



# Common System and Software Testing Pitfalls

How to Prevent and Mitigate Them

*Descriptions, Symptoms, Consequences,  
Causes, and Recommendations*

**Donald G. Firesmith**

*Foreword by Capers Jones*

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



## Praise for *Common System and Software Testing Pitfalls*

“Firesmith’s collection of actionable practices for real-world, non-trivial testing and the processes in which they’re applied is comprehensive and uniquely valuable. Nothing published about software testing in recent years provides anything like it.”

—Robert V. Binder, robertvbinder.com

“Don’s compilation of real-world testing problems, symptoms, and solutions is the most comprehensive available. You can use it early in your project to prevent these problems. Or you can use it at the end, as a ready list of costly lessons you could have avoided, had you used it early on. I’m afraid this book’s publication will undermine a lot of excuses for repeating these mistakes.”

—Vince Alcalde, National Australia Bank

“Excellent, Excellent, Excellent! This book should be mandatory reading for anyone involved in product development. Donald’s book addresses the pitfalls that need to be understood and allowed for in all product development verification and validation planning. While the focus of the book is on software projects, most of the pitfalls are equally applicable to any size project that involves both hardware and software.”

—Louis S. Wheatcraft, Requirements Experts Inc.

“The potential impact of this book cannot be overstressed. Software systems that are not adequately tested do not adequately evolve. I highly recommend this book as a must-read for people directly involved in the development and management of software-intensive systems.”

—Dr. Kenneth E. Nidiffer, Director of Strategic Plans for Government Programs,  
Software Engineering Institute, Carnegie Mellon University

“*Common System and Software Testing Pitfalls* identifies realistic testing pitfalls. More importantly, it also identifies solutions for avoiding them on your next project. Every manager should read this book and follow the recommendations.”

—Barry Stanly, Enterprise Technology Alliance

“Whether you are a novice tester or a seasoned professional, you will find this book to be a valuable resource. The information on how to identify and prevent problem areas is clear, concise and, most importantly, actionable.”

—Allison Yeager, Blackbaud

“First of all, this is great material! It contains probably all of the testing problems I have faced in my career and some that I wasn’t aware of. . . . Thank you for the opportunity to read this superb material!”

—Alexandru Cosma, Frequentis

“As a tester, I consider *Common System and Software Testing Pitfalls* by Donald Firesmith to be a must-read book for all testers and QA engineers.”

—Thanh Huynh, LogiGear

“Your book provides very good insight and knowledge. After working in IT for over thirty years, and focusing on software testing the past thirteen years, I still learned more tips and best practices in software testing.”

—Tom Zalewski, Texas State Government

“This book is essential for the people in the cyber security business . . . I can see it becoming a classic. Don has done a great job.”

—Michael Hom, Compass360 Consulting

“Awesome work. Very mature.”

—Alejandro Salado, Kaiser, Threde GmbH

“All in all, a great document.”

—Peter Bolin, Revolution IT Pty Ltd.

---

## **Common System and Software Testing Pitfalls**

# The SEI Series in Software Engineering

Software Engineering Institute of Carnegie Mellon University and Addison-Wesley



Software Engineering Institute | Carnegie Mellon



Visit [informit.com/sei](http://informit.com/sei) for a complete list of available publications.

**T**he SEI Series in Software Engineering is a collaborative undertaking of the Carnegie Mellon Software Engineering Institute (SEI) and Addison-Wesley to develop and publish books on software engineering and related topics. The common goal of the SEI and Addison-Wesley is to provide the most current information on these topics in a form that is easily usable by practitioners and students.

Titles in the series describe frameworks, tools, methods, and technologies designed to help organizations, teams, and individuals improve their technical or management capabilities. Some books describe processes and practices for developing higher-quality software, acquiring programs for complex systems, or delivering services more effectively. Other books focus on software and system architecture and product-line development. Still others, from the SEI's CERT Program, describe technologies and practices needed to manage software and network security risk. These and all titles in the series address critical problems in software engineering for which practical solutions are available.



