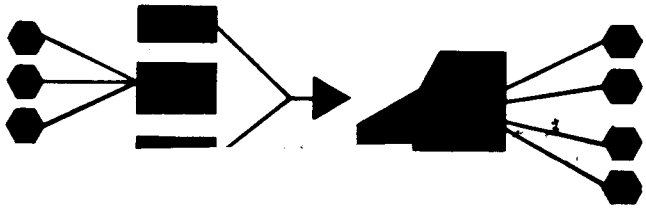
Software Testing and Evaluation

Richard A. DeMillo
W. Michael McCracken
R.J. Martin
John F. Passafiume

Software Testing and Evaluation

Richard A. DeMillo
W. Michael McCracken
R.J. Martin
John F. Passafiume

Software Engineering Research Center
Georgia Institute of Technology



THE BENJAMIN/CUMMINGS PUBLISHING COMPANY, INC.
Menlo Park, California • Reading, Massachusetts
Don Mills, Ontario • Wokingham, U.K. • Amsterdam • Sydney
Singapore • Tokyo • Madrid • Bogota • Santiago • San Juan

Sponsoring Editor: Alan Apt
Production Supervisor: Karen K. Gulliver
Cover Designer: Michael Rogondino

Library of Congress Cataloging-in-Publication Data

Software testing and evaluation

Includes index.
1. Computer software — Testing. 2. Computer software — Evaluation. I. DeMillo, Richard A.
QA76.76.T48S64 1987 005.1'4 86-7944
ISBN 0-8053-2535-2

ABCDEFGHIJ-MA-89876

The Benjamin/Cummings Publishing Company, Inc.
2727 Sand Hill Road
Menlo Park, California 94025

Preface

This book is an updated and edited version of the report of the Software Test and Evaluation Project to the Office of the Secretary of Defense (Research and Engineering). This report received wide circulation as a Technical Report issued by the Georgia Institute of Technology. While planning for a major revision and eventual publication of the report in book form, we continued to receive many requests for copies of the original document. Bill Riddle -- then Chairman of the ACM Special Interest Group in Software Engineering -- and Alan Apt of Benjamin Cummings were instrumental in persuading us that rapid publication of a slightly revised version would be a service to the community. The present volume is the result.

The Software Test and Evaluation Project (STEP) was initiated in 1981 by the Director Defense Test and Evaluation.† The primary objective of STEP was (and remains) the development of improved policy and guidance for use by the U.S. Department of Defense for the test and evaluation of computer software for so-called "mission-critical" applications. Our group at Georgia Tech was selected to develop and execute a plan for carrying out these improvements. As the project unfolded over the next three years, it became increasingly clear that the results had broad applicability in two directions. First, we were being called upon to develop a methodology for "technology transition" -- the rapid identification, demonstration and introduction to practical use of cutting-edge technologies in environments where they were critically needed. In the area of computer software (for commercial as well as mission-critical applications) it seemed to us that a structured approach to transitioning new technology into practical use might be useful in a variety of applications

† In 1984, special legislation of the U.S. Congress resulted in a reorganization of test and evaluation offices within the Department of Defense. Specifically, a Director of Operational Test and Evaluation was appointed to parallel the Development and Engineering Test and Evaluation carried out by the Office of the Undersecretary of Defense for Research and Engineering (Defense Test and Evaluation). In 1986, the principal office charged with these responsibilities was redesignated Deputy Undersecretary of Defense (Test and Evaluation). It is this latter office that oversees and directs STEP at the present time.

and settings.

Second, the approach which we settled upon -- the development of State-of-the-Art and State-of-Practice overviews -- turned out to be rich with generic technology and technology assessments. These overviews contained brief descriptions of major test methodologies, catalogs of automated tools to support them, essentially exhaustive bibliographies, case studies of good and bad examples of software testing and exegeses of major standards. With the exception of those materials that specifically addressed Military standards and regulations (reflecting the status of these documents in 1983), such overviews can be read and applied to any large-scale software engineering effort in which the test and evaluation of software effectiveness and suitability are major activities.

