

/THEORY/IN/PRACTICE

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

Beautiful Testing

Leading Professionals Reveal
How They Improve Software

O'REILLY®

Edited by Tim Riley
& Adam Goucher

Beautiful Testing

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

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

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

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

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

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

This book includes contributions from:

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

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

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

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

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

US \$49.99

CAN \$62.99

ISBN: 978-0-596-15981-8



9

Safari[®]
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
	Multiuser 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	269
	<i>by Tomasz Kojm</i>	
	The Clam Anti-Virus Project	270
	Testing Methods	270
	Summary	283
	Credits	283
21	WEB APPLICATION TESTING WITH WINDMILL	285
	<i>by Adam Christian</i>	
	Introduction	285
	Overview	286
	Writing Tests	286
	The Project	292
	Comparison	293
	Conclusions	293
	References	294
22	TESTING ONE MILLION WEB PAGES	295
	<i>by Tim Riley</i>	
	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	303
	<i>by Isaac Clerencia</i>	
	The Need for an Advanced Testing Tool in eBox	303
	Development of ANSTE to Improve the eBox QA Process	304
	How eBox Uses ANSTE	307
	How Other Projects Can Benefit from ANSTE	315
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

Testing Network Services in Multimachine Scenarios

Isaac Clerencia

The Need for an Advanced Testing Tool in eBox

WE HAD BEEN DEVELOPING eBox for a bit over a year when we started to struggle to get new releases out. The main problem we faced every time we had to release a new version was the testing and quality assurance (QA) process, which had quickly become the lengthiest and most dreaded task among eBox developers, although it did face tight competition from documentation writing in that honor.

The eBox platform is a complex open source web tool to manage corporate networks. It integrates some homegrown services, such as network configuration, a firewall, or traffic shaping, and well-known existing services such as Squid, Samba, or OpenLDAP. It is released as a set of independent modules that have to be thoroughly tested to prevent regressions and to verify that every new feature works as expected.

Even though eBox just offers a simplified interface to these services, the number of test cases still grew overwhelmingly fast as new modules were added and existing ones got more features that required new tests.

Another problem we faced was the complexity of the scenarios required to perform the tests. In the beginning, most modules needed quite simple scenarios, such as the proxy one, which

only required two machines, one client to try to browse from, and the proxy server managed by eBox that we wanted to test.

Later in the development process, new modules requiring far more complex scenarios, such as the OpenVPN module, started to appear. This module can be used to connect different offices or to allow road warriors to connect to a central office. To test the functionality of the module in certain setups, up to six machines may be needed. The tester not only had to install all these machines, but also had to configure each one of them for its particular role.

This configuration process was not only lengthy but also error-prone, as it required a lot of human intervention, including writing configuration files and setting up eBox through the web interface. Once the scenario is set up, determining whether the test has been successful is not a trivial task either. Appropriate tools have to be used for some modules, such as the traffic shaping one, to determine the outcome of the test.

To add to all these problems, eBox needs to be tested in several Ubuntu Linux and Debian GNU/Linux releases. Even if eBox is mainly targeted at the latest Ubuntu LTS (long-term support) release, we also release packages for the latest Ubuntu regular release and the latest Debian release to try to grow our user base. In addition, while making sure that all these releases work correctly, we have to start testing packages for the upcoming releases, to make sure our packages are ready as soon as the new releases are out. All of this adds up to a total of five different operating systems that we have to run tests for at a certain point.

The last problem is due to the fact that even if most members of the eBox team are in Spain, a few of them live in other countries. This makes it very difficult to have the required testing infrastructure in each of the different locations. Because of this, the ability to remotely schedule tests was an important requirement.

It can be argued that it is relatively easy to address some of these problems partially, for example, using virtual machines with predefined images to ease machine installation or writing scripts to automate configuration tasks. But even with these improvements, testing eBox would still have been a tiresome task. We wanted to aim for a beautiful solution that saved humans from having to do any daunting, repetitive task.

Development of ANSTE to Improve the eBox QA Process

We had decided to try to achieve full automatization for our tests. After a disappointing search for an existing solution for this problem, we set out to develop our own [open source](#) testing suite. The new product had to be strongly focused on our requirements, but we also wanted to keep it flexible enough so that it could be used to enhance the testing process of other software projects.

