

# **Software Testing Fundamentals**

**Methods and Metrics**

Marnie L. Hutcheson



Wiley Publishing, Inc.



# **Software Testing Fundamentals**

**Methods and Metrics**

Marnie L. Hutcheson



Wiley Publishing, Inc.

**Executive Publisher:** Robert Ipsen  
**Senior Editor:** Ben Ryan  
**Assistant Developmental Editor:** Scott Amerman  
**Editorial Manager:** Kathryn A. Malm  
**Assistant Managing Editor:** Vincent Kunkemueller  
**Text Design & Composition:** Wiley Composition Services

This book is printed on acid-free paper. ☺

Copyright © 2003 by Wiley Publishing Inc., Indianapolis, Indiana. All rights reserved.

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) 646-8700. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4447, E-mail: permcoordinator@wiley.com.

**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 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.

**Trademarks:** Wiley, the Wiley Publishing logo and related trade dress are trademarks or registered trademarks of Wiley Publishing, Inc., in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

*Library of Congress Cataloging-in-Publication Data:*

ISBN: 0-471-43020-X

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

<b>Acknowledgments</b>	<b>xiii</b>
<b>Introduction</b>	<b>xvii</b>
<b>Chapter 1 The State of Software Testing Today</b>	<b>1</b>
A Quick Look at How We Got Where We Are	4
The 1980s: The Big Blue and Big Iron Ruled	4
The 1990s: PCs Begin to Bring Computing to “Every Desktop”	5
Enter: The Web	7
The Current Financial Climate	8
Slowing Down	9
Getting More from What We Have	9
Guess What? The Best Methods Haven’t Changed	12
What the Auditors Want to Know from the Testers	12
The Challenges a Tester Faces Today	12
No Specification Means No Testing	12
Being First to Market: Market/Entrepreneurial Pressures Not to Test	14
The Lack of Trained Testers	15
Standards Reduce the Amount of Testing Required	17
Summary	20
<b>Chapter 2 Maintaining Quality Assurance in Today’s Software Testing Environment</b>	<b>23</b>
What’s Wrong with This Picture?	24
Problems with Traditional Quality Assurance	25
Traditional Definitions of Quality That Are Not Applicable	26
Some Faulty Assumptions	28
What’s Not Working: The Traditional Tools for Controlling Quality	32
Traditional QA and Testing Tools Can’t Keep Up	33
The Paper Problem	33
Solution: Improving the Quality Process	34
Picking the Correct Components for Quality in Your Environment	35
A Change in the Balance	35
Elimination by Market Forces of Competitors Who Fail to Provide Sufficient Quality	36

Picking the Correct Quality Control Tools for Your Environment	36
Automating Record Keeping	37
Improving Documentation Techniques	38
Trained Testers Using Methods and Metrics That Work	42
Summary	43
<b>Chapter 3 Approaches to Managing Software Testing</b>	<b>45</b>
The I-Feel-Lucky Approach to Software Development	45
Some Myths about Art, Science, and Software	47
Myths about Art	48
Myths about Science	49
Myths about Software	49
The Engineering Approach	51
Accountability and Performance	51
Stating the Methods Followed and Why	52
Stating Your Assumptions	52
Apply Adequate Factors of Safety	55
How to Determine a Factor of Safety	56
Always Get a Second Opinion	57
The Adversarial Approach versus the Team Approach	57
The Eclectic Approach to Integration and Testing:	
Balancing Art and Engineering	60
The Top-Down Approach versus the Bottom-Up Approach	65
Current Testing Strategies	66
Top-Down Broad-Focused Integration Testing	67
Organizational Strategies for Locating the Test Group	68
Have an Independent Test Group under Their Own Management	68
Put the Test Group in Development	68
Don't Have a Test Group At All	70
Put the Test Group in Operations	71
The Best Approach	72
Summary	73
<b>Chapter 4 The Most Important Tests (MITs) Method</b>	<b>77</b>
Overview of MITs	78
What MITs Does	78
How MITs Works	79
How to Succeed with MITs	81
Methods That Are Most Useful and Easiest to Implement	81