The context of these results is important. At the time that STEP was initiated, the role of software in escalating the cost and driving down the reliability of systems had become very visible. Virtually, every major Defense system planned or fielded over the previous decade contained at least one subsystem consisting of an embedded computer controlling some mission-critical function. Although hardware and software contributed in equal measure to the successful implementation of system functions, relative imbalances in their treatment during system development had long been observed. In 1974, for example, a Task Force of the Defense Science Board noted: "whereas the hardware development was monitored, tested and regularly evaluated, the software development was not." These findings were repeated in 1979. In 1980 and 1981, the U.S. Secretary of Defense (in his reports to Congress and the Military Services) directed the Armed Forces to "...give priority to development of tools and techniques for testing of embedded computers and software." He further directed that "Testing of software should be sufficient to achieve a balanced risk with the hardware of the same system."

The approach upon which we settled involved constructing two "snapshots". The first of these comprised a view of the sort of testing that could be supported by the state-of-the-art. This was essentially our view of what a technologically ideal world of software testing was capable of supporting. The second snapshot was a picture of current practices. There are methodological difficulties in constructing an overview of practices in a setting as vast as the U.S. Defense industry. Not the least among these are the many programs and organizations to be sampled and the lack of long-term institutional memory. However, by carefully selecting representative programs, organizations, and applications and by religiously adhering to strict data-gathering protocols an assessment of current practices emerged that we believe represents a composite view of how mission-critical software is actually tested. The technology shortfall -- the difference between what the ideal world would support and what was actually available to engineers -- was the gap to be closed.

Institutional forces within the Department of Defense and the Defense-related industries have responded encouragingly.

There are several keys to the success of this approach. We were at the outset in full agreement with the project sponsors that this was not to be a study in which sufficient "data" were gathered to support a previously identified conclusion. We were given sufficient resources and access to sufficiently many organizations to derive an objectively supportable assessment of what the true needs were. We were also encouraged to provide "top-down" recommendations to effect improvements. That is, for those instances in which the system development process needed to be modified, we were not discouraged from recommending such modifications.

The need for these improvements has scarcely diminished in the intervening five years. Software has assumed critical (often life-critical) roles in non-military applications ranging from telecommunications to civilian air traffic control. In the Defense sector, command and control, ballistic missile defense, and avionics are but three of the many mission areas in which software is the dominant source of system functionality (and, therefore, risk). The planned and disciplined engineering of software destined for these sorts of applications is, of course, essential. Part of any such engineering process is a technologically sound approach to testing.

Such an approach will incorporate:

- A chain of test plans and procedures that begins at the topmost levels of system development and proceeds through the most detailed levels.
- The use of varied software testing technologies employing automated tools and techniques that are appropriate to the criticality of the application.
- A system of reporting test results and deficiencies that supports objective evaluations of software system status.
- An effective decision-making apparatus, capable of incorporating evaluations of software status into overall assessments of risk associated with the development and eventual fielding of the system.

We intend this volume to be a sourcebook out of which such an approach can be constructed.

Many people have contributed to STEP since its inception. Charles Watt was Deputy Director Defense Test and Evaluation throughout most of the project and provided technical direction and leadership. Donald Greenlee (Deputy Director Operational Test and Evaluation, at this

writing) provided day-to-day oversight and guidance. In addition to their project management roles, both of them contributed technically in substantial ways. Edith Martin helped structure the technology transition methodology in its early stages. Control Data Corporation provided early technical support services, and the National Security Industrial Association (NSIA) provided technical forums for disseminating STEP results. Researchers at Georgia Tech who contributed to the effort include Sinasi Bilsel, Michael Merritt, and E. Pipat. Fred Sayward contributed substantially to the preparation of the STEP reports. Rena Faye Smith aided in the data gathering effort and helped compile much of the material in the overview of current practices. Jackson Dodsworth and Esther Richards entered most of the original manuscript. We gratefully acknowledge the contributions of all of these individuals.

This manuscript was typeset on an Imagen Laser Printer in the School of Information and Computer Science at Georgia Tech using Unix† document preparation tools. Figures and illustrations were developed on a Hewlett-Packard 9845C Graphics system in the Software Engineering Research Center. Ann Richliew and Glenn Barry provided invaluable assistance, for which we are grateful.

Richard A. DeMillo
W. Michael McCracken
R. J. Martin
John F. Passafiume

Georgia Institute of Technology

August 1986

† Unix is trademark of AT&T Bell Laboratories.