The set of requirements was very well defined, as we had been manually testing eBox for a long time by then and were fully aware of all the pitfalls in our testing process. Based on the extensive list of requirements, we were expecting a long development process. Fortunately,

building it on top of other open source software allowed us to speed up the process considerably, and after just three months of frenzied development by only one person, we had a tool that we could already use in our testing efforts. We named it Advanced Network Service Testing Environment, or [ANSTE](#).

The first feature developed in ANSTE was the ability to easily define complex network scenarios. Scenarios have the information about all the machines that will have to be created for a given test suite. The information for each machine includes basic details such as the amount of memory and hard disk space the machine will have and its hostname, but also more complex information, such as the number of network interfaces and their configuration, as well as the routing rules to reach other networks.

Defining the details for every machine in every scenario would be a daunting task, and in addition, building operating system images is a long process that takes a lot of memory and disk space. Thus, in order to avoid having to build images more often than strictly needed, the scenario framework supports inheritance for machine definitions, allowing the user to define as many base images as needed and that hosts can later inherit from.

A base image definition includes a name to refer to the image and an installation method. A program called *debootstrap* is used to install Debian-based distributions. This program is able to install a basic system on an empty disk, pulling the required packages from Internet mirrors and installing them in the new system. This installation method requires only one parameter: the name of the distribution to be installed.

Base images can be reused in different scenarios with just minor changes. Every machine in a scenario declares which base image to use, and then, if changes are required in the image, it specifies the values of the parameters that need to be different. Besides the network configuration, one of the most important parameters for customizing a machine is the one containing the software packages desired for that machine. As many as needed can be specified, and they will be installed on it using the system package manager once the machine has been set up.

ANSTE cannot provide a parameter for every minor detail that could be potentially configured in the installed operating system, so in order to allow further customization of the machines, scripts can be declared and will be run before and after performing the package installation.

These network scenarios will be automatically deployed when a test suite needs them. Scenarios are transformed into a number of configuration files for libvirt and a set of scripts that will be run at the right time. libvirt is a library developed by Red Hat that can interface easily with the different emerging virtualization systems. By using libvirt in ANSTE, different backends, such as KVM or Xen, can be used.

For every machine declared, a virtual machine is created and started using the preferred virtualization system, trying to replicate as closely as possible a real scenario, including the network links between machines. ANSTE is smart enough to put virtual machines in different virtual bridges to simulate the physical separation between the machines. This actually allows

ANSTE to work correctly with daemons that rely on broadcasting packets over the network, such as DHCP or Samba.

Once a scenario is correctly designed, it is possible to start running tests on it. In order to do that, test suites are defined. Test suites are simply files containing a set of tests that will be run sequentially on the same instance of a scenario. Running a few test suites at once is a common task, so a suite file can be used to aggregate several of them.

It is worth noticing that not all the tests in a suite are tests as such; some of them rather have a utility nature. The reason for this is that some individual tests may require certain preconfiguration before they are actually run. This configuration is usually performed using other tests, which could be seen as simple scripts. It could be argued that a test that requires the preconfiguration could do it by itself, but it is quite usual that the same configuration steps are required by different tests, so insulating them in a different script allows further reusability.

The tests in a test suite define several parameters. These include the machine where they will be executed and a directory containing the actual test. These directories can contain two types of tests, command-based and Selenium-based, which can and often will be mixed freely in a suite.

The first kind of tests, command-based ones, merely execute a command named `test`, which should be in the provided directory. If the command returns zero, the test is considered successfully passed; otherwise, it is considered to have failed. The use cases for these tests are really varied. For example, they can be used to modify configuration text files, restart involved daemons, or run any other command to make a functionality check.

Selenium-based tests use a web application testing framework called [Selenium](#). This framework is used to perform configuration steps using the web interface and to check the results of these actions. The main advantage of using Selenium is that it makes the tests not only cover the program logic but also the web interface.

Selenium tests are written in Selenese, a language that makes use of HTML tables to define a sequence of actions to run. A file named `suite.html` in the test directory is read first, and it declares in Selenese the rest of the files that are part of the test. These other files contain actions such as opening a given URL, clicking on a link, filling and sending a form, or checking whether a particular string exists in the response.

