. For me, in more than two decades of working with software and watching technologies come and go, it is the people that always make the difference. Just because it is beauty that cannot be seen on a canvas nor heard at a symphony doesn't make it less beautiful. I believe it is a story to be told.

From the Inside

It was just a few years ago when I heard about the need for a software tester on a project with two people I had known for some years. It sounded like an unusual project; a project that I might be able to sink my teeth into. It was testing the software that communicates to and directs a medical device. I'd never worked with software that could so immediately affect a person. I recall asking what the worst-case scenario was if the software didn't perform correctly. The answer "patient death" made me open my eyes wide and think hard about accepting the work.

After all, it wasn't but a couple of years earlier that I had been working on an e-commerce site that sold groceries. On that team we occasionally made lighthearted jokes on stressful days that the worst that could happen was a customer would be missing his milk and bread. Before accepting the new project, I recall thinking that there would probably be no lighthearted days on this one. That was a good general realization to have before beginning work.

On a personal level, I knew one of the leads, Michael Purcell, from previous work experience, but I had never worked with him directly on a project. His reputation and work ethic are well known. I'd admired his work from afar, and the chance to work with him on a project appealed to me. Even more appealing was the prospect of being paired directly with him.

The second person I knew was Ed Young. I'd hired him some years before. I remember thinking that even if the work would be intense, I could relax in one regard: I knew I could rely on Ed and Michael. Ethically, I could trust both of them; there was simply no way either person would ship a product he didn't believe was ready. And I also knew that they would listen carefully and closely to anyone working on the project, regardless of the person's stature or employment status. I've seen projects before where contractors were treated poorly and their input barely listened to. There would none of that with Ed and Michael leading.

I wanted to be on this team.

I was given the title of Study Author, as about half of my time would be writing and the other half devoted to hands-on testing. This was a perfect pairing of tasks for me.

In some ways I was part of the entire team working toward ensuring that the product was ready for market. In other ways, I worked alone, which has always suited me well. Although several testers were focused on specific, discrete parts of the product suite, my focus was to be broader. I would test alone, away from the other testers, and test from a wider perspective with a more holistic approach, while the other testers would focus on specifics and details. I sat near the team, but we rarely had the same schedule. Ed and a test lead spent time together coordinating activities while Michael and I coordinated our own efforts. And yet we communicated as an entire team frequently.

Notably, on every project I've worked on that I would consider successful, constant communication was always a contributor.

I was paired with Michael and functioned in several ways as his right hand. We talked candidly then and we talk candidly now about what my strengths are and are not. Michael is a tough but honest critic, and this includes his ability to critique his own work and strengths. Michael and I agreed that one of my strengths was my ability to write. My past experience writing validation documents that could stand up to an FDA audit would be critical. And Michael's ability to edit my writing proved to be another perfect pairing. (In fact, as a personal request, Michael was my preview editor of this chapter.)

We recognized that Michael knew the product better than I did, but we also believed I had a more comprehensive testing background compared to his engineering background.

It's interesting that physical beauty often includes a sense of harmony and symmetry. These pairings—testing and writing, testing knowledge versus engineering knowledge, low-level specific component testing versus high-level general system testing—brought symmetry to our work. These were forms of checks and balances.

Many times, Michael would explain his testing ideas to me, and at least two things would take place during these exchanges. One was that I was slowly and in detail learning about the product. And the second was that we would use the strengths, insights, and experiences we each had while we took turns raising challenging questions. Questions such as: How could we

test better? What else can we do to challenge the product? What else can we do to gain faith that the product will work well in production?

One of the subtlest aspects of working on a project is the atmosphere. It doesn't seem to be something that is discussed often. It seems we are more engrossed by the technology or the tools or the raw skills of the people, not to mention how often we focus on the deadlines. And yet it's the atmosphere on a project that I believe contributes to my ability to do my best work. Working with Michael I've always felt like I can muse out loud, share my thoughts—really brainstorm without recrimination. I can go out to the far edges and toss ideas out. And of course, after chatting over ideas, we do toss a fair amount of them out. But I think that in order for team members to generate test ideas, they need wide-open space to explore what might be possible before they can lasso those ideas into reality.