Contents

Preface

Part I, State-of-the-Art Overview

Chapter 1
Definitions and Theory of Testing 3

1.1 Program Specifications and Correctness 3
1.2 Reliability and Validity 6
1.3 Deductive Approaches -- Proofs of Correctness 8
1.4 Mathematical Terminology 12
1.5 Statistical Reliability Models 14

Chapter 2
Software Testing 19

2.1 Testing Strategies 19
2.2 Testing Techniques 27
2.2.1 Static Analysis Techniques 27
2.2.2 Symbolic Testing 30
2.2.3 Program Instrumentation 38
2.2.4 Program Mutation Testing 42
2.2.5 Input Space Partitioning 48
2.2.6 Functional Program Testing 53
2.2.7 Algebraic Program Testing 56
2.2.8 Random Testing 59
2.2.9 Grammar-Based Testing 60
2.2.10 Data-Flow Guided Testing 62
2.2.11 Compiler Testing 64
2.2.12 Real-Time Software and Testing 66
2.3 Other Strategies for Constructing Reliable Software 73
2.4 Comparative Evaluation of Testing Techniques 75

Chapter 3
Testing and Evaluation Tools 78

3.1 Introduction 78
3.1.1 General Views of Testing Tools 78
3.1.2 Classification 80
3.2 Static Analysis Tools 81
3.2.1 Static Analysis Tool Classification 81
3.2.2 Static Analyzers 83
Catalog Listing of Static Analyzer Tools 88
3.3 Dynamic Analysis Tools 96
3.3.1 Dynamic Tool Classification 96
3.3.2 Symbolic Evaluators 97
Catalog Listing of Symbolic Evaluators 103
3.3.3 Test Data Generators 107
Catalog Listing of Test Data Generators 114
3.3.4 Program Instrumenters 118
Catalog Listing of Program Instrumenters 125
3.3.5 Mutation Testing Tools 134
Catalog Listing of Mutation Testing Tools 140
3.4 Test Supporting Tools 143
3.4.1 Automatic Test Drivers 143
Catalog Listing of Automatic Test Drivers 146
3.4.2 Comparators 148
Catalog Listing of Comparators 149

Part II, Current Defense Practices Overview

Chapter 4
Overview and Data Gathering Procedure 152

Chapter 5
Current Defense Practices 154

An overview of organizations and the technology they use

5.1 Military Headquarters and Development Commands 154

A review of how the Military Services implement the requirements of test and evaluation of software and how that process can be improved

5.2 Program and Project Management Commands 159

A case study of seven projects to determine how the military organizations responsible for the development of the system, planned, managed, and conducted the test and evaluation of software on those projects

5.2.1 Overview of Projects and Development Process 159
5.2.2 Analysis Techniques 169
5.2.3 Testing and Measurement 171
5.2.4 Risk Reduction Activities 177
5.2.5 General Comments -- Lessons Learned 182

5.3 Independent Test Organizations 184

A review of the Military Service's operational test agencies' involvement in the test and evaluation of software of weapon systems

5.4 Development Organizations: Private Contractors, Vendors, and Other Software Suppliers 194

The state of the practice of software development as implemented by twelve contractors

5.4.1 Applications and End-User Software 195
5.4.1.1 Overview of Development Process 195
5.4.1.2 Analysis Techniques 200
5.4.1.3 Testing and Measurement 203
5.4.1.4 Risk Reduction Activities 208
5.4.1.5 New Technology Trends and Lessons Learned 211
5.4.2 Support and System Software 220
5.4.2.1 Overview of Development Process 220
5.4.2.2 Analysis Techniques 222
5.4.2.3 Testing and Measurement 223
5.4.2.4 Risk Reduction Activities 225
5.4.2.5 New Technology Trends and Lessons Learned 226

5.5 Independent Verification and Validation Organizations 232

IV&V contractors discuss how they are conducting IV&V on four projects

5.5.1 Overview of Development Process 232
5.5.2 Analysis Techniques 234
5.5.3 Testing and Measurement 236
5.5.4 Risk Reduction Activities 238
5.5.5 New Technology Trends and Lessons Learned 239

