Quantitative Software Engineering Series

Lawrence Bernstein, Series Editor

SOFTWARE TESTING

Concepts and Operations

Ali Mili • Fairouz Tchier

WILEY

Software Testing

Concepts and Operations

Ali Mili
NJIT, USA

Fairouz Tchier
KSU, KSA

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data

Mili, Ali.
Software testing : concepts and operations / Ali Mili.
pages cm
Includes bibliographical references and index.
ISBN 978-1-118-66287-8 (cloth)
1. Computer software–Testing. I. Title.
QA76.76.T48M56 2015
005.1′4–dc23
2015001931

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Software Testing

Quantitative Software Engineering Series

The Quantitative Engineering Series focuses on the convergence of systems engineering with emphasis on quantitative engineering trade-off analysis. Each title brings the principles and theory of programming in-the-large and industrial strength software into focus.

This practical series helps software developers, software engineers, systems engineers, and graduate students understand and benefit from this convergence through the unique weaving of software engineering case histories, quantitative analysis, and technology into the project effort. You will find each publication reinforces the series goal of assisting the reader with producing useful, well-engineered software systems.

Series Editor: **Lawrence Bernstein**

Late Professor Bernstein was Industrial Research Professor at the Stevens Institute of Technology. He previously pursued a distinguished executive career at Bell Laboratories. He was a fellow of the IEEE and ACM.

Trustworthy Systems for Quantitative Software Engineering / Larry Bernstein and C.M. Yuhas

Software Measurement and Estimation: A Practical Approach / Linda M. Laird and M. Carol Brennan

World Wide Web Application Engineering and Implementation / Steven A. Gabarro

Software Performance and Scalability / Henry H. Liu

Managing the Development of Software-Intensive Systems / James McDonald

Trustworthy Compilers / Vladimir O. Safonov

Oracle Database Performance and Scalability: A Quantitative Approach / Henry H. Liu

Enterprise Software Architecture and Design: Entities, Services and Resources / Dominic Duggan

Software Testing: Concepts and Operations / Ali Mili and Fairouz Tchier

Dedicated to my parents
in honor of their 68 years of mutual devotion
and to Amel, Noor, Farah Aisha, and Serena Aida.
May they realize their hopes and dreams.
A.M.

Dedicated to my loving parents,
my husband Jamel, and my children Sarah, Bellal, and Amine.
May their lives be filled with happiness and success.
F.T.

Preface

Software engineering is the only engineering discipline where product testing is a major technical and organizational concern, as well as an important cost factor. Several factors contribute to this state of affairs:

- The first factor that makes software testing such a big concern is, of course, the size and complexity of software products, which make the design of software products a high-risk, error-prone proposition.
- The second factor is the lack of a standardized development process for software products, which means that product quality cannot be ensured by process controls, and has to be ensured by product controls instead.
- The third factor is the scarcity of practical, scalable methods that can ensure product quality through static product analysis, shifting the burden to dynamic methods.
- Other factors include the absence of a general reuse discipline, the lack of scalability of correctness-preserving development methods, and the pervasiveness of specification changes through the development, maintenance, and evolution process, etc.

The subject of this book is the study of software testing; amongst the many books that are currently available on the same subject, this book can be characterized by the following premises:

- *Software testing as an integral part of software quality assurance.* We view software testing as part of a comprehensive strategy for software quality assurance, alongside many other techniques. The law of diminishing returns advocates the

use of a variety of diverse techniques, which complement each other, in such a way that each is used wherever it delivers the greatest return on investment. Hence software testing is better studied in a broader context that also encompasses other methods rather than to be studied as an isolated set of techniques.

- *Software testing as a complementary technique to static analysis.* Since the early days of software engineering, we have witnessed a colorful debate on the respective merits of software testing versus static program analysis in terms of effectiveness, scalability, ease of use, etc. We take the position that each of these techniques is effective against some type of specifications and less effective against other types; also, very often, when we find that one technique or another is difficult to use, it is not the result of any intrinsic shortcoming of the technique, rather it is because the technique is used against the wrong kind of specification. Of course, we do not always get to choose the specification against which we must ensure product correctness; but we can, in fact, decompose a complex specification into components and map each component to the technique that is most adapted to it. This is illustrated in Chapter 6.
- *Software testing as a systematic stepwise process.* Early on, software testing earned the reputation of being a means to prove the presence of faults in programs, but never their absence; this is an undeserved reputation, in fact, because testing can be used for all sorts of goals, as we discuss in Chapter 7. Nevertheless, whether deserved or not, this reputation has had two consequences: first, the assumption that the only possible goal of testing is fault exposure, diagnosis, and removal. Second, the (consequent) belief that testing amounts merely to test data generation, specifically the generation of test data that has the greatest potential to expose faults. By contrast, we argue that testing follows a multiphase process that includes goal identification and analysis, test data generation, oracle design, test driver design, test deployment, and test outcome analysis. We devote different chapters to each one of these phases.
- *Software testing as a formal/formalizable process.* Because it requires relatively little analysis of the software product under test or its specification, testing is often perceived as an activity that can be carried out casually, and informally. By contrast, we argue that testing ought to be carried out with the same level of rigor as static program verification, and that to perform testing effectively, one must be knowledgeable in software specifications, in program correctness, in relative correctness, in the meaning of a fault, in fault removal, etc.. This is discussed in more detail in Chapter 6.
- *Software testing as a goal-oriented activity.* We argue that far from being solely dedicated to finding and removing faults, testing may have a wide range of goals, including such goals as estimating fault density, estimating reliability, certifying reliability, etc. This is discussed in detail in Chapter 7.

