PAUL AMMANN AND JEFF OFFUTT

INTRODUCTION TO SOFTWARE TESTING

Second Edition

SKN008787

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.

005.30287
A518

INTRODUCTION TO SOFTWARE TESTING

Paul Ammann
George Mason University

Jeff Offutt
George Mason University

THƯ VIỆN TRƯỜNG ĐHSPKT
SKN 008787

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

© Paul Ammann and Jeff Offutt 2017

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

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/.

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party Internet Web sites referred to in this publication and does not guarantee that any content on such Web sites is, or will remain, accurate or appropriate.

Contents

List of Figures *page* ix
List of Tables xii
Preface to the Second Edition xiv

Part 1 Foundations 1
1 Why Do We Test Software? 3
1.1 When Software Goes Bad 4
1.2 Goals of Testing Software 8
1.3 Bibliographic Notes 17

2 Model-Driven Test Design 19
2.1 Software Testing Foundations 20
2.2 Software Testing Activities 21
2.3 Testing Levels Based on Software Activity 22
2.4 Coverage Criteria 25
2.5 Model-Driven Test Design 27
2.5.1 Test Design 28
2.5.2 Test Automation 29
2.5.3 Test Execution 29
2.5.4 Test Evaluation 29
2.5.5 Test Personnel and Abstraction 29
2.6 Why MDTD Matters 31
2.7 Bibliographic Notes 33

3 Test Automation 35
3.1 Software Testability 36
3.2 Components of a Test Case 36

3.3 A Test Automation Framework 39
3.3.1 The JUnit Test Framework 40
3.3.2 Data-Driven Tests 44
3.3.3 Adding Parameters to Unit Tests 47
3.3.4 JUnit from the Command Line 50
3.4 Beyond Test Automation 50
3.5 Bibliographic Notes 53

4 Putting Testing First 54
4.1 Taming the Cost-of-Change Curve 54
4.1.1 Is the Curve Really Tamed? 56
4.2 The Test Harness as Guardian 57
4.2.1 Continuous Integration 58
4.2.2 System Tests in Agile Methods 59
4.2.3 Adding Tests to Legacy Systems 60
4.2.4 Weaknesses in Agile Methods for Testing 61
4.3 Bibliographic Notes 62

5 Criteria-Based Test Design 64
5.1 Coverage Criteria Defined 64
5.2 Infeasibility and Subsumption 68
5.3 Advantages of Using Coverage Criteria 68
5.4 Next Up 70
5.5 Bibliographic Notes 70

Part 2 Coverage Criteria 73

6 Input Space Partitioning 75
6.1 Input Domain Modeling 77
6.1.1 Interface-Based Input Domain Modeling 79
6.1.2 Functionality-Based Input Domain Modeling 79
6.1.3 Designing Characteristics 80
6.1.4 Choosing Blocks and Values 81
6.1.5 Checking the Input Domain Model 84
6.2 Combination Strategies Criteria 86
6.3 Handling Constraints Among Characteristics 92
6.4 Extended Example: Deriving an IDM from JavaDoc 93
6.4.1 Tasks in Designing IDM-Based Tests 93
6.4.2 Designing IDM-Based Tests for Iterator 94
6.5 Bibliographic Notes 102

7 Graph Coverage 106
7.1 Overview 106
7.2 Graph Coverage Criteria 111
7.2.1 Structural Coverage Criteria 112
7.2.2 Touring, Sidetrips, and Detours 116
7.2.3 Data Flow Criteria 123
7.2.4 Subsumption Relationships Among Graph Coverage Criteria 130

7.3 Graph Coverage for Source Code 131
7.3.1 Structural Graph Coverage for Source Code 132
7.3.2 Data Flow Graph Coverage for Source Code 136
7.4 Graph Coverage for Design Elements 146
7.4.1 Structural Graph Coverage for Design Elements 147
7.4.2 Data Flow Graph Coverage for Design Elements 148
7.5 Graph Coverage for Specifications 157
7.5.1 Testing Sequencing Constraints 157
7.5.2 Testing State Behavior of Software 160
7.6 Graph Coverage for Use Cases 169
7.6.1 Use Case Scenarios 171
7.7 Bibliographic Notes 173