Make sure to connect with us!  
[informit.com/socialconnect](http://informit.com/socialconnect)



---

# Common System and Software Testing Pitfalls

How to Prevent and Mitigate Them

*Descriptions, Symptoms, Consequences, Causes, and  
Recommendations*

Donald G. Firesmith

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco  
New York • Toronto • Montreal • London • Munich • Paris • Madrid  
Capetown • Sydney • Tokyo • Singapore • Mexico City



The SEI Series in Software Engineering

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [international@pearsoned.com](mailto:international@pearsoned.com).

Visit us on the Web: [informit.com/aw](http://informit.com/aw)

*Library of Congress Cataloging-in-Publication Data*

Firesmith, Donald G., 1952–

Common system and software testing pitfalls : how to prevent and mitigate them : descriptions, symptoms, consequences, causes, and recommendations / Donald G. Firesmith.

pages cm.

Includes bibliographical references and index.

ISBN 978-0-13-374855-0 (pbk. : alk. paper)

1. Computer software—Testing—Automation. 2. Software failures—Prevention. I. Title.

QA76.76.T48F58 2013

005.3028'7—dc23

2013039235

Copyright © 2014 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-13-374855-0

ISBN-10: 0-13-374855-3

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.  
First printing, December 2013

# CONTENTS

FOREWORD .....	xiii
PREFACE .....	xvii
Scope .....	xviii
Intended Audience .....	xviii
How to Use This Book and Its Contents .....	xix
Organization of This Book .....	xx
Acknowledgments .....	xxi
ABOUT THE AUTHOR .....	xxiii
<b>1 OVERVIEW .....</b>	<b>1</b>
1.1 What Is Testing? .....	1
1.2 Testing and the V Models .....	2
1.3 What Is a Defect? .....	5
1.4 Why Is Testing Critical? .....	7
1.5 The Limitations of Testing .....	9
1.6 What Is a Testing Pitfall? .....	10
1.7 Categorizing Pitfalls .....	11
1.8 Pitfall Specifications .....	11

<b>2</b>	<b>BRIEF OVERVIEWS OF THE TESTING PITFALLS</b>	<b>13</b>
2.1	General Testing Pitfalls	13
2.1.1	Test Planning and Scheduling Pitfalls	13
2.1.2	Stakeholder Involvement and Commitment Pitfalls	14
2.1.3	Management-Related Testing Pitfalls	14
2.1.4	Staffing Pitfalls	15
2.1.5	Test-Process Pitfalls	16
2.1.6	Test Tools and Environments Pitfalls	17
2.1.7	Test Communication Pitfalls	18
2.1.8	Requirements-Related Testing Pitfalls	19
2.2	Test-Type-Specific Pitfalls	20
2.2.1	Unit Testing Pitfalls	20
2.2.2	Integration Testing Pitfalls	20
2.2.3	Specialty Engineering Testing Pitfalls	21
2.2.4	System Testing Pitfalls	22
2.2.5	System of Systems (SoS) Testing Pitfalls	22
2.2.6	Regression Testing Pitfalls	23
<b>3</b>	<b>DETAILED DESCRIPTIONS OF THE TESTING PITFALLS</b>	<b>25</b>
3.1	Common Negative Consequences	25
3.2	General Recommendations	26
3.3	General Testing Pitfalls	28
3.3.1	Test Planning and Scheduling Pitfalls	28
	No Separate Test Planning Documentation (GEN-TPS-1)	28
	Incomplete Test Planning (GEN-TPS-2)	31
	Test Plans Ignored (GEN-TPS-3)	35
	Test-Case Documents as Test Plans (GEN-TPS-4)	37
	Inadequate Test Schedule (GEN-TPS-5)	39
	Testing at the End (GEN-TPS-6)	42
3.3.2	Stakeholder Involvement and Commitment Pitfalls	44
	Wrong Testing Mindset (GEN-SIC-1)	44
	Unrealistic Testing Expectations (GEN-SIC-2)	47
	Lack of Stakeholder Commitment to Testing (GEN-SIC-3)	49
3.3.3	Management-Related Testing Pitfalls	51
	Inadequate Test Resources (GEN-MGMT-1)	52
	Inappropriate External Pressures (GEN-MGMT-2)	54
	Inadequate Test-Related Risk Management (GEN-MGMT-3)	57
	Inadequate Test Metrics (GEN-MGMT-4)	59