Network testing frameworks always have to deal with the problem of synchronization between the different machines. For example, most scenarios will involve at least one machine having to wait until another one is ready. Other frameworks opt for complex solutions, usually sending the jobs to every machine in parallel and then relying on synchronization primitives to coordinate the execution in the different machines. eBox goes for the simplest option: running the tests sequentially, with a master (the host machine) explicitly telling each machine when it should run each test.

When machines are deployed in a scenario, they notify their availability to the master, which will not start running tests until all of them have been successfully started. If it is important to make sure that a certain action has taken place in a machine before the master is notified, a post-install script can be defined that should wait and check whether such an action has already occurred.

As tests are run in a serialized manner, if we want two actions to be run in parallel, we have to write tests to launch processes asynchronously and leave them running in the background. Then it is the responsibility of the test writer to stop these processes when they are no longer needed, writing another test for that task.

Once the tests are ready, they can be scheduled to be run and ANSTE will notify the tester when they have finished. A full report is made available through a web interface. The report includes all the information that the developer may need to check the outcome of the tests. First of all, the result of every test is provided. In addition, it also has the output logs for all the scripts that were run during the test and a video recording of the browser for the tests that failed.

How eBox Uses ANSTE

Once ANSTE was ready, the next step was to introduce it into our testing process. A machine powerful enough to deploy even the largest scenarios was purchased, and ANSTE was installed on it.

Module developers started to write ANSTE tests for every feature that needed to be checked. Initially the tests were run only when new beta releases of a module came out, but soon developers wanted to take advantage of ANSTE to do preliminary tests as soon as they developed new features.

In the beginning, a first-come, first-served approach was taken, but it was soon obvious that it was not an optimal solution. Developers had to coordinate manually to share the access to the machine and check whether someone else was running tests at the moment.

To address this problem, a scheduling daemon was developed. Testing jobs can be submitted to this daemon by ANSTE users. Each job has a priority, which allows the scheduler to run them in the appropriate order. Users have different maximum priorities, so the release manager can schedule jobs with a higher priority than regular developers, as well as change the priority of any scheduled test. The scheduler might run some jobs out of order to try to optimize the use of resources. For example, if a test is being run and there is only one job in the queue that has a memory footprint small enough to allow it to be run at the same time, such a job will be processed immediately, regardless of its position in the queue.

When a job is scheduled, the user receives the URL of an RSS feed that can be used to keep track of the test results. In addition, the user can provide an email address and, once all the tests are finished, a notification will be sent to that address.

At the moment, there are 455 tests covering almost every feature in most modules. The whole suite of tests takes around three hours to be run with our current equipment.

ANSTE has been specially useful when merging large framework changes that affected every module. These kinds of merges are very likely to cause unexpected regressions that were very difficult to find in the past. Now almost every regression is immediately detected by the testing framework without any human effort.

Sample ANSTE Tests for eBox

The first example is a test from the proxy module.

The eBox machine acts as a router between the client and the gateway. The test is supposed to check whether the proxy service is configured properly, allowing the client to browse the Internet through the proxy.

The scenario is defined in a file called *proxy.xml*, which includes the machines required for the test. First of all, the router connecting the rest of the machines to the Internet is declared:

```
<host type="router">
  <name>router</name>
  <desc>Internet Router</desc>
  <baseimage>hardy-mini</baseimage>
  <network>
    <interface type="static">
      <name>eth1</name>
      <address>192.168.3.254</address>
      <netmask>255.255.255.0</netmask>
    </interface>
  </network>
</host>
```

A host of type `router` provides the rest of the machines with a connection to the Internet, setting up a network interface and the necessary parameters to provide the connection. A scenario without a host of type `router` will not have access to the Internet. The router uses a base image called `hardy-mini`, which contains a minimal Linux system using Ubuntu Hardy. Another network interface is added so that other machines in the scenario can connect to the router. In this case, just the eBox machine, which is described as follows, will be connected:

```
<host>
  <name>ebox-server</name>
  <desc>ebox server</desc>
  <baseimage>{dist}-ebox-base</baseimage>
  <network>
    <interface type="static">
      <name>eth1</name>
      <address>192.168.2.1</address>
      <netmask>255.255.255.0</netmask>
    </interface>
    <interface type="static">
      <name>eth2</name>
```

```

        <address>192.168.3.1</address>
        <netmask>255.255.255.0</netmask>
        <gateway>192.168.3.254</gateway>
    </interface>
</network>

    <post-install>
        <script>ebox-import-network.sh</script>
        <script>ebox-wait-start.sh</script>
    </post-install>
</host>

```

Variables can be used in the XML scenario definition, as can be observed in the `baseimage` declaration. It uses the variable `dist` so the test can be used without any modifications for every supported distribution.