This book stems from lecture notes of a course on software testing and quality assurance and hence is primarily intended for classroom use; though it may also be of interest to practicing software engineers, as well as to researchers in software

engineering. The book is divided into five broad parts, including 3 or 4 chapters per part, to a total of 16 chapters.

- Part I introduces software testing in the broader context of software engineering and explores the qualities that testing aims to achieve or ascertain, as well as the lifecycle of software testing.
- Part II introduces mathematical foundations of software testing, which include software specification, program correctness and verification, concepts of software dependability, and a software testing taxonomy. It is uncommon for a software testing book to discuss specifications, verification, and dependability to the extent that we do in this book. We do it in this book for many reasons:
  - First, we believe that it is not possible to study software testing without a sound understanding of software specifications, since these capture the functional attributes that are testing candidate programs against and are the basis for oracle design.
  - Second, when we test a program in the context of product certification or in the context of acceptance testing, what is at stake is whether the candidate program is correct; surely, we need to understand what correctness means, for this purpose.
  - Third, if dynamic program testing and static program analysis are to be used in concert, to reach a more complete conclusion than any one method alone, they need to be cast in the same mathematical model.
  - Fourth, the act of removing a fault from a program, which is so central to testing, can only be modeled by defining the property of *relative correctness*, which provides that the program is *more-correct* once the fault is removed than it was prior to fault removal; relative correctness, in turn, can only be defined and understood if we understand the property of (absolute) correctness.

  The taxonomy of software testing techniques classify these techniques according to a number of criteria, including in particular the criterion of goals: It is important to recognize the different goals that one may pursue in conducting software testing, and how each goal affects all the phases of the testing lifecycle, from test data generation to oracle design to test deployment to test outcome analysis.
- Part III explores a phase of software testing that has so dominated the attention of researchers and practitioners that it is often viewed as the only worthwhile issue in software testing: test data generation. In this part, we briefly discuss some general concepts of test data generation and then we explore the two broad criteria of test data generation, namely: functional criteria (Chapter 9) and structural criteria (Chapter 10). We discuss test data generation for simple programs that map initial states onto final states, as well as for state-bearing programs, whose output depends on their input history.
- Part IV discusses the remaining phases of the software testing lifecycle that arise after test data generation and include test oracle design, test driver design, and

test outcome analysis. Test oracles (Chapter 11) are derived from target specifications according to the definition of correctness and depend on whether we are talking about simple state-free programs or about programs that have an internal state. Test driver design (Chapter 12) depends on whether test data has been generated offline and is merely deployed from an existing medium, or whether it is generated at random according to some probability law. As for the analysis of test outcomes (Chapter 13), it depends of course on the goal of the test and ranges from reliability estimation to reliability certification to fault density estimation to product acceptance, etc.

- Part V concludes the book by surveying some managerial aspects of software testing, including software metrics (Chapter 14), software testing tools (Chapter 15), and software product line testing (Chapter 16).

In compiling the material of this book, we focused our attention on analyzing and modeling important aspects of software testing, rather than on surveying and synthesizing the latest research on the topic; several premises determined this decision:

- This book is primarily intended to be an educational tool rather than a research monograph.
- In an area of active research such as software testing, students are better served by focusing on fundamental concepts that will serve them in the long run regardless of what problem they may encounter rather than to focus on the *latest techniques*, which by definition will not remain *latest* for too long.

In the perennial academic debate of whether we serve our students best by making them operational in the short term or by presenting them with fundamentals and enabling them to adapt in the long run, we have decided to err on the side of the latter option.

ACKNOWLEDGMENT

Special thanks are due to the late Professor Lawrence Bernstein for inviting us to write this book for inclusion in his distinguished series.

We thank our successive cohorts of students, who tolerated our caprices as we fine-tuned and refined the contents of our lecture notes term after term. We also thank Slim Frikha, a summer intern from ParisTech, France, who reviewed and evaluated software testing tools to help us with Chapter 15. This publication was made possible in part by a grant from the Qatar National Research Fund NPRP 04-1109-1-174. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the QNRF.

F. Tchier
A. Mili

Contents