And so, this is how we worked. We worked together; we worked alone. We constantly reverted back to guiding questions and challenged each other and our ideas. We kept plodding on, doing our best.

And yet, something was missing.

Adding Different Perspectives

There was a day when the three of us, Michael, Ed, and I, were talking in a hallway. We were talking about how each of us felt pretty certain there were still defects in the product. Even though the test team had been diligent in testing, something was missing. Where was our confidence and conviction that the product was ready? We each had the sense that there were issues still lurking. It was more than a hunch; we each knew of a few bugs that hadn't been consistently reproduced and corrected. Each of the testers felt the same: the product wasn't ready yet.

How could we solve the problem?

From a business perspective, Ed worked to get the approvals and financing needed to implement our new ideas. It's easy not to think about what efforts Ed had to work through with management, because he gracefully kept those concerns from draining our energies. Michael and I remained focused on the testing efforts.

Like many complex problems, it took more than one solution. First, we added a different approach to testing. We wanted an opportunity to follow through on our hunches where issues remained. A new team of testers would be brought in to execute testing in an exploratory approach while the existing testers who had been working with prepared written test scripts would continue. The plan was to keep the scripted testers executing the test scripts that would be required for the FDA audit before product launch and to have the second team find what was missing. The second team of testers would be trained, fed hunches, and would primarily function as exploratory testers.

Once the decision to bring in more testers was made, the urgency on the project picked up pace. A training room was turned into a makeshift test lab. The room had several rows of workbenches with computers. The computers had the needed mix of operating systems and software already in place. Arrangements were made so that the new test team could have the training room without disruption for several weeks.

Unfortunately, the computers were all desktop models and cabled such that moving any equipment required cutting cable wraps. The workbenches were narrow and the spaces between the computers limited, so keeping a notebook alongside a computer was difficult. The room was never designed for working such long days, and the days were long. It was a windowless room with limited unused space. And by having so many of the computers working so many hours per day, the room could get hot and sticky, even though it was winter.

Still, the room had energy. Every person working there, whether all day or for brief bursts of time, knew they were there for a reason. The work was clear. When the software you're testing is responsible for what dose a patient receives and death is a very real risk, there is a sense of purpose that's difficult to articulate but a feeling and sensation that hangs in the air each day.

The exploratory testers were hired quickly. They had less time to learn the product than the scripted testers, but they received more personal interaction from various members of the team who were anxious to get them up to speed. Before long, there was a testing frenzy taking place, with both teams of testers working to get the product in shippable shape.

Another solution we came up with was to build and use a scientific test lab that would be equipped with the medical devices and the computers for testing, which would more closely simulate real use. The science lab would require space and equipment that hadn't been planned previously. The theory was that we would rigorously exercise the software using yet another test strategy. Instead of focusing on details and purposely looking to discover issues at the micro level, the testing planned for the science lab was designed to find issues at a more macro level. We needed time to explore all the components together.

Another solution was to carve out time to purposely test from a multiuser perspective. We knew that in a busy medical lab multiple people would be using the software in bursts of activities. We wanted to test resource contention, file-sharing issues, and periods of stress on the system in specific ways that we believed would mimic real-life use. I had previous experience testing software in multiuser scenarios, and even though my experience was not related to medical software, that background was helpful in the planning process.

Exploratory testing, stress testing, multiuser testing, and real-life simulation are all different ways of exploring software. By altering focus, different views and issues can be found and exposed. In our case, none of these approaches were frivolous or far-fetched; in fact, each form of testing was designed to be as practical and realistic as we knew how to make it.

Exploratory, Ad-Hoc, and Scripted Testing

Cem Kaner first introduced the term "exploratory testing" in the book *Testing Computer Software* (Wiley). In his use of the term, he offered an approach that emphasized the value of testing as a brain-engaged, thoughtful process, as opposed to testing by executing prescriptive test scripts. Since then, software testing expert James Bach has devoted more than a decade to teaching, writing, and presenting on the topic of exploratory testing. The following is a definition of exploratory testing taken from James's website:

> The plainest definition of exploratory testing is test design and test execution at the same time.

