Software

System Testing and Quality Assurance

Boris Beizer

Van Nostrand Reinhold Electrical/Computer Science and Engineering Series

BIBLIOTHEQUE DU CERIST

C
1642

Centre de Recherche sur l'Information Scientifique et Technique
Bibliothéque

SOFTWARE SYSTEM TESTING AND QUALITY ASSURANCE

Boris Beizer
Data Systems Analysts, Inc.
Pennsauken, New Jersey

Van Nostrand Reinhold Electrical/Computer Science and Engineering Series

VNR VAN NOSTRAND REINHOLD COMPANY
New York

BIBLIOTHEQUE DU CERIST

PREFACE

BIBLIOTHEQUE DU CERIST

This book concerns system software testing and quality assurance methods. The companion volume, *Software Testing Techniques* (BEIZ82), is concerned with structural and functional test techniques that are basic to all testing. The sum is not equal to the whole of its parts: the best designed, implemented, and tested individual units can come together in hopeless chaos in the absence of system-wide test plans, quality assurance, and integration testing. Unit testing, no matter how well done, just isn't enough.

I set out to write a single book that would cover the gamut from testing to quality assurance—from unit to system—and from individual test and quality assurance methods to system testing and quality assurance. It would have been a big, cumbersome book, and holding it to a publishable size would have required unfortunate compromises. The book would have been unsatisfying to me and to most readers. Accordingly, with Van Nostrand Reinhold's concurrence, I declared a book split, which yielded some unexpected dividends. It was no longer necessary to compromise between unit-level and system-level issues or between the programmer's and manager's points of views.

Software Testing Techniques focuses on what the individual programmer can and should do to create reliable software units. It is written for the programmer from a programmer's point of view. I thought at first that *this* book would have a single point of view, but soon found that there were four distinct voices with contradictory demands vying for my attention. The subject of system testing and quality assurance can be seen from the project manager's or the quality assurance manager's point of view and can be addressed to the project manager or to the quality assurance manager—for four voices in all. I quickly put the idea of a four-way split out of my mind—it would have yielded four thin books. I have instead used all four voices in this book. When discussing desk checking, for example, an activity in which I believe quality assurance has no productive role to play, it is as project manager to project manager. The problems of how to select QA personnel and how to shield them from hostility are clearly a conversation between QA managers. The inherent conflicts in system functional testing lead to the crossed conversations. The voice, therefore, changes from section to section, and within sections, without warning. There are so many conflicting goals inherent in system testing and quality assurance that it's essential that all parties in the process understand each other's point of view, and shifting it is one way to help that understanding.

Actually, while there may be beneficial tensions, there are no real conflicts, despite appearances to the contrary, just as there should be no real conflict between the programmer and program manager. In the ideal project, unit testing is well planned, thorough, well documented, and well managed. There is a hierarchy of testing methods from unit to system. The programmer can work in confidence that management and quality assurance methods are in place that relieve much of the uncertainty of interfacing with other programs. Well-wrought pieces of software come together because someone has taken the trouble to emplace the methods that assure that that will be the case. The project manager is not burdened with agonizing over the outcome of thousands of microscopic unit tests, because those tests are developed and executed under a uniform methodology. There is no need to agonize over what might have been forgotten, because quality assurance keeps track of everything. There is no need to worry about what surprises may come from a user's acceptance test, or from the first few months in the field, because quality assurance has savaged the system more than any user ever could, and done it in privacy, where the consequences are merely extra work rather than litigation.

Boris Beizer
Abington, Pennsylvania

BIBLIOTHEQUE DU CERIST

CONTENTS

PREFACE xi

1—INTRODUCTION 1

1. Testing and Quality Assurance Goals 1
    - 1.1. What We Do 1
    - 1.2. Attitudes and Beatitudes 2
    - 1.3. Productivity and Quality Assurance Goals 2
    - 1.4. Test Design and Design for Test 4
2. Some Dichotomies 5
    - 2.1. Testing Versus Debugging 5
    - 2.2. Units, Subsystems, and Such 6
    - 2.3. Testing Versus Integration 7
    - 2.4. Structure Versus Function 7
    - 2.5. Designer Versus Tester 8
    - 2.6. Builder Versus Buyer 9
3. A Model Project 10
4. The Search for El Dorado 11
    - 4.1. Do Bugs Exist? 11
    - 4.2. Is Complete Testing Possible? 12
    - 4.3. Realistic Goals 14