The Steps for the MITs Method	83
Complimenting Your Development Methodologies	84
MITs for a Plan-Driven Test Effort	86
MITs for a Structured RAD/Agile Test Effort	87
MITs for a Free-for-All RAD/Agile Test Effort	89
Integrating Projects with Multiple Development Methodologies	90
Real-World Example: Heavyweight, Middleweight, and Lightweight Development Efforts	90
Overview of the Project	91
The Scheduling System	92
The Billing System	93
The Document Generation System	96
Integrating the Entire System	100
Summary	102
<b>Chapter 5 Fundamental Metrics for Software Testing</b>	<b>105</b>
Measures and Metrics	106
Craft: The Link between Art and Engineering	107
Operational Definitions: Fundamental Metrics	107
How Metrics Develop and Gain Acceptance	108
What to Measure in Software Testing	108
Fundamental Testing Metrics: How Big Is It?	109
Time	110
The Cost of Testing	111
Tests	111
Bugs	112
Severity	112
Bug Type Classification	113
The Number of Bugs Found	113
Bug Density per Unit: Where Are the Bugs Being Found?	115
Bug Composition: How Many of the Bugs Are Serious?	116
The Bug Fix Rate: How Many of the Bugs That Were Found Were Fixed?	117
Metrics to Measure the Test Effort	119
Test Coverage: How Much of It Was Tested?	119
Test Effectiveness: How Good Were the Tests?	120
Metrics to Track Testing: Will It Be Ready on Time?	122
How Much Did It Cost to Test It?	122
Metrics to Measure Performance: Was It Worth It?	122
The Cost of Not Testing: How Much Did Testing Save?	124
Other Contributions of the Test Effort	125
Summary	125

<b>Chapter 6</b>	<b>The Test Inventory</b>	<b>127</b>
	The Goal: A Successful Test Effort	128
	Planning for Adequate Testing: How Much Is Enough?	129
	Planning for an Adequate Test Effort	130
	Determining the Scope of Testing: How Big Is <i>It</i> ?	132
	Test Units	133
	One Test Script: Many Types of Tests	134
	Using Historical Data in Estimating Effort	135
	Reasons for Using an Inventory	139
	The Sources of the Tests on the Inventory	141
	Tests Based on Requirements	142
	Analytical Methods	142
	Inspections, Reviews, and Walk-throughs	143
	Path Analysis	144
	Data Analysis	144
	Environment Catalog	145
	Usage Statistics: User Profiles	145
	Nonanalytical Methods	147
	Brainstorming Sessions	147
	Expert Testers	148
	Summary	149
 <b>Chapter 7</b>	 <b>How to Build a Test Inventory</b>	 <b>151</b>
	Starting with the Requirements	151
	Sample Inventory from the Structured RAD Project	152
	The Billing System Preliminary Inventory	153
	On the Project Web Site	155
	Preliminary Inventory from a Web Project	158
	Bug Fix Information	160
	New Function (See Req. D071 and D072)	160
	Structural Information	160
	Application Base Function	160
	Environment Catalog	160
	Identifying and Listing the Test Environments	160
	Testing a Large System	162
	Testing Multiple Environments	163
	Adding Detail and Process Layers	164
	The Interviews: What and How to Prepare	166
	Why Two Sets of Interviews?	166
	My Outline for High-Level Interviews	168
	My Outline for Mid-Level Interviews	169
	Example Questions from the Real-World Example	170
	Conducting the Interviews	171
	How to Conduct the Interviews	172
	Instructions for Interviewers	172



