Introduction to Software Testing

Extensively class tested, this text takes an innovative approach to software testing: it defines testing as the process of applying a few well-defined, general-purpose test criteria to a structure or model of the software. The structure of the text directly reflects the pedagogical approach and incorporates the latest innovations in testing, including modern types of software such as OO, Web applications, and embedded software. The book contains numerous examples throughout. An instructor's solution manual, PowerPoint slides, sample syllabi, additional examples and updates, testing tools for students, and example software programs in Java are available on an extensive Web site at www.introsoftwaretesting.com.

Paul Ammann, PhD, is an Associate Professor of software engineering at George Mason University. He received an outstanding teaching award in 2007 from the Volgenau School of Information Technology and Engineering. Dr. Ammann earned an AB degree in computer science from Dartmouth College and MS and PhD degrees in computer science from the University of Virginia.

Jeff Offutt, PhD, is a Professor of software engineering at George Mason University. He is editor-in-chief of the *Journal of Software Testing, Verification and Reliability*; chair of the steering committee for the IEEE International Conference on Software Testing, Verification, and Validation; and on the editorial boards for several journals. He recived the outstanding teacher award from the Volgenau School of Information Technology and Engineering in 2003. Dr. Offutt earned a BS degree in mathematics and data processing from Morehead State University and MS and PhD degrees in computer science from the Georgia Institute of Technology.

INTRODUCTION TO SOFTWARE TESTING

Paul Ammann
George Mason University

Jeff Offutt
George Mason University

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo, Delhi

Cambridge University Press
32 Avenue of the Americas, New York, NY 10013-2473, USA

www.cambridge.org
Information on this title: www.cambridge.org/9780521880381

First published 2008

Printed in the United States of America

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

Library of Congress Cataloging in Publication Data

Ammann, Paul, 1961–
Introduction to software testing / Paul Ammann, Jeff Offutt.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-521-88038-1 (hardback)
1. Computer software – Testing. I. Offutt, Jeff, 1961– II. Title.
QA76.76.T48A56 2008
004.2′4–dc22 2007035077

ISBN 978-0-521-88038-1 hardback

Contents

List of Figures

List of Tables

Preface

This book presents software testing as a practical engineering activity, essential to producing high-quality software. It is designed to be used as the primary textbook in either an undergraduate or graduate course on software testing, as a supplement to a general course on software engineering or data structures, and as a resource for software test engineers and developers. This book has a number of unique features:

- It organizes the complex and confusing landscape of test coverage criteria with a novel and extremely simple structure. At a technical level, software testing is based on satisfying coverage criteria. The book's central observation is that there are few truly different coverage criteria, each of which fits easily into one of four categories: graphs, logical expressions, input space, and syntax structures. This not only simplifies testing, but it also allows a convenient and direct theoretical treatment of each category. This approach contrasts strongly with the traditional view of testing, which treats testing at each phase in the development process differently.
- It is designed and written to be a textbook. The writing style is direct, it builds the concepts from the ground up with a minimum of required background, and it includes lots of examples, homework problems, and teaching materials. It provides a balance of theory and practical application, presenting testing as a collection of objective, quantitative activities that can be measured and repeated. The theoretical concepts are presented when needed to support the practical activities that test engineers follow.
- It assumes that testing is part of a mental discipline that helps all IT professionals develop higher-quality software. Testing is not an anti-engineering activity, and it is not an inherently destructive process. Neither is it only for testing specialists or domain experts who know little about programming or math.
- It is designed with modular, interconnecting pieces; thus it can be used in multiple courses. Most of the book requires only basic discrete math and introductory programming, and the parts that need more background are clearly marked. By

using the appropriate sections, this book can support several classes, as described later in the preface.

- It assumes the reader is learning to be an engineer whose goal is to produce the best possible software with the lowest possible cost. The concepts in this book are well grounded in theory, are practical, and most are currently in use.

WHY SHOULD THIS BOOK BE USED?

Not very long ago, software development companies could afford to employ programmers who could not test and testers who could not program. For most of the industry, it was not necessary for either group to know the technical principles behind software testing or even software development. Software testing in industry historically has been a nontechnical activity. Industry viewed testing primarily from the managerial and process perspective and had limited expectations of practitioners' technical training.

