Introduction to Software Testing

This extensively classroom-tested text takes an innovative approach to explaining software testing that defines it as the process of applying a few precise, general-purpose criteria to a structure or model of the software. The text incorporates cutting-edge developments, including techniques to test modern types of software such as OO, web applications, and embedded software. This revised second edition significantly expands coverage of the basics, thoroughly discussing test automaton frameworks, and adds new, improved examples and numerous exercises. Key features include:

- The theory of coverage criteria is carefully, cleanly explained to help students understand concepts before delving into practical applications.
- Extensive use of the JUnit test framework gives students practical experience in a test framework popular in industry.
- Exercises feature specifically tailored tools that allow students to check their own work.
- Instructor's manual, PowerPoint slides, testing tools for students, and example software programs in Java are available from the book's website.

Paul Ammann is Associate Professor of Software Engineering at George Mason University. He earned the Volgenau School's Outstanding Teaching Award in 2007. He led the development of the Applied Computer Science degree, and has served as Director of the MS Software Engineering program. He has taught courses in software testing, applied object-oriented theory, formal methods for software engineering, web software, and distributed software engineering. Ammann has published more than eighty papers in software engineering, with an emphasis on software testing, security, dependability, and software engineering education.

Jeff Offutt is Professor of Software Engineering at George Mason University. He leads the MS in Software Engineering program, teaches software engineering courses at all levels, and developed new courses on several software engineering subjects. He was awarded the George Mason University Teaching Excellence Award, Teaching with Technology, in 2013. Offutt has published more than 165 papers in areas such as model-based testing, criteria-based testing, test automaton, empirical software engineering, and software maintenance. He is Editor-in-Chief of the *Journal of Software Testing, Verification and Reliability*; helped found the IEEE International Conference on Software Testing; and is the founder of the μJava project.

INTRODUCTION TO SOFTWARE TESTING

Paul Ammann
George Mason University

Jeff Offutt
George Mason University

CAMBRIDGE UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom
One Liberty Plaza, 20th Floor, New York, NY 10006, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi – 110002, India
79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107172012
DOI: 10.1017/9781316771273

First published 2017

Printed in the United States of America by Sheridan Books, Inc.

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloguing in Publication Data
Names: Ammann, Paul, 1961– author. | Offutt, Jeff, 1961– author.
Title: Introduction to software testing / Paul Ammann, George Mason University, Jeff Offutt, George Mason University.
Description: Edition 2. | Cambridge, United Kingdom; New York, NY, USA: Cambridge University Press, [2016]
Identifiers: LCCN 2016032808 | ISBN 9781107172012 (hardback)
Subjects: LCSH: Computer software–Testing.
Classification: LCC QA76.76.T48 A56 2016 | DDC 005.3028/7–dc23
LC record available at https://lccn.loc.gov/2016032808

ISBN 978-1-107-17201-2 Hardback

Additional resources for this publication at https://cs.gmu.edu/~offutt/softwaretest/.

Contents

Figures

Tables

Preface to the Second Edition

Much has changed in the field of testing in the eight years since the first edition was published. High-quality testing is now more common in industry. Test automation is now ubiquitous, and almost assumed in large segments of the industry. Agile processes and test-driven development are now widely known and used. Many more colleges offer courses on software testing, both at the undergraduate and graduate levels. The ACM curriculum guidelines for software engineering include software testing in several places, including as a strongly recommended course [Ardis et al., 2015].

The second edition of *Introduction to Software Testing* incorporates new features and material, yet retains the structure, philosophy, and online resources that have been so popular among the hundreds of teachers who have used the book.

What is new about the second edition?

The first thing any instructor has to do when presented with a new edition of a book is analyze what must be changed in the course. Since we have been in that situation many times, we want to make it as easy as possible for our audience. We start with a chapter-to-chapter mapping.