Analyzing the Results: Lessons Learned	173
Being Ready to Act on What You Learn	174
What Was Learned in the Real-World Example	175
What Was Learned in the Lightweight Project	178
Summary	179
For Further Exploration	180
Product A	180
Product B	181
<b>Chapter 8 Tools to Automate the Test Inventory</b>	<b>183</b>
The Evolving Test Inventory	186
The Process in Brief	187
Inventory in PowerPoint	188
Inventory in a Document	189
The Hood and What's Under It	190
Inventory in Outline View	190
Inventory in a Table of Contents	193
Inventory in a Table	196
Inventory in a Spreadsheet	198
Inventory in a High-Function Web Site	200
Summary	204
<b>Chapter 9 Risk Analysis</b>	<b>205</b>
The Risk Analysis Process in Brief	206
Benefits of Risk Analysis	208
Ensuring Correct Focus for the Test Effort	208
Publishing Testers' Assumptions	209
Promoting Reuse by Other Groups	209
Providing a Powerful Project Management Tool for the Test Effort	211
How the Inventory Can Be Used	211
Building the Inventory on-the-Fly	212
Project Management Strategies and Risk	213
Examples from the Real World	214
Real-World Money for Information	214
Real-World Money for Flexibility	215
MITs Risk Analysis	217
Quantitative and Qualitative Analysis	217
The MITs Rank Index	219
Weighting an Index	219
The Art of Assigning a Rank Index	220
The MITs Ranking Criteria	221
Identifying Ranking Criteria	221
Assigning a Risk Index to Risk Criteria	222
Combining the Risk Factors to Get a Relative Index	223
Using Risk Ranking in Forensics	225

Case Study: The 401(k) Web Project	226
The Application	226
My Testing	226
Reporting My Results	227
Calculating the Value of Testing	228
Some Thoughts on This Project	229
Summary	230
<b>Chapter 10 Applied Risk Analysis</b>	<b>233</b>
Applying Risk Analysis to the Inventory	235
The Test Estimation Process	236
Where Do the Tests Come From?	237
The MITs Totals Worksheet	239
MITs by the Rank Average Method	239
MITs by the Summation Method	241
Risk-Based Test Coverage and Performance	241
The Sizing Worksheet	242
Item 1: MITs Tests and Coverage	243
Item 2: Test Units and Time to Create Tests	245
Item 3: Time to Run Tests and Create Automated Tests	246
Item 4: Estimate the Number of Errors That Will Be Found	246
Item 5: Code Turnovers, Test Cycles	246
Item 6: Test Environments and Total Tests	247
Item 7: Planning Time	247
Item 8: The Case for Test Automation	247
Item 9: Administration, Documentation, and Logging Time	248
Item 10: Factor of Safety	248
Item 11: Constraints, Assumptions, and Status	248
Negotiating the Test Effort	249
Don't Forget the Developers to Fix the Bugs	250
The eXtreme Project	252
The Contract to Test	252
Risk Criteria for Picking Tests	253
Validating and Verifying the System Requirements	253
Exercising the Most Important Functions in the System	254
Exercising the Most Important Paths and Data	254
Giving the Most Bang for the Buck	254
Summary	255
<b>Chapter 11 Path Analysis</b>	<b>257</b>
The Legend of the Maze of the Minotaur	257
Linear Independence	263
The Case Statement	264
Linear Independence	265
Statement Coverage	267
Branch Coverage	267
A Series of Required Decisions	268

Calculating the Number of Paths through a System	272
The Total Is Equal to the Sum of the Parts	273
What Is a Logic Flow Map?	274
The Elements of Logic Flow Mapping	274
The Rules for Logic Flow Mapping	275
The Equations Used to Calculate Paths	277
Summary	280
Path Analysis Samples for Further Exploration	282
Exercise 1	282
Exercise 2	283
Exercise 3	284
Exercise 4	285
<b>Chapter 12 Applied Path Analysis</b>	<b>287</b>
Correcting Problems with Depth Perception: Or How This All Got Started	287
The Game Show Analogy	289
Getting a Map	290
Divide and Conquer	294
The Paths Not Counted	295
The Price We Pay for Using Divide and Conquer	295
Building Path Test Estimates Quickly	296
Step 1: Building a Menu Map of the System	297
Step 2: Counting All the Menu Options in the System	297
Step 3: Counting the Exceptions for Each Menu Option	301
Sample Test Inventory 2	301
Step 4: Building the Logic Flow Maps for the System	302
Some Procedural Notes on Interviews	303
Back to Work: Some Things to Remember	305
The First Logic Flow Maps	306
Step 5: Recording the Paths	308
The Menu Map	309
Path and Data Dependencies	310
Summary	312
<b>Chapter 13 Data Analysis Techniques</b>	<b>313</b>
Testing Data Input by Users (the GUI)	314
Data-Dependent Paths	314
Some Thoughts about Error Messages	315
Boundary Value Analysis	316
Boundary Value Analysis Example	316
Field Validation Tests	319
Matrix Data Reduction Techniques	321
Data Set Tests	322
Data Set Truth Table	323
The Process for Building Sufficient Data Sets from the Bottom Up	326