With exploratory testing, test charters are written to focus testing and provide a strategy for a test session. Session-based exploratory testing adds the concept of defining a length of time for the session. Testing is executed in sessions where the focus remains primarily, if not exclusively, on the test charter. Within a test session, a tester can explore, create ideas, and execute these ideas, providing a sense of test coverage and confidence. And when it's not possible to finish all of the ideas in one test session, additional test sessions can be planned. Exploratory testing takes discipline.

There are elements from this definition posted on Wikipedia that are worth reflecting on:

> Exploratory testing seeks to find out how the software actually works, and to ask questions about how it will handle difficult and easy cases. The testing is dependent on the tester's skill of inventing test cases and finding defects. The more the tester knows about the product and different test methods, the better the testing will be.

The second group of testers used several different approaches to testing, including, but not limited to, exploratory testing. For their exploratory testing, the testers executed focused sessions. I think one important differentiator between ad-hoc testing and exploratory testing is focus. The testers were trained on the product and had access to everyone on the team, but were then left to their own thinking process to create test ideas, investigate, test, and discover. The exploratory testing was certainly reliant on the thinking and skills of the testers.

As testers completed test sessions, they would discuss their findings, resolve their questions, and report defects. Those post-execution conversations determined whether additional testing was needed and shaped those next steps. Those conversations provided valuable learning sessions—learning for the testers as well as for the other people on the team.

James Bach uses the term "debrief" to describe a conversation that can be held after testing that helps draw out information from the tester. He offers a debriefing checklist on his website at *http://www.satisfice.com/sbtm/debrief_checklist.htm*. These post-execution conversations that we held did not follow James's checklist of ideas. Instead, the questions and conversations were more specifically tailored according to the tester, the topic, the findings, and the overall context of what was taking place and what metrics the team was interested in maintaining.

There is no advantage in departing from the formality of scripted testing to adopt exploratory testing without adapting as needed. There is beauty in finding your own path, in understanding what is needed and applying what makes sense.

Another alternative to prescriptive test scripts and exploratory testing is ad-hoc testing. Ad-hoc testing is often confused with exploratory testing, but there are essential differences between the two.

James's site offers the following clarification of the difference between ad-hoc and exploratory testing:

> Exploratory testing is sometimes confused with "ad hoc" testing. Ad-hoc testing normally refers to a process of improvised, impromptu bug searching. By definition, anyone can do ad-hoc testing.

With ad-hoc testing, testers execute test ideas as the ideas tumble into their heads. The core advantage of ad-hoc testing is that everyone can test. Each tester's primary ideas have an opportunity to be explored. But this might be where the advantage of ad-hoc testing ends. Most people untrained in testing run out of ideas rather quickly, and so, after a short burst of test session, they're done. Untrained testers often cannot repeat the steps they took to find a defect. And with ad-hoc test sessions, testers bounce from one area of an application to another area; it is hard to understand what's been tested and what remains to be tested.

But ad-hoc testing has its advantage in generating energy, especially when it's executed in "bug bash" sessions. We used this tactic on our project as well. Someone on the team would share an idea about where a defect might remain, and then all the testers would pursue that idea for a short period of time. As people tested, they would talk out loud, and impromptu brainstorming would take place. Everyone had ideas. One benefit of having the second team of testers and a separate test room was having the chance to explore ideas and to participate in unplanned test activities. If one person shared an idea, everyone in the room could jump in and add to or extend the idea in some way.

Other tests sessions were executed when a tester was paired with someone on the team for a conversation and generating ideas about potential issues. Some of those ideas came from the scripted testers who had either a suspicion about a defect or had encountered an issue but had not been able to replicate the defect. Those ideas were shared in conversations before testing, and that same spirit of collaboration continued post-execution. The scripted testers were encouraged to share their test ideas with the exploratory testers. The scripted testers could continue the required test execution work, and the exploratory testers could work on hunches.