First Edition	Second Edition	Topic
Part I: Foundations		
Chapter 1	Chapter 01	Why do we test software? (motivation)
	Chapter 02	Model-driven test design (abstraction)
	Chapter 03	Test automation (JUnit)
	Chapter 04	Putting testing first (TDD)
	Chapter 05	Criteria-based test design (criteria)
Part II: Coverage Criteria		
Chapter 2	Chapter 07	Graph coverage
Chapter 3	Chapter 08	Logic coverage
Chapter 4	Chapter 09	Syntax-based testing
Chapter 5	Chapter 06	Input space partitioning

Part III: Testing in Practice		
Chapter 6	Chapter 10	Managing the test process
	Chapter 11	Writing test plans
	Chapter 12	Test implementation
	Chapter 13	Regression testing for evolving software
	Chapter 14	Writing effective test oracles
Chapter 7	*N/A*	Technologies
Chapter 8	*N/A*	Tools
Chapter 9	*N/A*	Challenges

The most obvious, and largest change, is that the introductory chapter 1 from the first edition has been expanded into five separate chapters. This is a significant expansion that we believe makes the book much better. The new part 1 grew out of our lectures. After the first edition came out, we started adding more foundational material to our testing courses. These new ideas were eventually reorganized into five new chapters. The new chapter 01[1] has much of the material from the first edition chapter 1, including motivation and basic definitions. It closes with a discussion of the cost of late testing, taken from the 2002 RTI report that is cited in every software testing research proposal. After completing the first edition, we realized that the key novel feature of the book, viewing test design as an abstract activity that is independent of the software artifact being used to design the tests, implied a completely different process. This led to chapter 02, which suggests how test criteria can fit into practice. Through our consulting, we have helped software companies modify their test processes to incorporate this model.

A flaw with the first edition was that it did not mention JUnit or other test automation frameworks. In 2016, JUnit is used very widely in industry, and is commonly used in CS1 and CS2 classes for automated grading. Chapter 03 rectifies this oversight by discussing test automation in general, the concepts that make test automation difficult, and explicitly teaches JUnit. Although the book is largely technology-neutral, having a consistent test framework throughout the book helps with examples and exercises. In our classes, we usually require tests to be automated and often ask students to try other "*-Unit" frameworks such as HttpUnit as homework exercises. We believe that test organizations cannot be ready to apply test criteria successfully before they have automated their tests.

Chapter 04 goes to the natural next step of test-driven development. Although TDD is a different take on testing than the rest of the book, it's an exciting topic for test educators and researchers precisely because it puts testing front and center—the tests become the requirements. Finally, chapter 05 introduces the concept of test criteria in an abstract way. The jelly bean example (which our students love, especially when we share), is still there, as are concepts such as subsumption.

Part 2, which is the heart of the book, has changed the least for the second edition. In 2014, Jeff asked Paul a very simple question: "Why are the four chapters in part 2 in that order?" The answer was stunned silence, as we realized that we had never asked which order they should appear in. It turns out that the RIPR model,

[1] To help reduce confusion, we developed the convention of using two digits for second edition chapters. Thus, in this preface, chapter 01 implies the second edition, whereas chapter 1 implies the first.

which is certainly central to software testing, dictates a logical order. Specifically, input space partitioning does not require reachability, infection, or propagation. Graph coverage criteria require execution to "get to" some location in the software artifact under test, that is, *reachability*, but not infection or propagation. Logic coverage criteria require that a predicate not only be reached, but be exercised in a particular way to affect the result of the predicate. That is, the predicate must be *infected*. Finally, syntax coverage not only requires that a location be reached, and that the program state of the "mutated" version be different from the original version, but that difference must be visible after execution finishes. That is, it must *propagate*. The second edition orders these four concepts based on the RIPR model, where each chapter now has successively stronger requirements. From a practical perspective, all we did was move the previous chapter 5 (now chapter 06) in front of the graph chapter (now chapter 07).

Another major structural change is that the second edition does **not** include chapters 7 through 9 from the first edition. The first edition material has become dated. Because it is used less than other material in the book, we decided not to delay this new edition of the book while we tried to find time to write this material. We plan to include better versions of these chapters in a third edition.