The eBox base image contains a complete and fresh installation of eBox. These base images are generated once and then copied as many times as needed. The images are always stored in a RAM disk to speed up the process.

Two static network interfaces are set up, and then the post-installation scripts are used to import the network configuration into eBox and to ensure that the testing process does not continue until eBox has been successfully started and is running correctly.

The only machine left to be defined in our scenario is the client that will try to browse through the eBox proxy, declared as follows:

```

<host>
    <name>test-client</name>
    <desc>simple client host</desc>
    <baseimage>hardy-mini</baseimage>

    <network>
        <interface type="static">
            <name>eth1</name>
            <address>192.168.2.2</address>
            <netmask>255.255.255.0</netmask>
            <!-- ebox is the gateway -->
            <gateway>192.168.2.1</gateway>
        </interface>
    </network>

    <packages>
        <package>wget</package>
        <package>netcat</package>
    </packages>
</host>

```

The `hardy-mini` image is used again for the browsing client. The machine is configured to use the eBox previously set up as gateway. This way, the HTTP requests made by the browser will go through the proxy.

Two extra packages are required to perform the tests in this scenario, `wget` and `netcat`. They are marked for installation in the packages section. This way, they will be available for the test scripts later.

The scenario file is ready now, and it can finally be used in the proxy test suite. This suite is declared in another XML file, which defines the name of the suite, a description, and the scenario to be used:

```
<suite>
  <name>eBox Proxy tests</name>
  <desc>
    Contains a set of tests to check
    that the eBox HTTP Proxy module works properly.
  </desc>

  <scenario>ebox/proxy.xml</scenario>

  ...

```

After these fields, the tests follow. The first ones are not functionality tests, but instead Selenium scripts to set up eBox through the web interface, although they also serve the purpose of testing the correctness of the web interface. For example:

```
<test type="selenium">
  <name>EnableModules</name>
  <desc>
    Enable network and firewall modules.
  </desc>

  <host>ebox-server</host>
  <dir>enable-modules</dir>
</test>

```

The *EnableModules* script enables the required modules in the eBox server. Enabling modules is a very common task, so there are scripts to enable each of the different modules that are shared along all the tests and then included as needed. These scripts programatically click on the appropriate interface links and buttons and wait until the actions are finished. The file *enable-proxy.html* contains a Selenium script to activate the proxy module:

```
<tr>
  <td>open</td>
  <td>/ebox/Summary/Index</td>
  <td></td>
</tr>
<tr>
  <td>click</td>
  <td>link=Module status</td>
  <td></td>
</tr>
<tr>
  <td>waitForElementPresent</td>
  <td>squid</td>
  <td></td>

```

```

</tr>
<tr>
  <td>click</td>
  <td>squid</td>
<td></td>
</tr>
<tr>
  <td>waitForElementPresent</td>
  <td>accept</td>
<td></td>
</tr>
<tr>
  <td>clickAndWait</td>
  <td>accept</td>
<td></td>
</tr>

```

The open function tells the browser to load the specified location. Once the page is loaded, we use the click function to make Selenium simulate a click on the link with the text “Module status.” Then we wait until the squid element is loaded using waitForElementPresent. We continue using these functions to click on the required links to finally enable the Squid eBox module.

Selenium scripts are verbose and boring to write, but fortunately they can be written automatically using [Selenium IDE](#), an integrated development environment for Selenium tests. This IDE allows the tester to perform the actions in a browser and have them automatically recorded as a Selenium test.

Selenium can perform a lot of actions besides opening locations and clicking on links. The functions most frequently used in eBox tests include typing into text fields, choosing an element in a selection combo, and checking whether a page contains a given text.

eBox uses AJAX pervasively, and a lot of user actions cause asynchronous requests. These alter parts of the page structure without actually loading it again completely, and it is not always easy for Selenium to detect when one of these requests has finished. To fix this, eBox provides a flag inside the HTML code of AJAX-enabled pages that gets changed when asynchronous requests are completed. This way, Selenium can just monitor this flag and know exactly when the request has finished, allowing the safe execution of the next steps.