And yet another type of testing took place when the exploratory test team looked at the scripted tests for a particular area of the product and generated more ideas after reading the scripts. In part, the test scripts provided training through having such detailed test steps written out. By reviewing the scripts, testers could extend their ideas based on whether those tests would include different data or different permutations, or sometimes they would just provide food

for thought. Is that ad hoc? Is that exploratory? Each of these approaches to testing had its positive impact. Bugs were being found and repaired, and our confidence was rising.

Multiuser Testing

I have a vision in my mind about performance and multiuser testing. The vision certainly applies when test automation is being used, but also when manual multiuser testing takes place. I envision multiple test automation scripts running at the same time, each script simulating multiple users performing a set of activities. I see each script running like a bar of color, like radio frequency waves. The scripts run like music, in a crescendo, the peaks coming close together, almost colliding and creating spikes, and at times running wildly with different peaks and lulls in a cadence of their own. I think of it especially when I hear full symphonic pieces of music, the sense of harmony, of working together.

Each instrument adds to the overall symphonic effect, just as each manual tester adds to the overall impact on the system under test. Even a single manual tester alters the environment by simply being on the system. It takes the entire orchestra to achieve some sounds, just as it takes a multitude of scripts or manual testers to achieve a production load simulation. The activity is no longer about one tester, but rather is about what is achieved as a collective.

Multiuser testing is not the same as performance testing per se. Performance testing is often focused primarily on transaction timings. In the case of multiuser testing, our goal was not about timings, but instead focused on what happens when multiple people execute a specific activity at the same time. Overall, the multiuser test ideas were about preventing data corruption, record contention, duplicate records, or system crashes.

The test concepts were sobering reminders of the importance of thoroughly testing a medical device. Although a test condition might have been, "Let's check race conditions on editing a patient record," underneath that test condition was a possibility that two lab technicians could edit a patient's record at the same time and cause either corrupted data or leave patient data in such a state that an inaccurate patient prescription could be created—and should the prescription be dispensed to the patient, that patient could die. Such was the case with nearly every test condition in the multiuser area of testing, and the potential consequences were chilling.

One advantage of having the exploratory testers in one physical room was that we could coordinate multiuser tests more easily. We'd speculated about a few issues in this area, but we didn't gather proof of how ugly it could be until we had the ability to orchestrate multiple people with multiple PCs running through multiple tests. We had accumulated a collection of ideas, and now we had the means to execute.

Michael crafted a set of multiuser tests before the exploratory testers joined the project. I had worked with multiuser testing previously, and we had discussed ideas. Our collection of ideas was centered on making sure patient data and patient prescriptions would not collide and could

not be corrupted. Since the application prevented more than one instance running on one computer, it had not been possible to easily plan or conduct this type of testing by any one tester alone at his desk with only one computer. We did have a small test lab with multiple workstations, where some multiuser tests had been attempted, but not with this much focus or attention and not with the ease of access to multiple computers and multiple testers whose activities could more readily be orchestrated.

The test suite grew significantly as team members added new ideas and "what if" scenarios. The test execution sessions had energy. One idea would spawn another, and that one yet another. Everyone had ideas and everyone contributed. The opportunity to brainstorm had been revived. No longer was testing disengaged from thinking, as is often the case with executing prescriptive test cases. Testing in the makeshift test lab had a professional atmosphere focused on finding and sharing ideas. When an intelligent group of people gathers together and is given the time and space to explore, the desire to come up with great ideas and to be a strong contributor flourishes. In contrast, isolated, solitary, mechanical test execution reduces brain engagement by virtue of the need to execute and prove that the test script was not varied in the execution process.

Upstairs, the scripted testers plodded along. Piles of executed test scripts stacked up with their traditionally daunting impression that everything had been tested. But the energy and the ideas flowing in the training room made it apparent that all the test ideas had not been thought of before and were not captured by the test scripts. Rather than viewing the test scripts as defective or inadequate, the general sense of the team was that the software and the devices offered more permutations and possible flaws than might ever be tested, found, or addressed.