We also made hundreds of changes at a more detailed level. Recent research has found that in addition to an incorrect value propagating to the output, testing only succeeds if our automated test oracle looks at the right part of the software output. That is, the test oracle must *reveal* the failure. Thus, the old RIP model is now the RIPR model. Several places in the book have discussions that go beyond or into more depth than is strictly needed. The second edition now includes "meta discussions," which are ancillary discussions that can be interesting or insightful to some students, but unnecessarily complicated for others.

The new chapter 06 now has a fully worked out example of deriving an input domain model from a widely used Java library interface (in section 06.4). Our students have found this helps them understand how to use the input space partitioning techniques. The first edition included a section on "Representing graphs algebraically." Although one of us found this material to be fun, we both found it hard to motivate and unlikely to be used in practice. It also has some subtle technical flaws. Thus, we removed this section from the second edition. The new chapter 08 (logic) has a significant structural modification. The DNF criteria (formerly in section 3.6) properly belong at the front of the chapter. Chapter 08 now starts with semantic logic criteria (ACC and ICC) in 08.1, then proceeds to syntactic logic criteria (DNF) in 08.2. The syntactic logic criteria have also changed. One was dropped (UTPC), and CUTPNFP has been joined by MUTP and MNFP. Together, these three criteria comprise MUMCUT.

Throughout the book (especially part 2), we have improved the examples, simplified definitions, and included more exercises. When the first edition was published we had a partial solution manual, which somehow took five years to complete. We are proud to say that we learned from that mistake: we made (and stuck by!) a rule that we couldn't add an exercise without also adding a solution. The reader might think of this rule as testing for exercises. We are glad to say that the second edition book website **debuts** with a complete solution manual.

The second edition also has many dozens of corrections (starting with the errata list from the first edition book website), but including many more that we found while preparing the second edition. The second edition also has a better index. We put together the index for the first edition in about a day, and it showed. This time we have been indexing as we write, and committed time near the end of the process to specifically focus on the index. For future book writers, indexing is hard work and not easy to turn over to a non-author!

What is still the same in the second edition?

The things that have stayed the same are those that were successful in the first edition. The overall observation that test criteria are based on only four types of structures is still the key organizing principle of the second edition. The second edition is also written from an engineering viewpoint, assuming that users of the book are engineers who want to produce the highest quality software with the lowest possible cost. The concepts are well grounded in theory, yet presented in a practical manner. That is, the book tries to make theory meet practice; the theory is sound according to the research literature, but we also show how the theory applies in practice.

The book is also written as a text book, with clear explanations, simple but illustrative examples, and lots of exercises suitable for in-class or out-of-class work. Each chapter ends with bibliographic notes so that beginning research students can proceed to learning the deeper ideas involved in software testing. The book website (https://cs.gmu.edu/~offutt/softwaretest/) is rich in materials with solution manuals, listings of all example programs in the text, high quality PowerPoint slides, and software to help students with graph coverage, logic coverage, and mutation analysis. Some explanatory videos are also available and we hope more will follow. The solution manual comes in two flavors. The student solution manual, with solutions to about half the exercises, is available to everyone. The instructor solution manual has solutions to all exercises and is only available to those who convince the authors that they are using a book to teach a course.

Using the book in the classroom

The book chapters are built in a modular, component-based manner. Most chapters are independent, and although they are presented in the order that we use them, inter-chapter dependencies are few and they could be used in almost any order. Our primary target courses at our university are a fourth-year course (SWE 437) and a first-year graduate course (SWE 637). Interested readers can search on those courses ("mason swe 437" or "mason swe 637") to see our schedules and how we use the book. Both courses are required; SWE 437 is required in the software engineering concentration in our Applied Computer Science major, and SWE 637 is required in our MS program in software engineering[2]. Chapters 01 and 03 can be used in an early course such as CS2 in two ways. First, to sensitize early students to

[2] Our MS program is practical in nature, not research-oriented. The majority of students are part-time students with five to ten years of experience in the software industry. SWE 637 begat this book when we realized Beizer's classic text [Beizer, 1990] was out of print.

