FOUNDATIONS OF SOFTWARE TESTING

Draft V2.0

Aditya P. Mathur

Purdue University

Contents

Preface . . . 13

Acknowledgements . . . 14

Part I: Preliminaries **16**

1. Preliminaries **17**

1.1. Humans, errors, and testing . . . 17

1.1.1. Errors, faults, and failures . . . 19

1.1.2. Test automation and the importance of testing . . . 20

1.1.3. Developer and tester as two roles . . . 20

1.2. Software quality . . . 20

1.2.1. Quality attributes . . . 21

1.2.2. Reliability . . . 22

1.3. Requirements, behavior, and correctness . . . 22

1.3.1. Input domain and program correctness . . . 23

1.3.2. Valid and invalid inputs . . . 25

1.4. Correctness versus reliability . . . 26

1.4.1. Correctness . . . 26

1.4.2. Reliability . . . 27

1.4.3. Program use and the operational profile . . . 27

1.5. Testing and debugging . . . 28

1.5.1. Preparing a test plan . . . 28

1.5.2. Constructing test data . . . 28

1.5.3. Executing the program . . . 31

1.5.4. Specifying program behavior . . . 31

1.5.5. Assessing the correctness of program behavior . . . 33

1.5.6. Construction of oracles . . . 35

1.6. Testing and verification . . . 37

1.7. Control Flow Graphs . . . 37

1.7.1. Basic blocks . . . 38

1.7.2. Flow graph: definition and pictorial representation . . . 39

1.7.3. Paths 41
1.8. Model-based testing 44
1.9. Strings, languages, and regular expressions 47
1.10. Finite state machines 48
1.10.1. Excitation using an input sequence 51
1.10.2. Tabular representation 53
1.10.3. Properties of FSM 53
1.11. Statecharts 55
1.11.1. States 55
1.11.2. Transitions 57
1.11.3. Pseudo states 58
1.11.4. Events 61
1.11.5. Actions 63
1.11.6. Statechart and behavior 64
1.11.7. Compound transitions 67
1.11.8. Scope of states and transitions 69
1.12. Types of Testing 70
1.12.1. Classifier: C1: Source of test generation 71
1.12.2. Classifier: C2: Life cycle phase 73
1.12.3. Classifier: C3: Goal directed testing 74
1.12.4. Classifier: C4: Artifact under test 75
1.13. Summary 77
Bibliographic Notes 77
Exercises 80

2. Errors, faults, and Failures 85

3. Software Test Process 87

Part II: Test Generation 88

4. Test Generation: From Requirements 89
4.1. Introduction 89
4.2. The test selection problem 91
4.3. Equivalence partitioning 92
4.3.1. Faults targeted 93
4.3.2. Relations and equivalence partitioning 94
4.3.3. Equivalence classes for variables 98
4.3.4. Uni-dimensional versus multidimensional equivalence partitioning 101
4.3.5. A systematic procedure for equivalence partitioning 103
4.3.6. Test selection based on equivalence classes 107

4.3.7. GUI design and equivalence classes . . . 110
4.4. Boundary value analysis . . . 111
4.5. Category-partition method . . . 117
4.6. Cause-effect graphing . . . 124
4.6.1. Notation used in Cause-effect graphing . . . 124
4.6.2. Creating cause-effect graphs . . . 127
4.6.3. Decision table from cause-effect graph . . . 130
4.6.4. Heuristics to avoid combinatorial explosion . . . 134
4.6.5. Test generation from a decision table . . . 137
4.7. Test generation from predicates . . . 137
4.7.1. Predicates and Boolean expressions . . . 137
4.7.2. Fault model for predicate testing . . . 139
4.7.3. Predicate constraints . . . 141
4.7.4. Predicate testing criteria . . . 143
4.7.5. Generating BOR, BRO, and BRE adequate tests . . . 144
4.7.6. Cause effect graphs and predicate testing . . . 156
4.7.7. Fault propagation . . . 157
4.7.8. Predicate testing in practice . . . 159
4.8. Summary . . . 162
Bibliographic Notes . . . 162
Exercises . . . 164