Caveats	329
Hidden Boundaries	329
What Was Really Going On	330
Web Services	330
Summary	332
<b>Chapter 14 What Our Analysis Tells Us and What's in Store in the Near Future</b>	<b>335</b>
Testing: The Next Generation	338
Collaboration	338
Mobile Computing	339
MITs Methods and the Next Generation	340
The Challenges	341
<b>Appendix A Answers to Exercises</b>	<b>345</b>
Chapter 7: How to Build a Test Inventory	345
Chapter 11: Path Analysis	346
Exercise 1	346
Exercise 2	347
Exercise 3	350
Exercise 4	351
<b>Appendix B Software Testing Questionnaire, Survey, and Results</b>	<b>355</b>
Making Your Benchmark	356
The Test Questionnaire	356
The State of Knowledge in Commercial Software Testing	360
The Written Survey	360
The Voice Survey	360
Results of the Seminar Evaluations	361
Easiest to Implement	361
More Difficult to Implement	361
Results of the Written Test Survey	361
Composition of the Respondents	362
Education	363
Test Activities	363
Types of Metrics Used	364
Types of Formal Methods Used	364
Test Automation	364
Knowledge of Common Test Terms	365
Recognized Survey Deficiencies	365
Results of the Voice Survey	365
Conclusions Drawn from the Surveys	367

<b>Appendix C Test Collateral Samples and Templates</b>	<b>369</b>
Sample Memo to Describe the Interview Process	369
High-Level Interviews (Duration 15 to 30 minutes)	370
Goals	370
Mid-Level Interviews (Duration 30 to 60 minutes)	371
Goals	371
Sample Project Inventory and Test	
Questionnaire for the Interviews	372
Preliminary Test Inventory	372
Test Questionnaire	373
<b>References</b>	<b>375</b>
<b>Glossary</b>	<b>379</b>
<b>Index</b>	<b>391</b>



When a subject is well understood, it can be explained in a few words, but the road to that understanding can be a long one, indeed. Every human being since the beginning of time has understood the effects of gravity—you trip, you fall down. Yet Sir Isaac Newton explained this phenomenon briefly and accurately only recently on the human time scale.

I have been working on developing the material in this book and circulating it back to testers and their management for many years. Most of the methods and techniques presented here are simple, but good answers don't have to be difficult. Many of these methods are about as old and patentable as oatmeal; others are new. Many of these methods have been discussed and debated for months and even years with colleagues.

The first four chapters of the original version of this book have been available online for four years. In that time the number of readers has risen steadily; presently about 350 visitors read these chapters each week. More than 2,500 software testers and managers in various industries reviewed this work and provided feedback on it. To all those who took part in those discussions, asked questions, picked nits, or came back and reported which steps they tried to implement and how it worked, I want to say again, "Thank you. And don't stop the feedback; that's how we improve our knowledge."

Thanks to Minna Beissinger, Joseph Mueller, Eric Mink, Beate Kim, Joan Rothman, L. Gary Nakashian, Joy Nemitz, Adrian Craig Wheeler, Lawrence Holland, John Chernievsky, Boris Beizer, Dorothy Graham, Roger Sherman, Greg Daich, and Tom Gilb.

I want to express my special thanks to my technical reviewer, researcher, and product developer, David Mayberry. This work would probably not exist except for his patience and continuing efforts. I also want to thank my editors at Wiley; Ben Ryan and Scott Amerman,

Vincent Kunkemueller, and all the others who were so patient and supportive as they added their talents to the making of this work. And finally, many thanks to my research assistants, and my artistic staff, Velvalee Boyd and Dane Boyd.



**Marnie Hutcheson** creates technical courseware for Microsoft Corporation and travels around the world training the trainers who teach these technologies to the world. She is an internationally published author and speaker in the areas of software development, testing and quality assurance, and systems administration.