8 Logic Coverage 177
8.1 Semantic Logic Coverage Criteria (Active) 178
8.1.1 Simple Logic Expression Coverage Criteria 179
8.1.2 Active Clause Coverage 181
8.1.3 Inactive Clause Coverage 185
8.1.4 Infeasibility and Subsumption 186
8.1.5 Making a Clause Determine a Predicate 187
8.1.6 Finding Satisfying Values 192
8.2 Syntactic Logic Coverage Criteria (DNF) 197
8.2.1 Implicant Coverage 198
8.2.2 Minimal DNF 199
8.2.3 The MUMCUT Coverage Criterion 200
8.2.4 Karnaugh Maps 205
8.3 Structural Logic Coverage of Programs 208
8.3.1 Satisfying Predicate Coverage 212
8.3.2 Satisfying Clause Coverage 213
8.3.3 Satisfying Active Clause Coverage 215
8.3.4 Predicate Transformation Issues 217
8.3.5 Side Effects in Predicates 220
8.4 Specification-Based Logic Coverage 223
8.5 Logic Coverage of Finite State Machines 226
8.6 Bibliographic Notes 231

9 Syntax-Based Testing 234
9.1 Syntax-Based Coverage Criteria 234
9.1.1 Grammar-Based Coverage Criteria 234
9.1.2 Mutation Testing 237
9.2 Program-Based Grammars 241
9.2.1 BNF Grammars for Compilers 241
9.2.2 Program-Based Mutation 242
9.3 Integration and Object-Oriented Testing 259
9.3.1 BNF Integration Testing 259
9.3.2 Integration Mutation 259
9.4 Specification-Based Grammars 266
9.4.1 BNF Grammars 266

9.4.2 Specification-Based Mutation 267
9.5 Input Space Grammars 271
9.5.1 BNF Grammars 271
9.5.2 Mutating Input Grammars 273
9.6 Bibliographic Notes 281

Part 3 Testing in Practice 283

10 Managing the Test Process 285
10.1 Overview 285
10.2 Requirements Analysis and Specification 286
10.3 System and Software Design 287
10.4 Intermediate Design 288
10.5 Detailed Design 288
10.6 Implementation 289
10.7 Integration 289
10.8 System Deployment 290
10.9 Operation and Maintenance 290
10.10 Implementing the Test Process 291
10.11 Bibliographic Notes 291

11 Writing Test Plans 292
11.1 Level Test Plan Example Template 293
11.2 Bibliographic Notes 295

12 Test Implementation 296
12.1 Integration Order 297
12.2 Test Doubles 298
12.2.1 Stubs and Mocks: Variations of Test Doubles 299
12.2.2 Using Test Doubles to Replace Components 300
12.3 Bibliographic Notes 303

13 Regression Testing for Evolving Software 304
13.1 Bibliographic Notes 306

14 Writing Effective Test Oracles 308
14.1 What Should Be Checked? 308
14.2 Determining Correct Values 310
14.2.1 Specification-Based Direct Verification of Outputs 310
14.2.2 Redundant Computations 311
14.2.3 Consistency Checks 312
14.2.4 Metamorphic Testing 312
14.3 Bibliographic Notes 314

List of Criteria 316

Bibliography 318

Index 337

Figures

1.1 Cost of late testing 12
2.1 Reachability, Infection, Propagation, Revealability (RIPR) model 21
2.2 Activities of test engineers 22
2.3 Software development activities and testing levels – the "V Model" 23
2.4 Model-driven test design 30
2.5 Example method, CFG, test requirements and test paths 31
3.1 Calc class example and JUnit test 41
3.2 Minimum element class 42
3.3 First three JUnit tests for Min class 43
3.4 Remaining JUnit test methods for Min class 45
3.5 Data-driven test class for Calc 46
3.6 JUnit Theory about sets 48
3.7 JUnit Theory data values 48
3.8 AllTests for the Min class example 49
4.1 Cost-of-change curve 55
4.2 The role of user stories in developing system (acceptance) tests 60
6.1 Partitioning of input domain *D* into three blocks 76
6.2 Subsumption relations among input space partitioning criteria 89
7.1 Graph (a) has a single initial node, graph (b) multiple initial nodes, and graph (c) (rejected) with no initial nodes 108
7.2 Example of paths 108
7.3 A Single-Entry Single-Exit graph 110
7.4 Test case mappings to test paths 110
7.5 A set of test cases and corresponding test paths 111
7.6 A graph showing Node Coverage and Edge Coverage 114
7.7 Two graphs showing prime path coverage 116
7.8 Graph with a loop 117
7.9 Tours, sidetrips, and detours in graph coverage 117
7.10 An example for prime test paths 119
7.11 A graph showing variables, def sets and use sets 124
7.12 A graph showing an example of du-paths 126