5. Test Generation: Random 171

6. Test Generation: FSM Models 173
6.1. Software design and testing . . . 173
6.2. Conformance testing . . . 175
6.2.1. Reset inputs . . . 176
6.2.2. The testing problem . . . 178
6.3. A fault model . . . 179
6.3.1. Mutants of FSMs . . . 180
6.3.2. Fault coverage . . . 183
6.4. Characterization set . . . 183
6.4.1. Construction of the k-equivalence partitions . . . 185
6.4.2. Deriving the characterization set . . . 187
6.4.3. Identification sets . . . 189
6.5. The W-method . . . 190
6.5.1. Assumptions . . . 190
6.5.2. Maximum number of states . . . 191
6.5.3. Computation of the transition cover set . . . 191
6.5.4. Constructing Z . . . 193

6.5.5. Deriving a test set 193
6.5.6. Testing using the W-method 194
6.5.7. The error detection process 196
6.6. The UIO-sequence method 197
6.6.1. Assumptions 197
6.6.2. UIO sequences 197
6.6.3. Core and non-core behavior 199
6.6.4. Generation of UIO sequences 201
6.6.5. Distinguishing signatures 211
6.6.6. Test generation 213
6.6.7. Test optimization 214
6.6.8. Fault detection 216
6.7. The partial W-method 218
6.7.1. Testing using the Wp-method for $m = n$ 219
6.7.2. Testing using the Wp-method for $m > n$ 221
6.8. Automata theoretic versus control-flow based techniques 223
6.8.1. n-switch-cover 226
6.8.2. Comparing automata theoretic methods 228
6.9. Summary 229
Bibliographic Notes 229
Exercises 232

7. Test Generation: Statecharts 237
7.1. Test generation from Statecharts 237
7.2. Statecharts as I/O transformers 238
7.3. Distinguishing pairs of states 239
7.4. Statecharts with no compound states 241
7.5. Test harness and internal events 243
7.6. Handling partial labels 245
7.7. Discarding and pruning tests 247
7.8. Handling connectors 251
7.9. Handling guards 254
7.9.1. Constructing P and W 255
7.9.2. Generation of variable-value pairs 258
7.10. Statecharts with OR states 261
7.11. Interlevel transitions 266
7.12. History connectors 267
7.13. Statecharts with AND states 270
7.14. Handling forks and merges 273
7.15. Fault detection in statecharts and the need for W-sets 276
7.16. Exploiting hierarchy in test generation 278

7.17. Test assessment and enhancement . . . 279
7.17.1. Statechart-based assessment . . . 279
7.17.2. IUT-based assessment . . . 280
7.18. Summary . . . 280
Bibliographic Notes . . . 281
Exercises . . . 283

8. Test Generation: Timed I/O Automata 293

8.1. Introduction . . . 293
8.2. Overview of the test methodology . . . 294
8.3. Timed automata . . . 296
8.3.1. Informal introduction . . . 296
8.3.2. Interpretation of clock guards . . . 299
8.3.3. Timed trace . . . 301
8.3.4. Clocks and real-time systems . . . 302
8.3.5. Time assignments . . . 305
8.3.6. Equivalence of time assignments . . . 306
8.3.7. Timed automata and Timed Input/Output Automata . . . 308
8.3.8. Clock regions . . . 309
8.3.9. Successor regions . . . 311
8.3.10. Region graphs . . . 312
8.4. Fault model . . . 314
8.4.1. Transfer and action faults . . . 314
8.4.2. Clock reset faults . . . 315
8.4.3. Faults in the specification of timing constraints . . . 318
8.5. Tests and delays . . . 320
8.6. Clock granularity . . . 322
8.7. Observing the IUT during test . . . 324
8.8. Test generation using TIOA models . . . 326
8.8.1. Grid automata . . . 327
8.8.2. NTFSMs and generation from grid automata . . . 330
8.8.3. Trace equivalence . . . 333
8.8.4. Delay and input identified states . . . 335
8.8.5. Test Generation using the timed Wp method . . . 336
8.8.6. Assumptions that guarantee fault detection . . . 341
8.8.7. Mapping faults to NTFSM . . . 343
8.8.8. Fault detection . . . 344
8.9. Summary . . . 347
Bibliographic Notes . . . 347
Exercises . . . 350