	Inconvenient Test Results Ignored (GEN-MGMT-5)	61
	Test Lessons Learned Ignored (GEN-MGMT-6)	64
3.3.4	Staffing Pitfalls	65
	Lack of Independence (GEN-STF-1)	66
	Unclear Testing Responsibilities (GEN-STF-2)	68
	Inadequate Testing Expertise (GEN-STF-3)	69
	Developers Responsible for All Testing (GEN-STF-4)	72
	Testers Responsible for All Testing (GEN-STF-5)	74
3.3.5	Test Process Pitfalls	75
	Testing and Engineering Processes Not Integrated (GEN-PRO-1)	76
	One-Size-Fits-All Testing (GEN-PRO-2)	77
	Inadequate Test Prioritization (GEN-PRO-3)	80
	Functionality Testing Overemphasized (GEN-PRO-4)	82
	Black-Box System Testing Overemphasized (GEN-PRO-5)	85
	Black-Box System Testing Underemphasized (GEN-PRO-6)	86
	Too Immature for Testing (GEN-PRO-7)	88
	Inadequate Evaluations of Test Assets (GEN-PRO-8)	90
	Inadequate Maintenance of Test Assets (GEN-PRO-9)	92
	Testing as a Phase (GEN-PRO-10)	94
	Testers Not Involved Early (GEN-PRO-11)	96
	Incomplete Testing (GEN-PRO-12)	98
	No Operational Testing (GEN-PRO-13)	100
	Inadequate Test Data (GEN-PRO-14)	102
	Test-Type Confusion (GEN-PRO-15)	104
3.3.6	Test Tools and Environments Pitfalls	106
	Over-Reliance on Manual Testing (GEN-TTE-1)	106
	Over-Reliance on Testing Tools (GEN-TTE-2)	108
	Too Many Target Platforms (GEN-TTE-3)	110
	Target Platform Difficult to Access (GEN-TTE-4)	112
	Inadequate Test Environments (GEN-TTE-5)	114
	Poor Fidelity of Test Environments (GEN-TTE-6)	118
	Inadequate Test Environment Quality (GEN-TTE-7)	122
	Test Assets Not Delivered (GEN-TTE-8)	124
	Inadequate Test Configuration Management (GEN-TTE-9)	126
	Developers Ignore Testability (GEN-TTE-10)	129
3.3.7	Test Communication Pitfalls	131
	Inadequate Architecture or Design	
	Documentation (GEN-COM-1)	131
	Inadequate Defect Reports (GEN-COM-2)	134