We might want to believe that all the testing has been executed, every bug has been found, and perfect software has been achieved, especially when it comes to a medical device. But perfection is not reality. Instead, assurance is found in knowing that a product works well and that, even without perfection, beauty exists.

With the mix of testing activities taking place, we were in fact finding issues. And in finding more flaws, our confidence that we were seeing the product more accurately increased. Better to find defects and argue about what needed to be addressed than not find the defects before shipping.

Most of the multiuser tests were executed with someone standing at the front of the room and explaining the test. Each tester would try out the software to make sure they understood what was being discussed and how to execute what was needed. The person leading the test would then orchestrate the testers to discover "what would happen if…?"

The testers worked through multiple executions with the same goal until we had good answers to the following questions:

- Did each tester understand what she needed to do?
- Was the test conductor satisfied that the timing and execution had achieved the goal?

- Had the test been executed enough times to conclude what the results were?
- Were any issues discovered?

In addition to the multiuser testing, testing was run to ensure that a busy medical lab with a volume of data, patients, and prescriptions would be ready to handle the anticipated load. At a high level, we knew what the correct responses should be: no duplicate records, no race conditions, no data corruption, and no system hangs. Beyond that, no tester needed specifically written expected results to know when something was wrong.

These tests were later turned into formal test scripts. What had been exploratory at the start became repeatable scripts. The value of turning the testing into repeatable scripts was the assurance that this form of multiuser testing would be executed any time the software was released in the future. This is an aspect of working with regulated software: test scripts deemed as essential for one release cannot be dismissed for the next release without a review and explanation. It had become clear to us that this form of testing needed to remain as part of the overall product testing for all future releases.

We couldn't ensure the creativity and energy of the future testing, but we could ensure that multiuser testing would be considered essential. To have a team able to function in such a way that it can achieve something no one person can achieve alone is, to me, the beauty of teamwork. For each of these sessions I participated in, and the multiuser testing sessions in particular—even when the testing found no issues—the sense of teamwork had immeasurable positive and lasting effects.

The Science Lab

Most of the testing throughout the project took place with just the software, and a smaller amount of the testing took place with the software and the medical devices paired together. But Michael felt that the full product couldn't be validated without a realistic workout, and that workout would be bringing all the components together in a lab with multiple days of heavy full usage.

Michael was in the process of designing what, to this day, I still think of as "the science lab." The intention of the lab was to pair computers with the medical devices. Once the lab was assembled, we paired the exploratory testers and the scripted testers for simulated use of the medical devices and the software in a scenario that mimicked real life as closely as possible. The lab testing days were planned as the final formal testing of the product.

Michael had planned every aspect of the science lab with some assistance from me. It was no simple task to think through, order, and arrange computers, software, cabling, workbenches, medical devices, more cabling, fluids, IV bags, tubing, printers, and labels. The room was also equipped with a laminar flow hood, which is a workbench designed to be partially enclosed to minimize contamination of the fluids being worked with. It can be loud when the air filtration is running, and it also takes up considerable space and adds some complexity to

working with the devices. But Michael felt that the flow hood was necessary, as most labs would have a flow hood and some, if not all, patient prescriptions would be built inside a flow hood. Each tester would be trained to work with it. For the computers, the network connections had to be configured, as did the software. For the medical devices, each device required some assembly and calibration before use.

Then it was time to assemble the lab, and we spent several days doing this together. I recall Michael's practical suggestion for the lab assembly days: wear jeans and bring a pocketknife if you own one. We cracked open the boxes for computers and the medical devices. I learned more about the devices while configuring the equipment. As we set up, we discussed the logistics of the coming days.

The day we finished setting up the lab, I stood at the door and looked back to see how everything looked. The lab was a large rectangular room with concrete walls. The back wall had a large sink and a long counter. In the middle of the room were two long worktables with two rows of carefully configured devices. Overhead, a long wire-framed tunnel loomed with countless cables running from devices to computers and from devices to printers. The computers were configured. The printers had paper. The devices had been calibrated and there were rows of fluids ready for use. The laminar flow hood had been set up and configured. The lab looked clean, organized, and ready.