She began her career in engineering at Prodigy Services Company in 1987 as the Lead Systems Integrator for Shopping, and later Banking and Financial Services. Over the years, she has become a leader in the development of the Web and has helped corporations like GTE and Microsoft develop and launch several major Internet technologies.

Prior to that, she was a student of the performing arts for over 25 years. She performed on stage, in classical ballet, musical theater, folk singing, and opera for 10 years in Denver, Montreal, Boston, and New York.

She also taught dance in institutions and colleges during those years. In the late 1970s, she was the dance instructor and assistant choreographer to U.S. Olympic skating coach, Don Laws, and Olympic choreographer Riki Harris. She worked with U.S. Olympians, elite skaters, and gymnasts in the United States and Canada.



I live in a software development world where product development is not an orderly consistent march toward a tangible goal. “The Project Plan” usually consists of a laundry list of functions dropped off by somebody from marketing. Management embellishes “The Plan” with start and end dates that are of highly questionable origins and totally unreachable. The design and implementation of the product are clandestinely guarded by developers. The product routinely arrives in test virtually unannounced and several weeks late. The tester has not finished the test plan because no one is quite sure what the thing does. The only sure thing is the product must ship on time, next week.

That is software development—chaotic and harried. This book is dedicated to the proposition that this development system is primitive and enormously wasteful. This book presents several methods that provide better ways to perform the business of understanding, controlling, and delivering the right product to the market on time. These methods, taken singly or in groups, provide large cost savings and better-quality products for software developers.

I am a practitioner. I work where the rubber meets the road. I am often present when the user puts their hands on the product for the first time. I deal with real solutions to real problems. I also deal with the frustration of both the customers (who are losing money because the product is failing in some way) and the front-line support people. Front-line support is typically caught in the middle between development groups, who have other priorities, and the customer, who needs the system fixed “right now.”

I work with the developer, whose job is to write good code. Developers do not have time to fill out all those forms quality assurance wants, or to compose an operations document that the test and support groups need. I work with the testers, who really don’t know what’s going on back there in the system. They keep breaking it, but they can’t reproduce the

problems for development. And I work with the document writers, who can't understand why the entire user interface changed just two weeks before the end of the test cycle.

My role is to prevent failures and enhance productivity through automation and process optimization. I work primarily on applications running in large networks. These systems are huge and contain a variety of components that need to be tested. Typically, there are object-oriented modules and graphical user interfaces (GUIs), and browser-based interfaces. These applications typically interact with databases, communications networks, specialized servers, and embedded code, driving specialized hardware—and all of them need to be tested. The methods in this book are distilled from experiences, both failures and successes, with projects that have touched all of these areas.

This is also a work about “how to solve problems,” so it is rich with commentary on human factors. Systems are designed, written, integrated, tested, deployed, and supported by human beings, for human beings. We cannot ignore the fact that human factors play a major role in virtually all system failures.

## What This Book Is About

---

This book is a software tester's guide to managing the software test effort. This is not a formula book of test techniques, though some powerful test techniques are presented. This book is about defensible test methods. It offers methods and metrics that improve the test effort, whether or not formal test techniques are used. It is about how to use metrics in the test effort. There is no incentive to take measurements if you don't know how to use the results to help your case, or if those results might be turned against you. This book shows how to use measurement to discover, to communicate those discoveries to others, and to make improvements.

Some time back I was presenting an overview of these methods at a conference. Part of the presentation was a case study. After these methods were applied, a test inventory was built, and the risk analysis was performed for the system, it was determined within this case study that the optimal test coverage given the time and resources allowed was 67 percent coverage of the entire test inventory.

During the question-and-answer session that followed my presentation, a very distinguished and tall fellow practitioner (he stands well over six feet) said, "Excuse me for mentioning this, but it strikes me that you are a very small person. I was wondering where you find the courage to tell your managing director that you only plan to test 67 percent of the system?"

My answer: "It is true that I am only 5'6", but I am big on the truth. If management wants to give me enough time and resources to test every item on the inventory, I will be happy to do so. But if they want me to do with less than that, I am not going to soft sell the fact that they will get less than 100 percent test coverage. If there isn't time and resources to test everything, then I want to be sure that the tests conducted are the *most important tests*."