As the software engineering profession matures, and as software becomes more pervasive in everyday life, there are increasingly stringent requirements for software reliability, maintainability, and security. Industry must respond to these changes by, among other things, improving the way software is tested. This requires increased technical expertise on the part of test engineers, as well as increased emphasis on testing by software developers. The good news is that the knowledge and technology are available and based on over 30 years of research and practice. This book puts that knowledge into a form that students, test engineers, test managers, and developers can access.

At the same time, it is relatively rare to find courses that teach testing in universities. Only a few undergraduate courses exist, almost no masters degree programs in computer science or software engineering require a course in software testing, and only a few dozen have an elective course. Not only is testing not covered as an essential part of undergraduate computer science education, most computer science students either never gain any knowledge about testing, or see only a few lectures as part of a general course in software engineering.

The authors of this book have been teaching software testing to software engineering and computer science students for more than 15 years. Over that time we somewhat reluctantly came to the conclusion that no one was going to write the book we wanted to use. Rather, to get the book we wanted, we would have to write it.

Previous testing books have presented software testing as a relatively simple subject that relies more on process than technical understanding of how software is constructed, as a complicated and fractured subject that requires detailed understanding of numerous software development technologies, or as a completely theoretical subject that can be mastered only by mathematicians and theoretical computer scientists. Most books on software testing are organized around the phases in a typical software development lifecycle, an approach that has the unfortunate side effect of obscuring common testing themes. Finally, most testing books are written as reference books, not textbooks. As a result, only instructors with prior expertise in software testing can easily teach the subject. **This book is accessible to instructors who are not already testing experts**.

This book differs from other books on software testing in other important ways. Many books address managing the testing process. While this is important, it is equally important to give testers specific techniques grounded in basic theory. This book provides a balance of theory and practical application. This is important information that software companies must have; however, this book focuses specifically on the technical nuts-and-bolts issues of designing and creating tests. Other testing books currently on the market focus on specific techniques or activities, such as system testing or unit testing. This book is intended to be comprehensive over the entire software development process and to cover as many techniques as possible.

As stated previously, the motivation for this book is to support courses in software testing. Our first target was our own software testing course in our Software Engineering MS program at George Mason University. This popular elective is taught to about 30 computer science and software engineering students every semester. We also teach PhD seminars in software testing, industry short courses on specialized aspects, and lectures on software testing in various undergraduate courses. Although few undergraduate courses on software testing exist, we believe that they should exist, and we expect they will in the near future. Most testing books are not designed for classroom use. We specifically wrote this book to support our classroom activities, and it is no accident that the syllabus for our testing course, available on the book's Web site (www.introsoftwaretesting.com), closely follows the table of contents for this book.

This book includes numerous carefully worked examples to help students and teachers alike learn the sometimes complicated concepts. The instructor's resources include high-quality powerpoint slides, presentation hints, solutions to exercises, and working software. Our philosophy is that we are doing more than writing a book; we are offering our course to the community. One of our goals was to write material that is scholarly and true to the published research literature, but that is also accessible to nonresearchers. Although the presentation in the book is quite a bit different from the research papers that the material is derived from, the essential ideas are true to the literature. To make the text flow more smoothly, we have removed the references from the presentation. For those interested in the research genealogy, each chapter closes with a bibliographic notes section that summarizes where the concepts come from.

WHO SHOULD READ THIS BOOK?

Students who read and use this book will learn the fundamental principles behind software testing, and how to apply these principles to produce better software, faster. They will not only become better programmers, they will also be prepared to carry out high-quality testing activities for their future employers. *Instructors* will be able to use this book in the classroom, even without prior practical expertise in software testing. The numerous exercises and thought-provoking problems, classroom-ready and classroom-tested slides, and suggested outside activities make this material teachable by instructors who are not already experts in software testing. *Research students* such as beginning PhD students will find this book to be an invaluable resource as a starting point to the field. The theory is sound and clearly

presented, the practical applications reveal what is useful and what is not, and the advanced reading and bibliographic notes provide pointers into the literature. Although the set of research students in software testing is a relatively small audience, we believe it is a key audience, because a common, easily achievable baseline would reduce the effort required for research students to join the community of testing researchers. *Researchers* who are already familiar with the field will find the criteria-approach to be novel and interesting. Some may disagree with the pedagogical approach, but we have found that the view that testing is an application of only a few criteria to a very few software structures to be very helpful to our research. We hope that testing research in the future will draw away from searches for more criteria to novel uses and evaluations of existing criteria.