At the end of the day, I taped a paper sign from the inside of the glass door. The sign had the project name and a short list of who to call for access to the room. We shut the lights off. Michael headed out for the day. I recall standing for a short time outside the lab door, peering through the glass, ready for the coming days of testing.

Simulating Real Use

A test execution plan was laid out for the testing days at the science lab. The core of the plan simulated multiple full days of real-as-possible use of not just the software, but also the software that worked with the devices and the devices that generated the solutions. We wanted to test what might be thought of as "end to end" testing, at least as far as we could take the simulation. After all, we didn't have patients. But we wanted the testing to be comprehensive and holistic. We also had an underlying desire to see the software and the devices working in harmony for multiple hours and days in a row. And although we weren't trying to stress-test either the software or the devices (this had been done previously), we did want testing to span full and hearty days of work.

Unlike many days on the project when people would work alone, the testing days at the science lab were very much a team exercise. Everyone had a role, and each person knew what work she needed to execute. Prior to the days in the lab, we'd mapped out a plan. We briefed everyone on the team and discussed the plan multiple times in advance, to both smooth out the plan and to incorporate ideas and feedback from the team. We wanted to be ready to rock and roll when we hit the lab.

Two teams were planned. Each team represented a technical configuration that would be used in production, the theory being that each team would mimic a medical lab with a specific computer, software, and medical device configuration. The overall lab would be busy, printers would have other prescriptions in the queue, fluids would need to be replaced, tubing that would get a heavy workout would need to be flushed and possibly changed. We planned the days to be busy, robust, and even a bit hectic in spots, just as a medical lab might be.

Each team had multiple patient prescriptions to fulfill, with each prescription representing a variation we wanted to address. And, of course, that itself could be a deviation from real use. Perhaps on an ordinary day in an actual medical lab there wouldn't be this much variation of prescriptions.

Understanding, knowing, and designing testing around real-life scenarios can be challenging. To know how a product will be used in the field by actual users means you have to gain insights from the user perspective. It can be particularly challenging if the product has never been released to production.

Even on products that are already in production, finding out what users really do with the software can be difficult. In the case of a web application, it might be possible to get that information from server logs. In the case of medical devices, we can't just ask a hospital or a lab for their production logs. Sensitive patient data is well protected (understandably) by the Health Insurance Portability and Accountability Act (HIPAA) and other regulations.

But what if we're not interested in the patient data? What if we just want to do the best possible testing and we believe having insight into field use will give us that? I've wanted to shout this question to some unknown group of medical doctors and lab specialists, but the short answer is that I've never seen that dilemma solved.

It would be great to have the test lead of each medical device paired with a medical practitioner to see real-life events unfolding and learn how a device or software is used under stressful, tight timing, and intense situations when the need is critical.

I envision a wonderful pairing: an opportunity for testers to better understand a product's use and for an end user to be able to give insight to a tester in a way that no requirement document or any use case could ever impart. Why isn't this just viewed as a practical, grounded solution? And why would the software testing community and the actual medical practitioners not be encouraged to make this happen?

Instead, as testers, we earnestly try to create scenarios we believe are realistic. But what do we know when we're sitting in an office surrounded by office workers and far away from the setting a product will be used in? We do the best we can.

In life, we don't know when a memory is being created. It isn't until time passes and we see the events that stay with us become the memories that we keep. I can recall the days in the science lab vividly in memory.

It seemed that everyone on the team was just a bit nervous, a bit anxious, and more willing to help each other than ever. There was an atmosphere that's hard to describe. Maybe a collection of words is the best way to describe those days in the science lab: serious, intent, stressful, exciting. It also seemed that we wanted each person to be able to execute what was needed. The sense of camaraderie was strong. We moved together. If one person needed a break, the whole team would have to wait. We arrived together, we worked together, we ate lunch together, and we couldn't wrap up for the day unless the whole team was ready.

The days in the lab were long. People made mistakes. We documented what we did. We had two team members on hand who did not execute but were there to collect documentation and review materials as they were generated. I think we each knew that we would have to uncover something truly significant in order for these days to be repeated. We hoped we were long past that part of the product cycle.