Originally the flag had to be reset manually, by running a JavaScript function provided by eBox from Selenium. A click on a link that will cause an AJAX request would have looked like this:

```

<tr>
  <td>runScript</td>
  <td>startAjaxRequest()</td>
<td></td>
</tr>
<tr>
  <td>waitForValue</td>
  <td>ajax_request_cookie</td>
<td>1</td>

```

```

</tr>
<tr>
  <td>click</td>
  <td>filter</td>
  <td></td>
</tr>
<tr>
  <td>waitForValue</td>
  <td>ajax_request_cookie</td>
  <td>0</td>
</tr>

```

The function `startAjaxRequest` is called, which sets the flag, named `ajax_request_cookie`, to 1. Then, we have to wait until the variable value actually changes to 1. Once this has happened, we can safely click on an AJAX-enabled link and just wait until the value of the flag is 0 again.

All these steps are still happening now, but a Selenium extension has been developed that allows us to replace all that code with just:

```

<tr>
  <td>clickAjax</td>
  <td>filter</td>
  <td></td>
</tr>

```

Another common task across eBox tests is configuring an interface as external in the eBox interface. An external interface is one that provides access to other networks that can be potentially dangerous, such as the Internet. eBox services will usually listen only on internal interfaces, and the firewall for external interfaces is quite strict.

This script should be usable from most scenarios that might need to configure different interfaces as external. Selenium does not support passing variables to scripts, but ANSTE provides a simple templating system to do it. Variables can be used to make scripts more flexible and thus more easily usable from different scenarios, as it is done in the next one:

```

<test type="selenium">
  <name>ConfigNetworkSetExternal</name>
  <desc>
    Sets the interface as external.
  </desc>

  <host>ebox-server</host>
  <dir>set-external</dir>
  <var name="IFACE" value="eth2"/>
</test>

```

We set the variable `IFACE` to the name of the desired interface, and ANSTE will replace the variable in the script before invoking it so that it will have the appropriate value when it is actually run.

After these generic configuration scripts, the actual scripts to test the proxy follow. They usually come in pairs, with one setting some configuration through the web application, and thus

testing the correctness of the user interface, and another one testing whether eBox works as expected after saving changes. The next set of scripts are:

```
<test type="selenium">
  <name>ConfigProxyFilter</name>
  <desc>
    Config the proxy with filter mode and forbid a host.
  </desc>

  <host>ebox-server</host>
  <dir>config-proxy-filter</dir>
  <var name="HOST" value="www.filterme.com"/>
</test>

<test>
  <name>FilterAllowDownload</name>
  <desc>Try to download a file from an allowed host</desc>

  <host>test-client</host>
  <dir>test-allow</dir>
  <var name="HOST" value="www.google.com"/>
</test>

<test type="selenium">
  <name>TestLogFilterAllowDownload</name>
  <desc>Tests if proxy logs works ok</desc>

  <host>ebox-server</host>
  <dir>test-logs</dir>
  <var name="URL" value="www.google.com"/>
  <var name="EVENT" value="Accepted"/>
</test>
```

The first script performs two actions. The first one is setting the proxy general policy to filter, meaning that every request will be filtered. The second one is adding a new domain, passed as a parameter to the script, and setting an “always deny” policy for it.

The second script is command-based and is run in the machine named `test-client`. It sets the proxy to be the eBox machine and then tries to download the specified URL using `wget`. Then the script tests whether it was able to download the file specified by the given URL. As it did not belong to the forbidden domain, the script will return a successful status if it did and one indicating failure otherwise.

The last script is again run against the web interface in the `ebox-server` machine. Its purpose is to verify that the access to the given URL was registered correctly in the logs for the proxy module and has an `Accepted` state. As this script is used often to check logs, it is properly parameterized to make it more reusable.

The proxy test suite includes many more scripts to continue checking other features of the proxy module, such as denying requests for forbidden domains or content filtering.