Testers in the industry will find this book to be an invaluable collection of techniques that will help improve their testing, no matter what their current process is. The criteria presented here are intended to be used as a "toolbox" of tricks that can be used to find faults. *Developers* who read this book will find numerous ways to improve their own software. Their self-testing activities can become more efficient and effective, and the discussions of software faults that test engineers search for will help developers avoid them. To paraphrase a famous parable, if you want to teach a person to be a better fisherman, explain how and where the fish swim. Finally, *managers* will find this book to be a useful explanation of how clever test engineers do their job, and of how test tools work. They will be able to make more effective decisions regarding hiring, promotions, and purchasing tools.

HOW CAN THIS BOOK BE USED?

A major advantage of the structure of this book is that it can be easily used for several different courses. Most of the book depends on material that is taught very early in college and some high schools: basic concepts from data structures and discrete math. The sections are organized so that the early material in each chapter is accessible to less advanced students, and material that requires more advanced knowledge is clearly marked.

Specifically, the book defines six separate sets of chapter sections that form *streams* through the book:

1. A module within a CS II course
2. A sophomore-level course on software testing
3. A module in a general software engineering course
4. A senior-level course on software testing
5. A first-year MS level course on software testing
6. An advanced graduate research-oriented course on software testing
7. Industry practioner relevant sections

The stream approach is illustrated in the abbreviated table of contents in the figure shown on pp. xix–xx. Each chapter section is marked with which stream it belongs too. Of course, individual instructors, students, and readers may prefer to adapt the stream to their own interests or purposes. We suggest that the first two sections of Chapter 1 and the first two sections of Chapter 6 are appropriate reading for a module in a data structures (CS II) class, to be followed by a simple

Stream 1: Module in a CS II course.

Stream 2: Sophomore-level course on software testing.

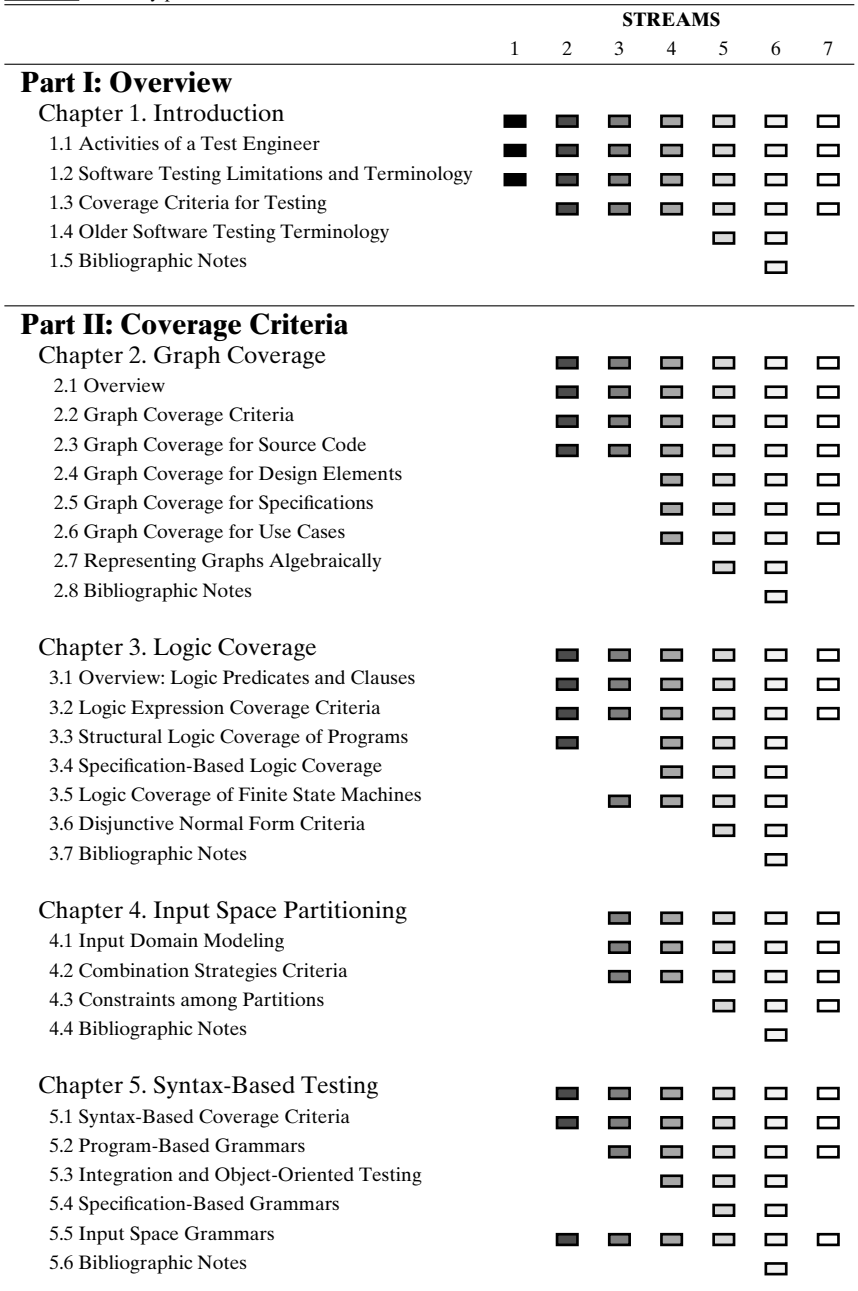
Stream 3: Module in a general software engineering course.