	Inadequate Test Documentation (GEN-COM-3)	136
	Source Documents Not Maintained (GEN-COM-4)	139
	Inadequate Communication Concerning	
	Testing (GEN-COM-5)	140
3.3.8	Requirements-Related Testing Pitfalls	143
	Ambiguous Requirements (GEN-REQ-1)	144
	Obsolete Requirements (GEN-REQ-2)	147
	Missing Requirements (GEN-REQ-3)	150
	Incomplete Requirements (GEN-REQ-4)	152
	Incorrect Requirements (GEN-REQ-5)	154
	Requirements Churn (GEN-REQ-6)	156
	Improperly Derived Requirements (GEN-REQ-7)	159
	Verification Methods Not Properly Specified (GEN-REQ-8)	161
	Lack of Requirements Trace (GEN-REQ-9)	162
3.4	Test-Type-Specific Pitfalls	164
3.4.1	Unit Testing Pitfalls	164
	Testing Does <i>Not</i> Drive Design and	
	Implementation (TTS-UNT-1)	165
	Conflict of Interest (TTS-UNT-2)	167
3.4.2	Integration Testing Pitfalls	169
	Integration Decreases Testability Ignored (TTS-INT-1)	169
	Inadequate Self-Monitoring (TTS-INT-2)	172
	Unavailable Components (TTS-INT-3)	173
	System Testing as Integration Testing (TTS-INT-4)	175
3.4.3	Specialty Engineering Testing Pitfalls	177
	Inadequate Capacity Testing (TTS-SPC-1)	178
	Inadequate Concurrency Testing (TTS-SPC-2)	181
	Inadequate Internationalization Testing (TTS-SPC-3)	183
	Inadequate Interoperability Testing (TTS-SPC-4)	185
	Inadequate Performance Testing (TTS-SPC-5)	188
	Inadequate Reliability Testing (TTS-SPC-6)	190
	Inadequate Robustness Testing (TTS-SPC-7)	193
	Inadequate Safety Testing (TTS-SPC-8)	197
	Inadequate Security Testing (TTS-SPC-9)	200
	Inadequate Usability Testing (TTS-SPC-10)	203
3.4.4	System Testing Pitfalls	206
	Test Hooks Remain (TTS-SYS-1)	206
	Lack of Test Hooks (TTS-SYS-2)	208
	Inadequate End-To-End Testing (TTS-SYS-3)	209

3.4.5	System of Systems (SoS) Testing Pitfalls .....	211
	Inadequate SoS Test Planning (TTS-SoS-1) .....	212
	Unclear SoS Testing Responsibilities (TTS-SoS-2) .....	213
	Inadequate Resources for SoS Testing (TTS-SoS-3) .....	215
	SoS Testing Not Properly Scheduled (TTS-SoS-4) .....	217
	Inadequate SoS Requirements (TTS-SoS-5) .....	219
	Inadequate Support from Individual System	
	Projects (TTS-SoS-6) .....	220
	Inadequate Defect Tracking Across Projects (TTS-SoS-7) .....	222
	Finger-Pointing (TTS-SoS-8) .....	224
3.4.6	Regression Testing Pitfalls .....	225
	Inadequate Regression Test Automation (TTS-REG-1) .....	225
	Regression Testing Not Performed (TTS-REG-2) .....	228
	Inadequate Scope of Regression Testing (TTS-REG-3) .....	231
	Only Low-Level Regression Tests (TTS-REG-4) .....	234
	Test Resources Not Delivered for Maintenance (TTS-REG-5) .....	236
	Only Functional Regression Testing (TTS-REG-6) .....	237
<b>4</b>	<b>CONCLUSION</b> .....	241
	4.1 Future Work .....	241
	4.2 Maintaining the Lists of Pitfalls .....	242
<b>A</b>	<b>GLOSSARY</b> .....	243
<b>B</b>	<b>ACRONYMS</b> .....	253
<b>C</b>	<b>NOTES</b> .....	255
<b>D</b>	<b>REFERENCES</b> .....	269
<b>E</b>	<b>PLANNING CHECKLIST</b> .....	271
	<b>INDEX</b> .....	279

*This page intentionally left blank*

## FOREWORD

As a general rule, about 50 cents out of every dollar spent on software projects goes toward finding and fixing bugs. About 40 cents out of that 50 is spent on various kinds of testing, of which there are more than twenty types in total.

Software testing is a curious part of software engineering. Given that it is the key cost driver for software projects, and that testing costs go up with application size, it is alarming that testing is seldom done well. And yet, the topic is covered by some of the best software engineering books by some of the best authors. A quick search of the web or online bookstores will turn up dozens of books about software testing, and many of these are quite good.

There seem to be some social reasons for why testing is not as sophisticated as it should be. One of these is that not every test team member actually reads any of the books on testing. Another is that tests by certified test personnel who do know how to test well are still outnumbered by amateur testers or developers who may lack training and who may not have read the testing books either. A third reason, addressed by this book, is that many of the older testing books cover only part of the problem of achieving good testing results.