Checking the correct functionality of the Squid module is relatively easy, but other modules require far more elaborate tests. For example, to check whether the Jabber module is working correctly, a small script had to be written using the `Net::Jabber` Perl module. This script creates a pair of Jabber connections, verifies that the authentication is successful, and makes sure communication between the two connections is possible through the Jabber server.

Most eBox tests would have been really difficult to develop from scratch, but fortunately a whole set of libraries and commands is available on Linux systems. Besides the already mentioned `Net::Jabber`, other Perl libraries are used in the tests, for example `Mail::IMAPClient` or `Net::SMTP`. A plethora of small but handy Unix commands, such as `netcat`, `wget`, or `traceroute`, are used as well.

Another module with a difficult functionality verification process is the OpenVPN one. In this case, the complexity comes from the large number of machines involved in the scenario. As tests can affect only one machine, different tests have to interact with each other and leave running processes in the machines. Of course, tests are also required to stop these processes eventually.

First of all, the networking between the machines has to be set up. We have a machine running eBox, which will act as an OpenVPN server; two clients that will establish VPNs to the eBox machine; and a couple of routers connecting the clients with the server.

Once the machines are set up, the first test enables an OpenVPN server using the web interface in the eBox machine. The next step would be to have the OpenVPN clients connecting to eBox, but some things need to be configured before this can happen.

The first problem is that the clients cannot actually connect directly to the eBox machine. A redirection rule needs to be established in the routers in order to have connections to the router in the OpenVPN port automatically redirect to the eBox machine. This is achieved by running a utility test that adds the rule to the firewall in each of the hosts.

The next problem we have to deal with is the configuration of the clients. The eBox OpenVPN module provides configuration bundles that include all the necessary files for a given client. Unfortunately, Selenium does not allow downloading files, so we have to look for another way. In the end, this is done by a utility test that uses the eBox API to create the bundles in the server and then moves them into a place from which they can be downloaded through HTTP. These generated bundles are then downloaded to the client machines.

At this point, the clients are ready to connect. First of all, we want to check whether a client can connect successfully to the VPN server, so we start the OpenVPN connection from the first client and check whether it has been established successfully.

One of the options that needs to be tested is the one that enables client-to-client communication. This option determines whether clients connected to the eBox OpenVPN server should be able to communicate with each other. The option is set to false by default, so we have to check that clients indeed cannot communicate with each other. To accomplish this,

we run a test that starts the OpenVPN client in the second machine and tries to connect to the other client. The test checks for a successful connection and returns successfully in that case. As we want just the opposite, we have to add a configuration option in the test to invert the result.

The next step is making sure that changing the configuration option actually changes the behavior. To do that, we modify the option using Selenium and then proceed to stop the VPN connections and restart the OpenVPN server. Now we can use the same test we used before, just not inverting the result this time, and thus verifying that the two clients can actually communicate between each other.

How Other Projects Can Benefit from ANSTE

ANSTE was developed explicitly to help in the testing process of software that involves several machines connected to a network. Any project that matches this description can easily improve its quality assurance process by using ANSTE to automate it.

One of the projects that can easily take advantage of ANSTE is MySQL. MySQL is a widely used open source database management system. It already has a great testing framework, but it is heavily oriented toward SQL testing on a single host. However, there are some parts of MySQL, such as replication and clustering, that require several MySQL servers.

The testing framework can run multiple servers in the same machine to perform these tests, but although these tests might cover the most usual behavior, they do not faithfully reflect the real scenarios with several machines, as they fail to take into account possible failures in the connections between the machines, which actually is a very important test case for database clusters.

ANSTE has already been successfully used to test MySQL clusters. As ANSTE is designed to deal with several machines, it can replicate more closely a real cluster scenario. For example, a network failure can be easily simulated by bringing down the network interface from a script, making testing this issue a straightforward and completely automated task.

It is quite easy to find further examples of projects that can benefit from ANSTE. There is no need to even look away from eBox, as most of its tests actually check functionality provided by the underlying daemons. Software such as jabberd, the Jabber server behind the eBox instant messaging module, or Squid, the web proxy, are perfect candidates for ANSTE testing. eBox uses and tests only a small subset of the features of these projects, but they have a complete feature set that could be covered by ANSTE tests, providing a thorough and automatic way to enhance the current QA process.

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 poster 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.