Stream 4: Senior-level course on software testing.

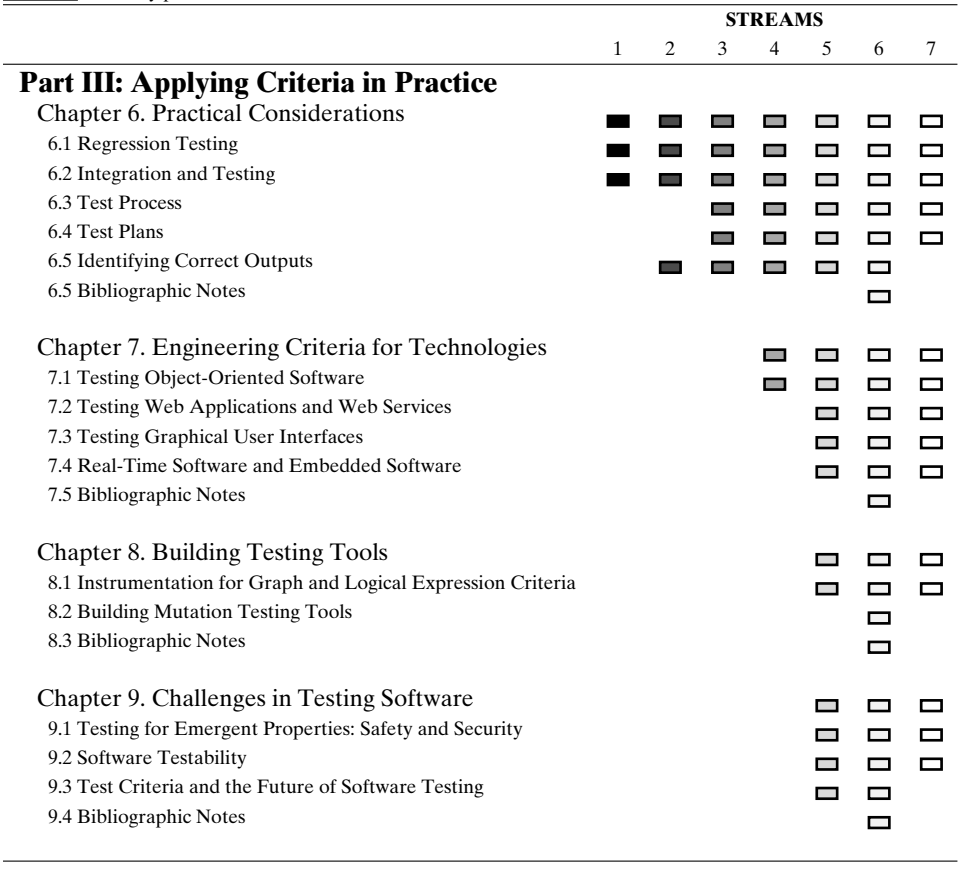
Stream 5: First-year MS course on software testing.

Stream 6: Advanced graduate research-oriented course on software testing.