7.13 Graph showing explicit def and use sets 128
7.14 Example of the differences among the three data flow coverage criteria 129
7.15 Subsumption relations among graph coverage criteria 132
7.16 CFG fragment for the *if-else* structure 133
7.17 CFG fragment for the *if* structure without an *else* 133
7.18 CFG fragment for the *if* structure with a *return* 133
7.19 CFG fragment for the *while* loop structure 134
7.20 CFG fragment for the *for* loop structure 134
7.21 CFG fragment for the *do-while* structure 135
7.22 CFG fragment for the *while* loop with a *break* structure 136
7.23 CFG fragment for the *case* structure 136
7.24 CFG fragment for the *try-catch* structure 137
7.25 Method patternIndex() for data flow example 141
7.26 A simple call graph 147
7.27 A simple inheritance hierarchy 148
7.28 An inheritance hierarchy with objects instantiated 149
7.29 An example of parameter coupling 150
7.30 Coupling du-pairs 151
7.31 Last-defs and first-uses 151
7.32 Quadratic root program 153
7.33 Def-use pairs under intra-procedural and inter-procedural data flow 154
7.34 Def-use pairs in object-oriented software 155
7.35 Def-use pairs in web applications and other distributed software 155
7.36 Control flow graph using the File ADT 159
7.37 Elevator door open transition 161
7.38 Watch–Part A 163
7.39 Watch–Part B 164
7.40 An FSM representing Watch, based on control flow graphs of the methods 165
7.41 An FSM representing Watch, based on the structure of the software 165
7.42 An FSM representing Watch, based on modeling state variables 167
7.43 ATM actor and use cases 169
7.44 Activity graph for ATM withdraw funds 172
8.1 Subsumption relations among logic coverage criteria 187
8.2 Fault detection relationships 202
8.3 Thermostat class 210
8.4 PC true test for Thermostat class 213
8.5 CC test assignments for Thermostat class 214
8.6 Calendar method 226
8.7 FSM for a memory car seat–Nissan Maxima 2012 227
9.1 Method Min and six mutants 243
9.2 Mutation testing process 246
9.3 Partial truth table for $(a \wedge b)$ 253
9.4 Finite state machine for SMV specification 268
9.5 Mutated finite state machine for SMV specification 269
9.6 Finite state machine for bank example 271

9.7 Finite state machine for bank example grammar 272
9.8 Simple XML message for books 274
9.9 XML schema for books 275
12.1 Test double example: Replacing a component 300

"My favorite text-book on software testing just became even better! I particularly like the new chapters on model-driven testing, JUnit, test plans and test oracles. As before it covers testing concepts and techniques that are important for students and practitioners alike. The new edition is still compact, the writing is [illegible] and easily understandable. A must for any software engineering library."

Anneliese Amschler, [illegible], University of Denver

"This book manages to [illegible] into concise, clear, and precise form – the most important concepts pertaining to [illegible] testing. This is the best introduction to the topic that is available, as it includes many [illegible] and practical insights that will provide the readers with a solid foundation in the subject matter."

Lionel Briand, Interdisciplinary Centre for Security, Reliability, and Trust (SnT), University of Luxembourg

"This software testing textbook creatively uses only four models to capture current software testing techniques helping students, researchers, and practitioners to deeply grasp these techniques from an abstract, systematic perspective. The textbook provides excellent balance between theory and practice! The 2nd edition includes valuable new contents to allow readers to learn these software testing techniques in common settings in industrial practices."

Tao Xie, University of Illinois at Urbana-Champaign

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.

Online Resources
www.cambridge.org/ammann&offutt

- Powerpoint slides
- Example programs list
- Figures and tables from the book
- Solution manuals

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

ISBN 978-1-107-17201-2

9 781107 172012 >

Cover design: Andrew Ward