the importance of software quality, and second to get them started with test automation (we use JUnit at Mason). A second-year course in testing could cover all of part 1, chapter 06 from part 2, and all or part of part 3. The other chapters in part 2 are probably more than what such students need, but input space partitioning is a very accessible introduction to structured, high-end testing. A common course in north American computer science programs is a third-year course on general software engineering. Part 1 would be very appropriate for such a course. In 2016 we are introducing an advanced graduate course on software testing, which will span cutting-edge knowledge and current research. This course will use some of part 3, the material that we are currently developing for part 4, and selected research papers.

Teaching software testing

Both authors have become students of teaching over the past decade. In the early 2000s, we ran fairly traditional classrooms. We lectured for most of the available class time, kept organized with extensive PowerPoint slides, required homework assignments to be completed individually, and gave challenging, high-pressure exams. The PowerPoint slides and exercises in the first edition were designed for this model.

However, our teaching has evolved. We replaced our midterm exam with weekly quizzes, given in the first 15 minutes of class. This distributed a large component of the grade through the semester, relieved much of the stress of midterms, encouraged the students to keep up on a weekly basis instead of cramming right before the exam, and helped us identify students who were succeeding or struggling early in the term.

After learning about the "flipped classroom" model, we experimented with recorded lectures, viewed online, followed by doing the "homework" assignments in class with us available for immediate help. We found this particularly helpful with the more mathematically sophisticated material such as logic coverage, and especially beneficial to struggling students. As the educational research evidence against the benefits of lectures has mounted, we have been moving away from the "sage on a stage" model of talking for two hours straight. We now often talk for 10 to 20 minutes, then give in-class exercises[3] where the students immediately try to solve problems or answer questions. We confess that this is difficult for us, because we love to talk! Or, instead of showing an example during our lecture, we introduce the example, let the students work the next step in small groups, and then share the results. Sometimes our solutions are better, sometimes theirs are better, and sometimes solutions differ in interesting ways that spur discussion.

There is no doubt that this approach to teaching takes time and cannot acccomodate all of the PowerPoint slides we have developed. We believe that although we *cover* less material, we *uncover* more, a perception consistent with how our students perform on our final exams.

Most of the in-class exercises are done in small groups. We also encourage students to work out-of-class assignments collaboratively. Not only does evidence show

[3] These in-class exercises are not yet a formal part of the book website. But we often draw them from regular exercises in the text. Interested readers can extract recent versions from our course web pages with a search engine.

that students learn more when they work collaboratively ("peer-learning"), they enjoy it more, and it matches the industrial reality. Very few software engineers work alone.

Of course, you can use this book in your class as you see fit. We offer these insights simply as examples for things that work for us. We summarize our current philosophy of teaching simply: *Less talking, more teaching*.

Acknowledgments

It is our pleasure to acknowledge by name the many contributers to this text. We begin with students at George Mason who provided excellent feedback on early draft chapters from the second edition: Firass Almiski, Natalia Anpilova, Khalid Bargqdle, Mathew Fadoul, Mark Feghali, Angelica Garcia, Mahmoud Hammad, Husam Hilal, Carolyn Koerner, Han-Tsung Liu, Charon Lu, Brian Mitchell, Tuan Nguyen, Bill Shelton, Dzung Tran, Dzung Tray, Sam Tryon, Jing Wu, Zhonghua Xi, and Chris Yeung.

We are particularly grateful to colleagues who used draft chapters of the second edition. These early adopters provided valuable feedback that was extremely helpful in making the final document *classroom-ready*. Thanks to: Moataz Ahmed, King Fahd University of Petroleum & Minerals; Jeff Carver, University of Alabama; Richard Carver, George Mason University; Jens Hannemann, Kentucky State University; Jane Hayes, University of Kentucky; Kathleen Keogh, Federation University Australia; Robyn Lutz, Iowa State University; Upsorn Praphamontripong, George Mason University; Alper Sen, Bogazici University; Marjan Sirjani, Reykjavik University; Mary Lou Soffa, University of Virginia; Katie Stolee, North Carolina State University; and Xiaohong Wang, Salisbury University.