Chapter 6
Policy and Standards 245

Case Studies from the Department of Defense

Overview 245
6.1 Department of Defense Policy 245
6.1.1 Major Systems Acquisitions (DoD Directive 5000.1 and DoD Instruction 5000.2) 245
6.1.2 Test and Evaluation (DoD Directive 5000.3) 249
6.1.3 Management of Computer Resources in Major Defense Systems (DoD Directive 5000.29) 252
6.1.4 Major Automated Information Systems Approval Process (DoD Instruction 7920.2) 253
6 2 Standards used in Contracts 254
6.2.1 Military Standards 254
6.2.2 Data Items 262
6.2.3 Nuclear Safety Considerations 264
6.2.4 The Software Development Standards (SDS) Package 265
6.3 Regulations and Standards of the Armed Forces 272
6.3.1 Air Force Regulations 272
6.3.2 Army Regulations 280
6.3.3 Navy Regulations and Standards 294
6.4 Additional Useful Resources 302
6.4.1 DoD Acquisition Program 302
6.4.2 Strategy for a DOD Software Initiative 307
6.4.3 Embedded Computer Resources and the DSARC Process 311
6.4.4 Proceedings of the Joint Logistics Commanders Policy Coordinating Group on Computer Resource Management 316
6.4.5 Report of the Army Science Board ad hoc Subgroup on Testing of Electronic Systems 323
6.4.6 Air Force Electronics Systems Division Guidebooks for Software Management 327
6.4.7 Air Force Space Division Management Guide for Independent Verification and Validation 348
6.4.8 Air Force Space Division Guide to Management of Embedded Computer Resources 350

Appendix A, Information Sources for Testing Tools 352

Appendix B, The Testing Tools Index 354

B.1 Alphabetical Listing of Cataloged Tools 354
B.2 Testing Tool Data Sheets 357

Appendix C, Data Gathering Guides 421

Appendix D, Comprehensive Bibliography 458

This appendix contains a comprehensive listing of books, survey articles and detailed articles compiled from the open literature

Theory of Testing 459
Software Testing 467
Testing Strategies 467
Testing Techniques 476
Static Analysis Techniques 476
Symbolic Testing 478
Program Instrumentation 480
Program Mutation Testing 483
Input Space Partitioning 485
Functional Program Testing 487
Algebraic Program Testing 488
Random Testing 489
Grammar Based Testing 490
Data-Flow Guided Testing 491
Compiler Testing 492
Real-Time Software and Testing 494
Other Strategies for Constructing Reliable Software 499
Comparative Evaluation of Testing Techniques 506
Testing and Evaluation Tools 507
General 507
Static Analysis Tools 510
Dynamic Analysis Tools 515
Test Supporting Tools 526

Appendix E, Bibliography of DoD Standards, Regulations and Guidance 531

Part I

State-of-the-Art Overview

Chapter 1

Definitions and Theory of Testing

1.1 Program Specification and Correctness

When asked to give a reason for testing a computer program, typical programmers respond, "To see if it works." In practice, the notion of a "working" program is a complex one which takes into account not only the technical requirements of the programming task but also economics, maintainability, ease of interface to other systems and many other less easily quantifiable program characteristics. As these characteristics become more complex, testing to see if a particular piece of software has those characteristics becomes more difficult. The technical literature on program testing tends to deal with "working" in one simplified disguise: *correctness*.

For most of this overview, we will consider the following (simplified) model of the program development cycle (see Figure 1.1).

At the start of the programming task, the programmer is supplied with a *specification* of the program. The specification may be as formal as a document which details the intended behavior of the program in all possible circumstances or it may be as informal as a few instances of what the program is intended to do. In practice, the programmer has available to him several sources of information which comprise the specification. These may include a formal specification document, a working prototype, instances of program behavior, and a priori knowledge about similar software. All of these sources contribute to the programmer's understanding of the task.

Working from this specification, the programmer develops the software. The test -- or, more generally, validation -- of the software lies in the comparison of the software product with the specification of intended behavior.