I am also going to tell management how good that selection of tests was, how many bugs the test effort found, how serious they were and how much it cost to find them, and if possible, how much was saved because we found and removed them. I will measure the performance of the test effort and be able to show at any time whether we are on schedule or not, if the error densities are too high, or if the bug-fix rate is too low. If we cannot stay on schedule, I can give management the high-quality information it needs to do what it does best, specifically, manage the situation.

## Industry and Technology Trends: Why I Think It's Time to Publish This Book

---

I began developing these methods in the late 1980s when I worked at Prodigy. They evolved to suit the needs of a fast-paced development environment feeding a large, complex real-time system. They were called the Most Important Tests method, or MITs. MITs quickly became the standard for testing methods at Prodigy, and I began writing and publishing case studies of MITs projects in 1990. I took MITs with me when I left Prodigy in 1993, and it continued to evolve as I tackled more and more testing projects in other industries. I spent most of the last 10 years helping businesses embrace and profit from integrating large systems and the Internet.

The (PowerPoint-based) syllabus that I developed to teach MITs since 1993 is based on the first seven chapters that I wrote for the original book, *Software Testing Methods and Metrics*. The emphasis then was on client/server testing, not the Internet, and that is reflected in the original chapters.

First offered in 1993, the course has been taught several times each year ever since. I put the original first four chapters on my Web site in 1997. The number of people reading these four chapters has increased steadily over the years. This year some 17,000 visitors have downloaded these chapters. The most popular is Chapter 2, “Fundamental Methods.” Because of its popularity and the many e-mail discussions it has sparked, it has been expanded here into Chapter 3: “Approaches to Managing Software Testing,” and Chapter 4: “The Most Important Tests (MITs) Method.”

## Changing Times

---

I spent most of 2001 working on Microsoft’s .NET developer training materials, and in the process, I became very familiar with most of the facets of .NET. Bottom line is, the new .NET architecture, with its unified libraries, its “all languages are equal” attitude about development languages, and its enablement of copy-and-run applications and Web services brings us back to the way we did things in the early 1990s—those heady days I spent at Prodigy. The big and exciting difference is that Prodigy was small and proprietary; .NET will be global and public (as well as private).

It will be two years before we really begin to see the global effects of this latest shortening of the software development cycle. Literally, anyone can deploy and sell software as a Web service on global scale, without ever burning a CD, or writing a manual, or paying for an ad in a magazine. And at some point, that software will be tested.

The picture becomes even more interesting when you consider that we are now just beginning the next wave of Internet evolution; the mobile Internet. Just as the PC revolutionized the way we do business today, the smart phones and pocket PCs will allow more people than ever before to access dynamic applications on small screens, with tenuous data links. The methods in this book evolved in just such an environment, and were successful.

Software testing must show that it adds value, and that it is necessary for product success. Otherwise, market forces will encourage competitive shops to forego testing and give the product to the users as fast as they can write it and copy it to the server.

This book is about fundamentals, and fortunately, “fundamental” concepts, while sometimes out of style, evolve very slowly. The examples in the original Prodigy work were out of style in the client/server days; they are very much back in style in the .NET world. In many ways, revising this work to be current today is actually taking it back to its original state.

## Scope of This Book and Who Will Find It Useful

---

This book is a field guide aimed squarely at testers and management involved with developing software systems and applications. It contains practical solutions, not theory. Theory and background are presented only to support the practical solutions.

This is a tester’s survival guide because it helps testers supply answers that management understands and respects. Testers need to answer the question, “Why can’t you have it tested by next week?” This work is also a manager’s survival guide because managers have to explain why things are the way they are, how much it’s going to cost, and why.

The methods presented here were developed in large networks. Often these networks are running a combination of Web-based and client/server-based applications, some on the public Internet and some running privately behind the firewall. These systems are generally written at least in part using object-oriented languages, and all are accessed using graphical user interfaces, be they dedicated clients or Web pages running in a browser. These test methods have been used to test a rich variety of other software systems as well, including telecommunications, business applications, embedded firmware, and game software.

The process described in this book is a *top-down approach* to testing. These methods can be used to test at the unit level, but they are more useful in integration, system, and end-to-end test efforts. These test methods are often used later in the project life cycle, in load testing, performance testing, and production system monitoring. Opportunities for automation or test reuse are noted as appropriate.