2—BUGS IN PERSPECTIVE 16

1. Synopsis 16
2. Bug Measures 16
    - 2.1. Can Bugs Be Measured? 16
    - 2.2. Can Bugs Be Counted? 17
    - 2.3. Public and Private Bugs 18
    - 2.4. Equal Rights for Bugs? 19
3. A Taxonomy for Bugs 21
    - 3.1. General 21
    - 3.2. Function-Related Bugs 22
    - 3.3. System Bugs 23

BIBLIOTHEQUE DU CERIST

3.4. Process Errors 26
3.5. Data Errors 27
3.6. Code Errors 32
3.7. Testing and Style 33
4. Some Bug Statistics 33
5. Tasks for a Quality Assurance Department 35

3—OVERVIEW OF TEST TECHNIQUES 37

1. Synopsis 37
2. Is There a Best Method? 37
3. Path Testing 37
3.1. General 37
3.2. Fundamental Path Selection Criteria 39
3.3. Loop Testing 42
3.4. Test Execution Time 44
3.5. The Rest of Path Test Design 45
3.6. Tasks for a Quality Assurance Department 51
4. Transaction Flow Testing 51
4.1. General 51
4.2. Transaction Flows 52
4.3. Transaction-Flow Testing 55
4.4. Tasks for a Quality Assurance Department 59
5. Input Validation and Syntax Testing 60
5.1. The Hostile World 60
5.2. Principles 62
5.3. Test Case Generation 64
5.4. Running, Validation, and Ad-lib Tests 68
5.5. Application 70
5.6. Tasks for a Quality Assurance Department 71
6. Logic-Based Testing 71
6.1. Definitions and Notation 71
6.2. Decision Tables as a Basis for Test Case Design 73
6.3. Expansion of Immaterial Cases 74
6.4. Test Case Design 75
6.5. Decision Tables and Structure 77
6.6. Logic and Boolean Algebra in Test Design 78
6.7. Tasks for a Quality Assurance Department 79
7. State Transition Testing 79
7.1. General 79
7.2. Good State Graphs and Bad 81
7.3. State Testing 87
8. Tasks for a Quality Assurance Department 89

BIBLIOTHEQUE DU CERIST

4—UNIT TESTING, ELEMENT TESTING, AND QUALITY ASSURANCE 91

1. Synopsis 91
2. Software Sociology 91
   2.1. Unit Testing 91
   2.2. Subjective Goals 93
   2.3. Objective Goals 95
3. Software Anthropology 98
   3.1. Prerequisites 98
   3.2. Mores and Taboos 103
   3.3. Environment 108
4. Unit Testing Procedures and Execution 113
   4.1. General 113
   4.2. Mechanical Analysis 113
   4.3. Desk Checking 119
   4.4. Reviews and Audits 124
   4.5. Execution, Correction, and Refinement 131
   4.6. Implementation 132
5. Higher Level Element Testing 133
   5.1. General 133
   5.2. Decision Nodes and Path Testing 133
   5.3. Syntax-Directed Testing 134
   5.4. Logic-Based Testing 134
   5.5. State Testing 135
   5.6. The Question of Stratification 135
   5.7. Stratified Designs 138
6. Tasks for a Quality Assurance Department 139

5—INTEGRATION 141

1. Synopsis 141
2. Objectives 141
3. Integration Tactics 143
   3.1. Graphs and Standards 143
   3.2. What to Test 147
   3.3. Data Corruption and Residues 152
   3.4. Special Problems 154
4. Integration Strategy 156
   4.1. Objectives 156
   4.2. Top-Down Integration and Testing 157
   4.3. Bottom-Up Testing and Integration 159
   4.4. Big-Bang Testing 160
   4.5. An Integration Strategy 161
5. Who Integrates? 162
6. Tasks for a Quality Assurance Department 163

BIBLIOTHEQUE DU CERIST

6—SYSTEM TESTING 165

1. Synopsis 165
2. What's Left to Test? 165
3. Cross-Reference Construction 166
   3.1. Objectives 166
   3.2. Specification Cross-Reference 167
   3.3. Design Documentation Cross-Reference 170
   3.4. The Data-Base Cross-Reference 171
   3.5. Expanding and Editing the File 171
   3.6. Interviews and Confrontations 173
4. System Functional Testing and Acceptance Testing 179
   4.1. Perspective 179
   4.2. The Test Plan 182
   4.3. The Tests 185
   4.4. Execution 192
5. Tasks for a Quality Assurance Department 198