Stream 7: Industry practitioner relevant sections



Stream 1: Module in a CS II course.
Stream 2: Sophomore-level course on software testing.
Stream 3: Module in a general software engineering course.
Stream 4: Senior-level course on software testing.
Stream 5: First-year MS course on software testing.
Stream 6: Advanced graduate research-oriented course on software testing.
Stream 7: Industry practitioner relevant sections



assignment. Our favorite is to ask the students to retrieve one of their previously graded programs and satisfy some simple test criterion like branch coverage. We offer points for every fault found, driving home two concepts: an "A" grade doesn't mean the program always works, and finding faults is a good thing.

The sophomore-level course on software testing (stream 2) is designed to immediately follow a data structures course (CS II). The marked sections contain material that depends only on data structures and discrete math.

A module in a general software engineering course (stream 3) could augment the survey material typical in such courses. The sections marked provide basic literacy in software testing.

The senior-level course on software testing (stream 4) is the primary target for this text. It adds material that requires a little more sophistication in terms of

software development than the sophomore stream. This includes sections in Chapter 2 on data flow testing, sections that involve integration testing of multiple modules, and sections that rely on grammars or finite state machines. Most senior computer science students will have seen this material in their other courses. Most of the sections that appear in stream 4 but not stream 2 could be added to stream 2 with appropriate short introductions. It is important to note that a test engineer does not need to know all the theory of parsing to use data flow testing or all the theory on finite state machines to use statecharts for testing.

The graduate-level course on software testing (stream 5) adds some additional sections that rely on a broader context and that require more theoretical maturity. For example, these sections use knowledge of elementary formal methods, polymorphism, and some of the UML diagrams. Some of the more advanced topics and the entire chapter on building testing tools are also intended for a graduate audience. This chapter could form the basis for a good project, for example, to implement a simple coverage analyzer.

An advanced graduate course in software testing with a research emphasis such as a PhD seminar (stream 6) includes issues that are still unproven and research in nature. The bibliographic notes are recommended only for these students as indicators for future in-depth reading.

Finally, sections that are reasonably widely used in industry, especially those that have commercial tool support, are marked for stream 7. These sections have a minimum of theory and omit criteria that are still of questionable usefulness.

Extensive supplementary materials, including sample syllabuses, PowerPoint slides, presentation hints, solutions to exercises, working software, and errata are available on the book's companion Web site.

ACKNOWLEDGMENTS

Many people helped us write this book. Not only have the students in our Software Testing classes at George Mason been remarkably tolerant of using a work in progress, they have enthusiastically provided feedback on how to improve the text. We cannot acknowledge all by name (ten semesters worth of students have used it!), but the following have made especially large contributions: Aynur Abdurazik, Muhammad Abdulla, Yuquin Ding, Jyothi Chinman, Blaine Donley, Patrick Emery, Brian Geary, Mark Hinkle, Justin Hollingsworth, John King, Yuelan Li, Xiaojuan Liu, Chris Magrin, Jyothi Reddy, Raimi Rufai, Jeremy Schneider, Bill Shelton, Frank Shukis, Quansheng Xiao, and Linzhen Xue. We especially appreciate those who generously provided extensive comments on the entire book: Guillermo Calderon-Meza, Becky Hartley, Gary Kaminski, and Andrew J. Offutt. We gratefully acknowledge the feedback of early adopters at other educational institutions: Roger Alexander, Jane Hayes, Ling Liu, Darko Marinov, Arthur Reyes, Michael Shin, and Tao Xie. We also want to acknowledge several people who provided material for the book: Roger Alexander, Mats Grindal, Hong Huang, Gary Kaminski, Robert Nilsson, Greg Williams, Wuzhi Xu. We were lucky to receive excellent suggestion from Lionel Briand, Renée Bryce, Kim King, Sharon Ritchey, Bo Sanden, and Steve Schach. We are grateful to our editor, Heather Bergman,

for providing unwavering support and enforcing the occasional deadline to move the project along, as well as Kerry Cahill from Cambridge University Press for very strong support on this project.

We also acknowledge George Mason University for supporting both of us on sabbaticals and for providing GTA support at crucial times. Our department Chair, Hassan Gomaa, has enthusiastically supported this effort.

Finally, of course none of this is possible without the support of our families. Thanks to Becky, Jian, Steffi, Matt, Joyce, and Andrew for keeping us grounded in reality and helping keep us happy for the past five years.

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.

Paul Ammann
Jeff Offutt