Instead of testing the software in small, focused areas, and thinking about the software from a technical perspective, designing simulated real-life scenarios has the beauty of stringing multiple aspects of usage into full-length scenarios. There was a hope, and certainly an intent, that full-length scenarios would flush out defects not found until this type of simulated usage was executed.

After all, this is how the product would be used.

Testing in the Regulated World

In 2001, my boss Jim Kandler explained how to test software in a regulated world. His explanation to me (referring to the FDA) was: tell them what you going to do, do it, and then prove you did it. "Gee, that's it?", I can recall thinking. But it's a lot harder to do than it sounds.

What Jim meant by "tell them what you going to do" is this: document the process. Document it for multiple people—the team who will execute the process, the company so that they know the team has a process, and of course, for the FDA. I was taught that an FDA auditor will review the process documentation as one of the first parts of an audit and then ask for evidence that you've followed the process. Some of the other documents they request first are defect reports, the trace matrix, and the final validation report.

What I've found is that detailing the process you believe the team follows is harder than it sounds when you get down to the nuances, not to mention the exceptions that occur in real life.

In the regulated waterfall approach to software development that I've seen, requirements are drafted. Requirements are from a business perspective, the patient perspective, the customer perspective, and from a systems point of view. Often people write them without thinking about testing or about how they are going to prove the requirement. Often people write them without a software tester's involvement.

The requirements evolve into design specifications, and since the business analyst often cannot address the technical implementation, the design is usually written by development. One of the classic issues is that development often writes design specifications ahead of time, so they haven't yet hit the roadblocks of implementing the ideas. The design gets written ahead of time, ahead of knowledge, and then the documents go through a chronic revamp process, which leaves the documents in a state that is less than helpful.

It seems to me that when unregulated software is being developed, open conversations about possible remaining bugs are fairly common. Conversely, when the product undergoes FDA scrutiny, those open conversations are less easy to have for political reasons. For regulated products there is a formal process for documenting, reviewing, and resolving defects. But even with a regulated product, there is tremendous momentum in getting a product to ship. To announce late in the process that more tests could be done or that possible defects exist creates unsettled circumstances. Everyone knows that if critical bugs are found, those bugs need to be fixed. Opening up the code requires reexecuting numerous test cases and possibly extensive product release documentation adjustments. The cost can be significant. Pressure mounts. It's exactly a condition in which creating and asking to execute additional testing can be difficult. And it is exactly a condition under which more testing should be done—especially on products that have such critical outcomes.

Nonemployees are rarely part of the FDA audit at the end of the product development cycle. The lead person who works with the FDA to move the product through the audit is carefully selected. The lead person needs a mixture of regulatory experience, product experience, and enough knowledge about the team to assemble whatever additional information is needed rapidly and with confidence. I've never witnessed a hands-on tester pulled into the process.

In 2001, one of my first training experiences was being sent to FDA auditor training. My boss, Jim Kandler, wanted me to understand firsthand what an FDA auditor was trained to look for. Beyond the papers, if I were an FDA auditor myself, I'd want to talk to the testers from the team. I'd want to hear directly from the people who touched the software and got to know the product. I would not want to be left with a designated spokesperson.

What happened behind the doors of the FDA audit for this particular product, I will never know. I know the product is in use on the market.

At the End

On the last day of testing at the science lab, we lingered about. Most of us were working as contractors. We knew once the testing was done we'd roll off the project and likely never work as a team again. This is part of a contractor's work: projects end and people move onto other projects. Some people keep in touch, some don't. I like the frequent change. I like being on a project long enough to see what works and what doesn't. I like to get to know people. I like to see products launch and be used.

The days in the science lab were some of the last project days for the contractors. The overall product release moved forward with activities such as final validation documents, manufacturing considerations, and delivery details before final product launch.

I wasn't prepared to see a total parental nutrition product used to care for a family member. But I'm grateful that in this imperfect world, people do their best to create products that matter. And each of those products needs people who can test.

APPENDIX

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 postering 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.