7—CONFIGURATION, RECOVERY, AND SECURITY TESTING 200

1. Synopsis 200
2. Configuration Testing 200
   2.1. What Is It? 200
   2.2. System States 202
   2.3. Configuration Command Tests 205
   2.4. Rotation and Permutation Tests 207
   2.5. Degradation and Restoration Tests 209
3. Recovery Testing 211
   3.1. What Is It? 211
   3.2. Restart and Recovery 213
   3.3. Switchover 214
   3.4. Transaction Fidelity and Accountability Loss 216
4. Security Testing 218
   4.1. The Problem 218
   4.2. Frontal Attack Methods 222
   4.3. Indirect Attacks and Data Attacks 226
   4.4. Tiger Teams 230
5. Retrospective on Testing Costs 230
6. Tasks for a Quality Assurance Department 232

8—BACKGROUND, STRESS, AND PERFORMANCE TESTING 234

1. Synopsis 234
2. Transaction Generators 234
   2.1. General 234
   2.2. Real Loads 235

BIBLIOTHEQUE DU CERIST

2.3. Load Generators and Simulators 236
2.4. Cogeneration 240
2.5. Self-Generation 241
3. Loops and Snakes 242
3.1. What Are They? 242
3.2. Simple Snakes and Loops 243
3.3. Linear and Exponential Loops 245
3.4. Duplexed Loops 246
3.5. Loop Control 247
4. Background and Stress Testing 247
4.1. Background Testing 247
4.2. Stress Testing 250
5. Performance Testing 256
5.1. Objectives and Prerequisites 256
5.2. Performance Objectives 258
5.3. Instrumentation Issues 262
5.4. Software Instrumentation 263
5.5. Hardware Instrumentation 266
5.6. The Experimental Process 268
6. Tasks for a Quality Assurance Department 275

9—QUANTITATIVE METHODS 276

1. Synopsis 276
2. Quantitative Measures and Metrics 276
2.1. Philosophy 276
2.2. A Historical Perspective 277
2.3. Objectives 278
3. Complexity Measures 278
3.1. General 278
3.2. The Simplest Metric: Lines-of-Code 279
3.3. McCabe's and Related Metrics 280
3.4. Halstead's Metrics 282
3.5. Other Metrics 288
3.6. A Plan for Action 288
4. Software Reliability 289
4.1. General 289
4.2. The State of the Art 292
4.3. What's Missing? 293
4.4. Where Does That Leave Us? 294
5. When Will It Be Done? 295
5.1. Retrospective 295
5.2. Hitachi's Method 295
5.3. Other Methods of Tracking 304
6. Tasks for a Quality Assurance Department 306

BIBLIOTHEQUE DU CERIST

10—ACHIEVING QUALITY SOFTWARE 308

1. Synopsis 308
2. Testable Designs 308
   2.1. Changing the Perspective 308
   2.2. Techniques for Testable Designs 309
   2.3. Tools 311
   2.4. Performance Impact 312
3. Who Wants QA? 313
   3.1. Motives, Ulterior and Otherwise 313
   3.2. Up-Front Understanding 315
   3.3. Independence 315
4. Staffing and Management 317
   4.1. The Halt, the Lame, and the Blind 317
   4.2. The QA Worker 317
   4.3. Dealing with Hostility 319
   4.4. A Tour of Duty 320
5. Quality Assurance Squared 321
   5.1. The Problem 321
   5.2. Programming and QA Test Design Compared 321
   5.3. Test-Testing and Debugging 323
   5.4. The QA Manager's Role 325
   5.5. Be-Bugging 325
   5.6. Tit-for-Tat 326
6. Who Deserves the Credit, Who Deserves the Blame? 327
   6.1. A Charm Against Tigers 327
   6.2. Keeping Records and Public Relations 328
   6.3. Reasonable Expectations and Claims 329
   6.4. Internal and External Comparisons 329
7. Where to From Here? 329

REFERENCES 331

INDEX 349

BIBLIOTHEQUE DU CERIST