9. Test Generation: UML Specifications 355

10. Test Generation: Code Based 357

11. Test Generation: Regression 359

12. Test Generation: Combinatorial Designs 361

12.1. Combinatorial designs . . . 361
12.1.1. Test configuration and test set . . . 362
12.1.2. Modeling the input and configuration spaces . . . 362
12.2. A combinatorial test design process . . . 366
12.3. Fault model . . . 368
12.4. Latin Squares . . . 371
12.5. Mutually orthogonal Latin squares . . . 372
12.6. Pairwise designs: binary factors . . . 374
12.7. Pairwise design: multi-valued factors . . . 378
12.8. Orthogonal Arrays . . . 384
12.9. Covering and mixed-level covering arrays . . . 388
12.10. Arrays of strength > 2 . . . 390
12.11. Generating covering arrays . . . 391
12.12. Summary . . . 398
Bibliographic Notes . . . 398
Exercises . . . 401

13. Test Generation: Formal Specifications 405

13.1. Introduction . . . 405
13.2. The Z notation in brief . . . 407
13.2.1. A Z specification . . . 407
13.2.2. Sets . . . 407
13.2.3. Types . . . 409
13.2.4. Expressions and types . . . 409
13.2.5. Predicates . . . 410
13.2.6. Sequence . . . 410
13.2.7. Schema . . . 412
13.2.8. Combining schemas . . . 413
13.2.9. Operations and features . . . 418
13.3. Test generation using the Test Template framework . . . 419
13.3.1. Valid input domain . . . 420
13.3.2. Test templates . . . 421
13.3.3. Template division strategies . . . 423
13.3.4. Domain propagation . . . 424
13.3.5. Number of occurrences . . . 425

13.3.6. Specification mutation . . . 426
13.3.7. Test template instantiation . . . 428
13.3.8. Range of operations and the oracle . . . 431
13.4. Robustness testing using formal specifications . . . 433
13.5. Digression from formal specifications . . . 435
13.6. Summary . . . 435
Bibliographic Notes . . . 436
Exercises . . . 439

14. Test Generation: Concurrent and Distributed Systems 443

Part III: Test Adequacy Assessment and Enhancement 445

15. Test Adequacy: Control Flow and Data Flow 447
15.1. Test Adequacy: Basics . . . 447
15.1.1. What is test adequacy? . . . 447
15.1.2. Measurement of test adequacy . . . 448
15.1.3. Test enhancement using measurements of adequacy . . . 450
15.1.4. Infeasibility and test adequacy . . . 454
15.1.5. Error detection and test enhancement . . . 456
15.1.6. Single and multiple executions . . . 458
15.2. Adequacy Criteria based on Control Flow . . . 459
15.2.1. Statement and block coverage . . . 459
15.2.2. Conditions and decisions . . . 462
15.2.3. Decision coverage . . . 464
15.2.4. Condition coverage . . . 465
15.2.5. Condition/decision coverage . . . 467
15.2.6. Multiple condition coverage . . . 468
15.2.7. Linear code sequence and jump (LCSAJ) coverage . . . 471
15.2.8. Modified condition/decision coverage . . . 474
15.2.9. Generating MC/DC adequate tests for compound conditions . . . 475
15.2.10. Definition of MC/DC coverage . . . 479
15.2.11. Minimal MC/DC tests . . . 485
15.2.12. Error detection and MC/DC adequacy . . . 485
15.2.13. Short-circuit evaluation and infeasibility . . . 487
15.2.14. Tracing test cases to requirements . . . 488
15.3. Concepts from Data Flow . . . 490
15.3.1. Definitions and uses . . . 491
15.3.2. C-use and p-use . . . 492
15.3.3. Global and local definitions and uses . . . 493
15.3.4. Data-flow graph . . . 493