Last, this is not an all-or-none test guide. A process improvement can result from implementing parts of these methods, like adding a metric to test tracking, or prioritizing tests and keeping track of how long each one takes to run.

## How This Book Is Organized

---

The chapters have been organized to parallel the process flow in most software development and test efforts. Successive chapters tend to build on what came in the chapters before; so jumping right into the middle of the book may not be a good idea. It is best to proceed sequentially.

Case studies, notes on automation, test techniques, usability issues, and human factors appear throughout the text. It is broken into three main blocks:

**Chapters 1 to 5** concentrate on background and concepts.

**Chapters 6 to 8** focus on the inventory and how to make it.

**Chapters 9 to 14** cover tools and analysis techniques for test estimation, sizing, and planning.

## The Standard for Definitions in This Book

---

The standard for all definitions given in this book is *Webster's New World Dictionary of the American Language* (College Edition, Prentice Hall). However, any good dictionary should be acceptable. When I refer to a definition from some other work, I will cite the work. I have tried to limit such references to works that are readily available to everyone today.

One of the major stumbling blocks I have encountered in educating people involved in developing and testing software is the lack of consensus on the meaning of basic terms. This is richly illustrated in the test survey discussed in Chapter 1 and presented in Appendix B of this book.

The biggest reason for this lack of consensus is that while there are plenty of standards published in this industry, they are not readily available or easy to understand. The second reason for the lack of



consensus is simple disagreement. I have often heard the argument, “That’s fine, but it doesn’t apply here.” It’s usually true. Using the dictionary as the standard solves both of these problems. It is a starting point to which most people have access and can acknowledge. It is also necessary to go back to basics. In my research I am continually confronted with the fact that most people do not know the true meaning of words we use constantly, such as *test*, *verify*, *validate*, *quality*, *performance*, *effectiveness*, *efficiency*, *science*, *art*, and *engineering*.

To those who feel I am taking a step backward with this approach, it is a requirement of human development that we must learn to creep before we can crawl, to crawl before we can walk, and to walk before we can run. The level of mastery that can be achieved in any phase of development is directly dependent on the level of mastery achieved in the previous phase. I will make as few assumptions as possible about my readers’ level of knowledge.

We software developers and testers came to this industry from many directions, many disciplines, and many points of view. Because of this, consensus is difficult. Nevertheless, I believe that our diversity gives us great strength. The interplay of so many ideas constantly sparks invention and innovation. The computer industry is probably home to the largest cooperative inventive undertaking in human history. It simply needs to be managed.



# The State of Software Testing Today

The director asked the tester, “So you tested it? It’s ready to go to production?”

The tester responded, “Yes, I tested it. It’s ready to go.”

The director asked, “Well, what did you test?”

The tester responded, “I tested *it*.”

In this conversation, I was the tester. It was 1987 and I had just completed my first test assignment on a commercial software system.<sup>1</sup> I had spent six months working with and learning from some very good testers. They were very good at finding bugs, nailing them down, and getting development to fix them. But once you got beyond the bug statistics, the testers didn’t seem to have much to go on except *it*. Happily, the director never asked what exactly *it* was.

The experience made me resolve to never again be caught with such a poor answer. I could not always be so lucky as to have management

<sup>1</sup>Commercial software is software that is commercially available and can be purchased by the public. This distinguishes it from safety-critical, proprietary, or military software.

that would accept *it* for an answer. All my training as a structural engineer had prepared me to give my management a much better answer than *it*.

Suppose the supervisor on a building project asked if I tested the steel superstructure on Floor 34 and needed to know if it was safe to build Floor 35. If I said “yes” and if the supervisor then asked, “What did you test?” I would have a whole checklist of answers on the clipboard in my hand. I would have a list with every bolt connection, the patterns of those connections, the specified torque wrench loading used to test the bolts, and the results from every bolt I had touched. I would know exactly which bolts I had touched because each would be marked with fluorescent paint, both on my chart and on the steel.