Many testing books are, as might be expected, “how-to-do-it” books that step through a sequence of test planning, test-case design, test-case construction, test execution, defect identification, defect repair, and repair integration. These are all teachable skills, and they should be better known than they are.

Don Firesmith's new book on testing approaches testing from a different perspective. Rather than being another "how-to-do-it" book, this book examines testing from the opposite direction. It explains the errors and failures of testing that he has observed over his long career in software and his work with the Software Engineering Institute (SEI).

This reverse view makes Don's book a natural complement to "how-to-do-it" books. I think that this is the first book on testing that emphasizes what goes wrong and how these problems might be avoided.

In other fields, combining how to do something with avoiding common problems is a standard part of the instruction sequence. For example, when learning to play golf, modern golf schools have video cameras that film golf students during lessons. The golf instructors go through the videos with each student and show them exactly what they did well and what they did poorly. This is an effective training technique. It is actually much harder to stop doing the wrong things than it is to learn to do the right things.

Other fields, such as professional football, also divide training into what has to be done right and what happens when the basics are not done right. This is why coaches and players review films after every game.

Until this new book on software testing, the literature only emphasized doing things well; it was a bit short on what happens if things are not done well.

In this book, a "pitfall" is any action or decision that might lower the effectiveness of testing. Don identifies ninety-two of these pitfalls, which is certainly the largest number I've encountered to date.

This book somewhat resembles a classic medical textbook that defines various illnesses and then discusses the available therapies for them. Each of the ninety-two pitfalls is defined and discussed using a standard format and identical points, which makes the book very easy to follow.

The pitfall descriptions use common headings, such as:

- Descriptions
- Potential Applicability
- Characteristic Symptoms
- Potential Negative Consequences
- Potential Causes
- Recommendations
- Related Pitfalls

This format is similar to a book I wrote some years ago, entitled *Assessment and Control of Software Risks*, published by Prentice Hall. My book used an actual medical text, *Control of Communicable Diseases in Man*, published by the U.S. Public Health Service, as the pattern.

Having worked as a programmer and systems analyst for the U.S. Public Health Service, I was in close contact with medical diagnostic methods, and they seemed to be appropriate for diagnosing software problems.

Some of my topics are very similar to Don's, such as:

- Definition
- Severity
- Susceptibility and Resistance
- Methods of Prevention
- Methods of Control

Applying some of the diagnostic patterns from medical practice to software engineering problems is a useful way to understand serious and common conditions, and Don has taken this idea to a new level for software engineering, and especially so for testing.

Testing is the main form of defect removal for software applications, but it is not the only form. A synergistic combination of defect prevention; pretest removal, such as inspections and static analysis; and formal testing by trained and certified test personnel can approach or exceed 99% in cumulative defect-removal efficiency levels. Even better, these good results come with shorter schedules and lower costs since the main source of software project delay is excessive defects during testing, which stretches out test schedules to several times their planned durations.

I recommend *Common System and Software Testing Pitfalls* for project managers, test personnel, quality-assurance personnel, and software engineers at all levels. All of us in software should know the common problems that we face during testing and how these problems might be avoided or minimized.

Don's book is a very good addition both to the testing literature and to the literature on quality assurance and software engineering. It is likely to become a standard for test training as well as a good reference for professional testers and developers. It would be a good teaching aid for young software engineers and a good handbook for all of us

I would also recommend this book as background material for negotiating outsourced software contracts. I often work as an expert witness in litigation for software with very poor quality, and this book might well reduce or eliminate these lawsuits if it were consulted before contract negotiations.

—Capers Jones, VP and CTO  
Namcook Analytics LLC

*This page intentionally left blank*

## PREFACE

There are numerous good books on systems and software testing, and most testers probably already have several on their shelves. There seems to be no scarcity of how-to books on testing, and they are full of excellent advice on how to test software-reliant systems. They cover test planning, the many different types of tests, how testing fits in to the development cycle, test-case design—including test-case-selection and test-completion criteria—test tools and environments, and many other interesting and useful topics.