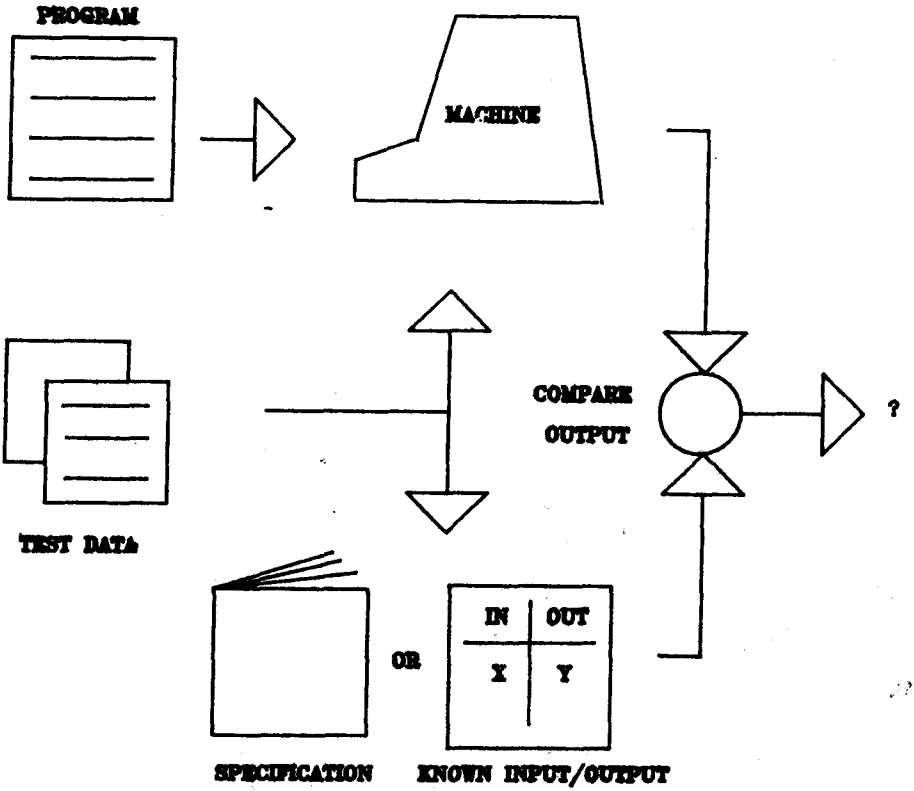


Figure 1.1. Testing for Correctness

For most of this overview, we will not be concerned with the exact nature of specifications. The examples we give will be small and understandable. For instance, the specification of a sorting program might be the following:

INPUT: up to 10,000 input records in the format (KEY1, KEY2, VALUE).

OUTPUT: a reordering of the input records with the following properties:

(1) the primary (KEY1) keys should appear in ascending order,

(2) if two records $R1$ and $R2$ have equal primary keys and if $R1$ precedes $R2$ in the input sequence, then $R1$ precedes $R2$ in the output.

Additional information could have been added to this specification. It could be required, for example, that the sorting program satisfy some performance criteria or that some standard interface conventions be followed.

Practical situations hardly ever give rise to such "clean" specifications. Much research has been devoted to the problem of specifying large and complex software systems [see references 1,3,6,8,12,15]. For our discussion of software testing research, we will not need to be more precise about the nature of program specification.

A specification provides a description of the input data for the program. This input data is called the *domain* of the program and is usually represented by D. The specification also provides a description of the intended behavior of the program on D. We will represent the intended behavior on input data d by $f(d)$. In practice, $f(d)$ may be quite complex.

Similarly, a program P represents some computational actions which will take place when the program is supplied with input data. Even though these actions may be quite complex, we will simplify things considerably by representing the behavior of program P by P^*. Thus, $P^*(d)$ is a mathematical idealization of the behavior of program P on data item d.

The shorthand notation $f(D)$ and $P^*(D)$ is used to represent the intended behavior on all input data and the behavior of P on all input data, respectively.

The program P is said to be *correct* with respect to a specification f if $f(D) = P^*(D)$, that is, if P's behavior matches the intended behavior on all input data.

A problem arises in practical applications of the mathematical theory. How is it possible to determine whether or not $f(d) = P^*(d)$ for some particular datum d in the domain? If the specification document is completely formal, then it should also answer this question. However, specification documents are hardly ever completely formalized (even when they are, determining $f(d)$ may be infeasible). When the specification is not formalized, $f(d)$ may be obtained by hand calculation, textbook requirements or by the application of estimates obtained from