Why should software testing be any different? I could certainly give my management a better answer than *it*. Many of these better answers were around when the pyramids were being built. When I am asked those questions today, my answer sounds something like this:

*As per our agreement, we have tested 67 percent of the test inventory. The tests we ran represent the most important tests in the inventory as determined by our joint risk analysis. The bug find rates and the severity composition of the bugs we found were within the expected range. Our bug fix rate is 85 percent.*

*It has been three weeks since we found a Severity 1 issue. There are currently no known Severity 1 issues open. Fixes for the last Severity 2 issues were regression-tested and approved a week ago. The testers have conducted some additional testing in a couple of the newer modules. Overall, the system seems to be stable.*

*The load testing has been concluded. The system failed at 90 percent of the design load. The system engineers believe they understand the problem, but they say they will need three months to implement the fix. Projections say the peak load should only be at 75 percent by then. If the actual loading goes above 90 percent, the system will fail.*

*Our recommendation is to ship on schedule, with the understanding that we have an exposure if the system utilization exceeds the projections before we have a chance to install the previously noted fix.*

The thing that I find most amazing is that answers like these are not widely used in the industry today. I regularly hear testers and developers using *it* metrics.

Throughout the 1990s I gave out a survey every time I taught a testing course. Probably 60 percent of the students taking these courses were new to testing, with less than one year as a tester. About 20 percent had from one to five years' experience, and the remainder were expert testers. The survey asked the student to define common testing terms like *test*, and it asked them to identify the methods and metrics that they regularly used as testers. The complete results of these surveys are presented in the appendix of this book. I will mention some of the highlights here.

- The only type of metrics used regularly have to do with counting bugs and ranking them by severity. Only a small percentage of respondents measure the bug find rate or the bug fix rate. No other metrics are widely used in development or testing, even among the best-educated and seemingly most competent testers. It can also be inferred from these results that the companies for which these testers work do not have a tradition of measuring their software development or test processes.
- Few respondents reported using formal methods such as *inspection* or *structured analysis*, meaning some documented structured or systematic method of analyzing the test needs of a system. The most commonly cited reason for attending the seminar was to learn some software testing methods.
- The majority of testers taking the survey (76 percent) had had some experience with automated test tools. Today an even greater percent of testers report that they have used automated test tools, but test automation is also voted as the most difficult test technique to implement and maintain in the test effort.
- The respondents who are not actively testing provided the most accurate definitions of the testing terms. The people performing the testing supplied the poorest definitions of the testing tasks that they are performing most frequently.

Overall, I believe that the quality of the tester's level of awareness is improving, and, certainly, software testing practices have improved in the commercial software development sector. Today there are more publications, tools, and discussion groups available than there were 10 years ago. There are certainly more shops attempting to use formal methods and testing tools. But the survey results haven't changed much over the years.

How did we get to this mind-set? How did these limitations in perceptions—and, unfortunately, all too often in practice—come about? To understand that, we need to examine the evolution of software development and testing during the last two decades.

## A Quick Look at How We Got Where We Are

---

Most of the formal methods and metrics around today had their start back in the 1970s and 1980s when industry began to use computers. Computer professionals of that time were scientists, usually mathematicians and electrical engineers. Their ideas about how to conduct business were based on older, established industries like manufacturing; civil projects like power plants; and military interests like avionics and ballistics.

### The 1980s: The Big Blue and Big Iron Ruled

By the 1980s, computers were widely used in industries that required lots of computation and data processing. Software compilers were empowering a new generation of programmers to write machine-specific programs.

In the 1980s computers were mainframes: big iron. Large corporations like IBM and Honeywell ruled the day. These computers were expensive and long-lived. We expected software to last for five years, and we expected hardware to last even longer—that is, how long it takes to depreciate the investment. As a result, buying decisions were not made lightly. The investments involved were large ones, and commitments were for the long term, so decisions were made only after careful consideration and multiyear projections.



**Fact: Computers in the 1980s: expensive, long-term commitment, lots of technical knowledge required**

Normally a vendor during the 1980s would sell hardware, software, support, education, and consulting. A partnership-style relationship existed between the customer and the vendor. Once a vendor was selected, the company was pretty much stuck with that company until the hardware and software were depreciated; a process that could take 10 or more years.