Yet we spend a huge amount of time and effort testing, and in spite of this, we deliver systems with dozens if not hundreds of residual defects. In addition to being a tester, I have also taken part in numerous internal as well as independent technical assessments (ITAs) of system and software development projects, including their testing organizations and programs. And in every single case, regardless of whether I was a member of the test team or the assessment team, I have always observed several significant testing problems. More specifically, I have observed testers and other developers falling into the same pitfalls over and over again. Clearly, how-to books—while highly useful—are not sufficient to make testing either efficient or effective.

The frustration I experienced by enduring and observing these commonly occurring testing pitfalls led to this book. If the many how-to books are insufficient by themselves, then clearly it is time to try something different: a *how-not-to* book.

You can think of this book as a catalog and repository of testing anti-patterns: the pitfalls to avoid, how to mitigate their negative consequences if you can't avoid them, and how to escape from them once you've fallen in. Like a naturalist's field guide to wild animals, let this be your guidebook to the dangerous world of testing mistakes and its denizens—the many creative ways people have discovered to botch testing.

---

## Scope

The scope of this book is *testing*, which is only one of several methods commonly used to validate that a system meets its stakeholders' needs and to verify that the system conforms to its specified requirements. Although other such methods (for example, inspections, demonstrations, reviews, analysis, simulation, reuse, and certification) exist and could be documented in a similar manner, they are beyond the scope of this book.

The scope of this book is also the testing of *software-reliant systems*, which often are heterogeneous aggregations of subsystems, hardware, software, data, facilities, material, and personnel. This includes the testing of pure software applications and their components. For simplicity's sake, I will use the term *system* to mean heterogeneous systems, software applications, and their architectural, design, and implementation components.

The pitfalls in this book primarily apply to large and medium-sized projects producing important systems and software applications that require at least a quasi-rigorous testing program and process. The pitfalls do not necessarily apply to very small and simple projects producing relatively trivial systems or software programs that (1) are neither business-, mission-, safety-, nor security-critical; (2) will be used only in-house with close collaboration between stakeholders and developers; (3) will be used once and not maintained; or (4) are prototypes that will not be placed into operation. Such systems often can be adequately tested in a highly informal and ad hoc manner. Some of the pitfalls apply only or primarily to testing systems having significant hardware, and these pitfalls therefore do not (primarily) apply to testing software-only applications.

---

## Intended Audience

This book is written primarily for testers and their technical managers. It is intended to help you recognize and avoid potential testing-related pitfalls.

This book is also written for all stakeholders in system development and sustainment who need a better understanding of what can go wrong with testing, both when preparing for testing and during the actual testing. This includes customer and user representatives, project managers and technical

leaders, requirements engineers, architects and designers, implementers, maintainers, and specialty engineers (such as configuration managers, quality engineers, reliability engineers, and human factors engineers).

Finally, this book is written for testing subject-matter experts, whether academics or consultants, who need a more organized and comprehensive understanding of what can go wrong with testing.

---

## How to Use This Book and Its Contents

The primary goal of this book is to provide the information you need to

- Avoid falling into any of the commonly occurring testing pitfalls
- Recognize when you have already fallen into one or more testing pitfalls
- Escape from these pitfalls while minimizing the resulting negative consequences

This book provides detailed information on the commonly occurring testing pitfalls, and it can be used

- To improve understanding of and communication about commonly occurring testing pitfalls
- As training material for testers and the stakeholders of testing
- As checklists[1]<sup>1</sup> when
  - ♦ Developing and reviewing an organizational or project testing process or strategy
  - ♦ Developing and reviewing test planning documentation, such as:
    - Test and Evaluation Master Plans (TEMPs), System Test Plans (STPs), or Test Strategy Documents (TSDs)
    - The testing sections of planning documents such as the System Engineering Management Plan (SEMP) and the System Development Plan (SDP)
    - Test planning presentations (for example, for training and status reporting)
    - Testing wikis, SharePoint sites, and Application Lifecycle Management (ALM) tool repositories
  - ♦ Evaluating the testing-related parts of contractor proposals
  - ♦ Evaluating test planning documentation, test descriptions, and test results (quality control)
  - ♦ Evaluating the actual as-performed testing process (quality assurance)[2]
  - ♦ Identifying testing risks and appropriate risk-mitigation approaches
- To categorize testing pitfalls for metrics collection, analysis, and reporting

