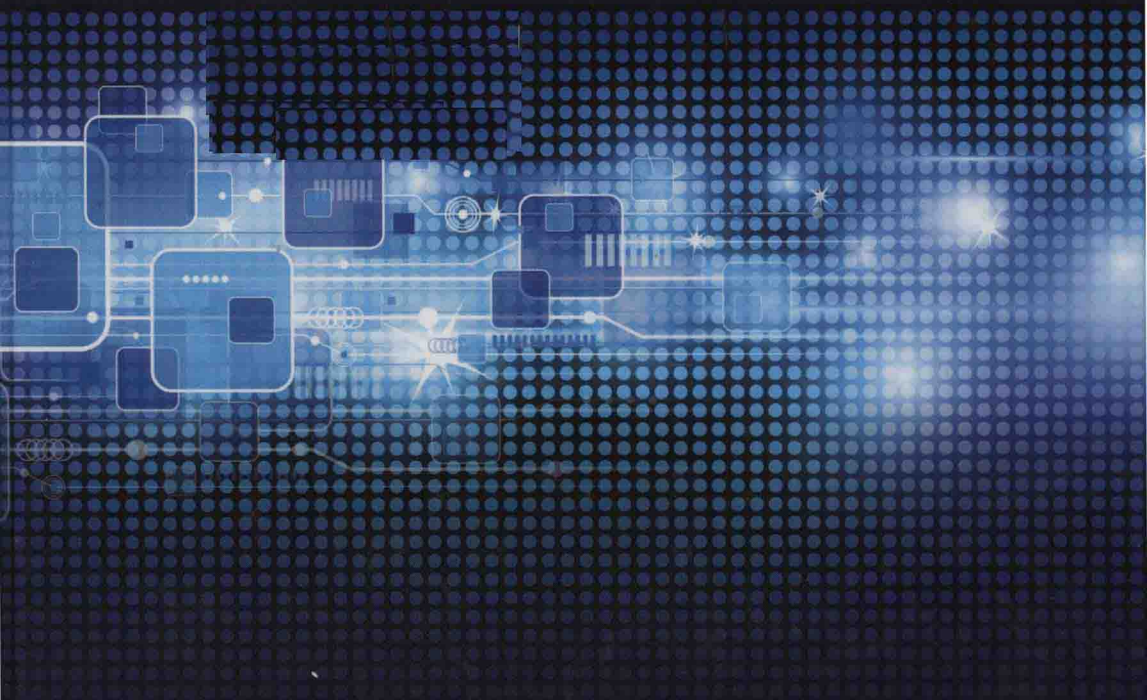


Quantitative Software Engineering Series

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

Copyright © 2015 by John Wiley & Sons, Inc. All rights reserved

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

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

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

PREFACE	xiv
PART I INTRODUCTION TO SOFTWARE TESTING	1
1 Software Engineering: A Discipline Like No Other	3
1.1 A Young, Restless Discipline / 3	
1.2 An Industry Under Stress / 5	
1.3 Large, Complex Products / 5	
1.4 Expensive Products / 7	
1.5 Absence of Reuse Practice / 9	
1.6 Fault-Prone Designs / 9	
1.7 Paradoxical Economics / 10	
1.7.1 A Labor-Intensive Industry / 10	
1.7.2 Absence of Automation / 11	
1.7.3 Limited Quality Control / 11	
1.7.4 Unbalanced Lifecycle Costs / 12	
1.7.5 Unbalanced Maintenance Costs / 12	
1.8 Chapter Summary / 13	
1.9 Bibliographic Notes / 13	

2	Software Quality Attributes	14
2.1	Functional Attributes / 15	
2.1.1	Boolean Attributes / 15	
2.1.2	Statistical Attributes / 15	
2.2	Operational Attributes / 17	
2.3	Usability Attributes / 18	
2.4	Business Attributes / 19	
2.5	Structural Attributes / 20	
2.6	Chapter Summary / 21	
2.7	Exercises / 21	
2.8	Bibliographic Notes / 22	
3	A Software Testing Lifecycle	23
3.1	A Software Engineering Lifecycle / 23	
3.2	A Software Testing Lifecycle / 27	
3.3	The V-Model of Software Testing / 32	
3.4	Chapter Summary / 33	
3.5	Bibliographic Notes / 34	
PART II	FOUNDATIONS OF SOFTWARE TESTING	35
4	Software Specifications	37
4.1	Principles of Sound Specification / 38	
4.1.1	A Discipline of Specification / 38	
4.2	Relational Mathematics / 39	
4.2.1	Sets and Relations / 39	
4.2.2	Operations on Relations / 39	
4.2.3	Properties of Relations / 41	
4.3	Simple Input Output Programs / 42	
4.3.1	Representing Specifications / 42	
4.3.2	Ordering Specifications / 46	
4.3.3	Specification Generation / 48	
4.3.4	Specification Validation / 53	
4.4	Reliability Versus Safety / 60	
4.5	State-based Systems / 61	
4.5.1	A Relational Model / 62	
4.5.2	Axiomatic Representation / 64	
4.5.3	Specification Validation / 70	
4.6	Chapter Summary / 72	
4.7	Exercises / 72	
4.8	Problems / 76	
4.9	Bibliographic Notes / 78	