Several colleagues provided exceptional feedback from the first edition: Andy Brooks, Mark Hampton, Jian Zhou, Jeff (Yu) Lei, and six anonymous reviewers contacted by our publisher. The following individuals corrected, and in some cases developed, exercise solutions: Sana'a Alshdefat, Yasmine Badr, Jim Bowring, Steven Dastvan, Justin Donnelly, Martin Gebert, JingJing Gu, Jane Hayes, Rama Kesavan, Ignacio Martín, Maricel Medina-Mora, Xin Meng, Beth Paredes, Matt Rutherford, Farida Sabry, Aya Salah, Hooman Safaee, Preetham Vemasani, and Greg Williams. The following George Mason students found, and often corrected, errors in the first edition: Arif Al-Mashhadani, Yousuf Ashparie, Parag Bhagwat, Firdu Bati, Andrew Hollingsworth, Gary Kaminski, Rama Kesavan, Steve Kinder, John Krause, Jae Hyuk Kwak, Nan Li, Mohita Mathur, Maricel Medina Mora, Upsorn Praphamontripong, Rowland Pitts, Mark Pumphrey, Mark Shapiro, Bill Shelton, David Sracic, Jose Torres, Preetham Vemasani, Shuang Wang, Lance Witkowski, Leonard S. Woody III, and Yanyan Zhu. The following individuals from elsewhere found, and often corrected, errors in the first edition: Sana'a Alshdefat, Alexandre Bartel, Don Braffitt, Andrew Brooks, Josh Dehlinger, Gordon Fraser, Rob Fredericks, Weiyi Li, Hassan Mirian, Alan Moraes, Miika Nurminen, Thomas Reinbacher, Hooman Rafat Safaee, Hossein Saiedian, Aya Salah, and Markku Sakkinen. Lian Yu of Peking University translated the the first edition into Mandarin Chinese.

We also want to acknowledge those who implicitly contributed to the second edition by explicitly contributing to the first edition: Aynur Abdurazik, Muhammad Abdulla, Roger Alexander, Lionel Briand, Renee Bryce, George P. Burdell, Guillermo Calderon-Meza, Jyothi Chinman, Yuquin Ding, Blaine Donley, Patrick Emery, Brian Geary, Hassan Gomaa, Mats Grindal, Becky Hartley, Jane Hayes, Mark Hinkle, Justin Hollingsworth, Hong Huang, Gary Kaminski, John King, Yuelan Li, Ling Liu, Xiaojuan Liu, Chris Magrin, Darko Marinov, Robert Nilsson, Andrew J. Offutt, Buzz Pioso, Jyothi Reddy, Arthur Reyes, Raimi Rufai, Bo Sanden, Jeremy Schneider, Bill Shelton, Michael Shin, Frank Shukis, Greg Williams, Quansheng Xiao, Tao Xie, Wuzhi Xu, and Linzhen Xue.

While developing the second edition, our graduate teaching assistants at George Mason gave us fantastic feedback on early drafts of chapters: Lin Deng, Jingjing Gu, Nan Li, and Upsorn Praphamontripong. In particular, Nan Li and Lin Deng were instrumental in completing, evolving, and maintaining the software coverage tools available on the book website.

We are grateful to our editor, Lauren Cowles, for providing unwavering support and enforcing the occasional deadline to move the project along, as well as Heather Bergmann, our former editor, for her strong support on this long-running project.

Finally, of course none of this is possible without the support of our families. Thanks to Becky, Jian, Steffi, Matt, Joyce, and Andrew for helping us stay balanced.

Just as all programs contain faults, all texts contain errors. Our text is no different. And, as responsibility for software faults rests with the developers, responsibility for errors in this text rests with us, the authors. In particular, the bibliographic notes sections reflect our perspective of the testing field, a body of work we readily acknowledge as large and complex. We apologize in advance for omissions, and invite pointers to relevant citations.