---

1. Notes are identified by a bracketed number ([#]), and are located in Appendix C, Notes.

- To help identify testing areas potentially needing improvement, both during a project and at its conclusion, such as during project postmortems

---

## Organization of This Book

This book is organized as follows:

- **Preface**

This preface begins with a brief introduction to the book, followed by a description of the book's scope and its intended audience. Next, it offers brief advice on how best to use the information provided here. Finally, I acknowledge the book's many technical reviewers, without whom it would not be half as good.

- **Chapter 1: Foundational Concepts**

The first chapter defines the most important concepts in this book: testing, defects, and testing pitfalls. It presents the system-engineering V Models that explain how different types of testing are associated with the project's work products. It addresses why testing is so important as well as explains why it has some significant limitations. Finally, it explains how the testing pitfalls are categorized and documented to make them easier to locate and to understand.

- **Chapter 2: Brief Overviews of the Testing Pitfalls**

The second chapter identifies and summarizes ninety-two commonly occurring testing pitfalls. The purpose of Chapter 2 is to provide a *very* brief, high-level overview of each pitfall, making it easy for readers to search for and identify pitfalls relevant or specific to their situations.

- **Chapter 3: Detailed Descriptions of the Testing Pitfalls**

The third chapter provides detailed descriptions of each of the commonly occurring testing pitfalls. Specifically, it documents each pitfall as follows: its name, description, applicability, characteristic symptoms, potential negative consequences, potential causes, and associated recommendations for avoiding the pitfall or limiting its consequences. Chapter 3 is intended to be used primarily as a handy reference once relevant pitfalls are identified via either the Contents section or Chapter 2. Thus, I suggest that you read this chapter as you would read the patterns in a patterns book: once through rapidly to get a basic understand of the pitfalls, then examine the detailed specifications of individual pitfalls on an as-needed basis.

- **Chapter 4: Conclusion**

The fourth and final chapter provides a holistic summary of the pitfalls before concluding with a brief look at potential future research that might make this categorization of testing pitfalls even more useful.

- **Appendixes**

The appendixes start with a glossary of terms and a list of acronyms. In order to keep the descriptions of the individual pitfalls reasonably short—especially for the experienced tester, who will recognize the majority of these pitfalls—Appendix C provides extensive notes for those who might desire a little extra information. The notes are identified by bracketed numbers [#] throughout the text. The book’s relatively short list of references comes next. The final appendix is a checklist that can be used when planning testing and assessing testing programs and organizations.

---

## **Acknowledgments**

I am grateful for the interest of the more than 350 testers, testing subject-matter experts (SMEs), and academics from forty-six countries who volunteered to review various draft versions of this book. I am extremely grateful to the following individuals who provided excellent review comments and recommendations, often several times, while reviewing different drafts of the manuscript:

Dimpy Adhikary, Amagi Media Labs, India  
 Vince Alcalde, Independent Consultant, Australia  
 Stephanie Benedetti, AIPSO, US  
 Laxmi Bhat, Minerva Networks, US  
 Robert V. Binder, System Verification Associates, US  
 Peter Bolin, Revolution IT Pty Ltd, Australia  
 Michael Bolton, DevelopSense, Canada  
 Paul Carvalho, Software Testing and Quality Services, Canada  
 Alexandru Cosma, Frequentis, Romania  
 John Dannenberg, Compuware Corporation, US  
 Jorge Alberto De Flon, Servicio de Administración Tributaria (SAT), Mexico  
 George Despotou, University of York, UK  
 Lee Eldridge, Independent Consultant, Australia  
 Eliazar Elisha, University of Liverpool, UK  
 Robin Goldsmith, Go Pro Management Inc., US  
 Jon L. Gross, Software Engineering Institute, US  
 Paolo Guolo, Private Consultant, Italy  
 Kobi Halperin, Ceragon Networks, Israel  
 Sam Harbaugh, Integrated Software Inc., US  
 John Hawrylak, Software Engineering Institute, US  
 M. E. Hom, Compass360 Consulting, US  
 Thanh Cong Huynh, LogiGear, Vietnam  
 Capers Jones, Namcook Analytics, US