5	Program Correctness and Verification	79
5.1	Correctness: A Definition / 80	
5.2	Correctness: Propositions / 83	
5.2.1	Correctness and Refinement / 83	
5.2.2	Set Theoretic Characterizations / 85	
5.2.3	Illustrations / 86	
5.3	Verification / 88	
5.3.1	Sample Formulas / 89	
5.3.2	An Inference System / 91	
5.3.3	Illustrative Examples / 94	
5.4	Chapter Summary / 98	
5.5	Exercises / 99	
5.6	Problems / 100	
5.7	Bibliographic Notes / 100	
6	Failures, Errors, and Faults	101
6.1	Failure, Error, and Fault / 101	
6.2	Faults and Relative Correctness / 103	
6.2.1	Fault, an Evasive Concept / 103	
6.2.2	Relative Correctness / 104	
6.3	Contingent Faults and Definite Faults / 107	
6.3.1	Contingent Faults / 107	
6.3.2	Monotonic Fault Removal / 109	
6.3.3	A Framework for Monotonic Fault Removal / 114	
6.3.4	Definite Faults / 114	
6.4	Fault Management / 116	
6.4.1	Lines of Defense / 116	
6.4.2	Hybrid Validation / 118	
6.5	Chapter Summary / 121	
6.6	Exercises / 122	
6.7	Problems / 123	
6.8	Bibliographic Notes / 124	
7	A Software Testing Taxonomy	125
7.1	The Trouble with Hyphenated Testing / 125	
7.2	A Classification Scheme / 126	
7.2.1	Primary Attributes / 127	
7.2.2	Secondary Attributes / 131	
7.3	Testing Taxonomy / 136	
7.3.1	Unit-Level Testing / 136	
7.3.2	System-Level Testing / 138	
7.4	Exercises / 139	
7.5	Bibliographic Notes / 140	

PART III TEST DATA GENERATION	141
8 Test Generation Concepts	143
8.1 Test Generation and Target Attributes / 143	
8.2 Test Outcomes / 146	
8.3 Test Generation Requirements / 148	
8.4 Test Generation Criteria / 152	
8.5 Empirical Adequacy Assessment / 155	
8.6 Chapter Summary / 160	
8.7 Exercises / 161	
8.8 Bibliographic Notes / 162	
8.9 Appendix: Mutation Program / 163	
9 Functional Criteria	165
9.1 Domain Partitioning / 165	
9.2 Test Data Generation from Tabular Expressions / 171	
9.3 Test Generation for State Based Systems / 176	
9.4 Random Test Data Generation / 184	
9.5 Tourism as a Metaphor for Test Data Selection / 188	
9.6 Chapter Summary / 190	
9.7 Exercises / 190	
9.8 Bibliographic Notes / 192	
10 Structural Criteria	193
10.1 Paths and Path Conditions / 194	
10.1.1 Execution Paths / 194	
10.1.2 Path Functions / 196	
10.1.3 Path Conditions / 201	
10.2 Control Flow Coverage / 202	
10.2.1 Statement Coverage / 202	
10.2.2 Branch Coverage / 204	
10.2.3 Condition Coverage / 207	
10.2.4 Path Coverage / 209	
10.3 Data Flow Coverage / 214	
10.3.1 Definitions and Uses / 214	
10.3.2 Test Generation Criteria / 217	
10.3.3 A Hierarchy of Criteria / 220	
10.4 Fault-Based Test Generation / 220	
10.4.1 Sensitizing Faults / 221	
10.4.2 Selecting Input Data for Fault Sensitization / 225	
10.4.3 Selecting Input Data for Error Propagation / 227	
10.5 Chapter Summary / 228	

- 10.6 Exercises / 229
- 10.7 Bibliographic Notes / 232

PART IV TEST DEPLOYMENT AND ANALYSIS **233**

11 Test Oracle Design **235**

- 11.1 Dilemmas of Oracle Design / 235
- 11.2 From Specifications to Oracles / 238
- 11.3 Oracles for State-Based Products / 242
 - 11.3.1 From Axioms to Oracles / 243
 - 11.3.2 From Rules to Oracles / 244
- 11.4 Chapter Summary / 250
- 11.5 Exercises / 251

12 Test Driver Design **253**

- 12.1 Selecting a Specification / 253
- 12.2 Selecting a Process / 255
- 12.3 Selecting a Specification Model / 257
 - 12.3.1 Random Test Generation / 257
 - 12.3.2 Pre-Generated Test Data / 263
 - 12.3.3 Faults and Fault Detection / 266
- 12.4 Testing by Symbolic Execution / 269
- 12.5 Chapter Summary / 274
- 12.6 Exercises / 275
- 12.7 Bibliographic Notes / 279

13 Test Outcome Analysis **280**

- 13.1 Logical Claims / 281
 - 13.1.1 Concrete Testing / 281
 - 13.1.2 Symbolic Testing / 282
 - 13.1.3 Concolic Testing / 283
- 13.2 Stochastic Claims: Fault Density / 284
- 13.3 Stochastic Claims: Failure Probability / 287
 - 13.3.1 Faults Are Not Created Equal / 287
 - 13.3.2 Defining/Quantifying Reliability / 289
 - 13.3.3 Modeling Software Reliability / 291
 - 13.3.4 Certification Testing / 294
 - 13.3.5 Reliability Estimation and Reliability Improvement / 295
 - 13.3.6 Reliability Standards / 299
 - 13.3.7 Reliability as an Economic Function / 300
- 13.4 Chapter Summary / 307