15.3.5. Def-clear paths . . . 495
15.3.6. Def-use pairs . . . 496
15.3.7. Def-use chains . . . 497
15.3.8. A little optimization . . . 498
15.3.9. Data contexts and ordered data contexts . . . 498
15.4. Adequacy criteria based on data flow . . . 500
15.4.1. *c-use* coverage . . . 501
15.4.2. *p-use* coverage . . . 502
15.4.3. *all-uses* coverage . . . 504
15.4.4. *k-dr* chain coverage . . . 504
15.4.5. Using the *k-dr* chain coverage . . . 506
15.4.6. Infeasible c- and p-uses . . . 506
15.4.7. Context coverage . . . 507
15.5. Control Flow versus Data Flow . . . 509
15.6. The "subsumes" Relation . . . 511
15.7. Structural and Functional Testing . . . 512
15.8. Scalability of coverage measurement . . . 514
15.9. Summary . . . 516
Bibliographic Notes . . . 516
Exercises . . . 522

16. Test adequacy: Program Mutation 529
16.1. Introduction . . . 529
16.2. Mutation and mutants . . . 530
16.2.1. First-order and higher-order mutants . . . 531
16.2.2. Syntax and semantics of mutants . . . 532
16.2.3. Strong and weak mutations . . . 535
16.2.4. Why mutate? . . . 536
16.3. Test assessment using mutation . . . 537
16.3.1. A procedure for test adequacy assessment . . . 537
16.3.2. Alternate procedures for test adequacy assessment . . . 546
16.3.3. "Distinguished" versus "killed" mutants . . . 546
16.3.4. Conditions for distinguishing a mutant . . . 546
16.4. Mutation operators . . . 548
16.4.1. Operator types . . . 549
16.4.2. Language dependence of mutation operators . . . 551
16.5. Design of mutation operators . . . 552
16.5.1. Goodness criteria for mutation operators . . . 552
16.5.2. Guidelines . . . 553
16.6. Founding principles of mutation testing . . . 554
16.6.1. The competent programmer hypothesis . . . 554

16.6.2. The coupling effect 554
16.7. Equivalent mutants 555
16.8. Fault detection using mutation 556
16.9. Types of mutants 558
16.10. Mutation operators for C 559
16.10.1. What is not mutated? 559
16.10.2. Linearization 560
16.10.3. Execution Sequence 561
16.10.4. Effect of an execution sequence 564
16.10.5. Global and local identifier Sets 565
16.10.6. Global and local reference sets 565
16.10.7. Mutating program constants 568
16.10.8. Mutating operators 569
16.10.9. Mutating statements 574
16.10.10Mutating program variables 586
16.11. Mutation operators for Java 590
16.11.1. Traditional mutation operators 592
16.11.2. Inheritence 592
16.11.3. Polymrophism and dynamic binding 596
16.11.4. Method overloading 597
16.11.5. Java specific mutation operators 599
16.12. Mutation operators for Fortran 77, C, and Java: a comparison 600
16.13. Tools for mutation testing 602
16.14. Mutation testing within budget 603
16.14.1. Prioritizing functions to be mutated 604
16.14.2. Selecting a subset of mutation operators 604
16.15. Summary 605
Bibliographic Notes 606
Exercises 617

17. Empirical Studies of Fault Detection Effectiveness 625

18. Theoretical Perspectives on Software Testing 627

Part IV: Testing Specific Software Artifacts and their Properties 628

19. GUI Testing 629

20. Testing for Security 631

21. Testing Object-oriented Systems 633

22. Testing WEB Applications **635**

23. Testing Databases **637**

24. Fault Injection Testing **639**

Bibliography 640