Ronald Kohl, Independent Consultant, US  
Wido Kunde, Baker Hughes, Germany  
Seth Landsman, The MITRE Corporation, US  
Philippe Lebacqz, Toyota Motors Europe, Belgium  
Stephen Masters, Software Engineering Institute, US  
Ken Nidiffer, Software Engineering Institute, US  
Anne Nieberding, Independent Consultant, US  
William Novak, Software Engineering Institute, US  
Mahesh Palan, Calypso Technology, US  
Dan Pautler, Elekta, US  
David C. Peterson, Protengent, US  
Mark Powel, Attwater Consulting, US  
James Redpath, Sogeti, US  
Sudip Saha, Navigators Software, India  
Alejandro Salado, Kayser—Threde GmbH, Germany  
David Schultz, NASA, US  
Matt Sheranko, Knowledge Code, US  
Oleg Spozito, Independent Consultant, Canada  
Barry Stanly, Independent Consultant, US  
Amit Wertheimer, EMC Corporation, RSA, Israel  
Lou Wheatcraft, Requirements Experts, US  
Kerry Wilkerson, Private Consultant, US  
Allison Yeager, Blackbaud, US  
Thomas Zalewski, Texas State Government, US

Although the vast majority of the comments and recommendations from each of the reviewers made it into this book in one form or another, that does not mean that every reviewer would necessarily agree with everything in it. Further, I naturally take responsibility for any errors that slipped past my diligent reviewers and made it into the final book.

I would like to thank John Foreman, an SEI fellow and member of the management team, who provided the funding and time I needed to finalize the manuscript for publication.

Finally, I would like to thank the acquisition and production teams at Addison-Wesley for their strong support in publishing this book. Deserving special mention are Bernard Goodwin, my acquisitions editor, and Vicki Rowland, who copyedited the manuscript and created a very high level of consistency of both content and format. Both worked closely with me through several iterations of the book's cover and contents, thereby making this the most enjoyable of my books to bring to completion.

## ABOUT THE AUTHOR

**Donald G. Firesmith**, a senior member of the technical staff in the Software Solutions Division at the Software Engineering Institute (SEI), helps the US Department of Defense and other governmental agencies acquire large, complex, software-reliant systems. He is an internationally recognized subject matter expert who has published seven software and systems engineering books in the areas of requirements engineering, architecture engineering, situational method engineering, testing, and object-oriented development. With thirty-five years of industry experience, he has also published dozens of technical articles, spoken at numerous international conferences, and has been the program chair or on the program committee of several software conferences. He has taught several hundred courses in commercial and governmental settings (both civilian and military, as well as numerous tutorials at international conferences. Copies of his numerous papers and presentations are downloadable from his personal website: <http://donald.firesmith.net>.

*This page intentionally left blank*

## OVERVIEW

---

### 1.1 What Is Testing?

*Testing* is the activity of executing a system, subsystem, or component under specific preconditions (for example, pretest mode, states, stored data, and external conditions) with specific inputs so that its actual behavior (outputs and postconditions) can be compared with its required or expected behavior.

Testing differs from other verification and validation methods (for example, analysis, demonstration, and inspection) in that it is a dynamic, as opposed to a static, analysis method that involves the actual execution of the thing being tested.

Testing has the following goals:

- Primary goal:
  - ♦ Enable the system under test (SUT) to be improved by:
    - “Breaking” it (that is, by causing faults and failures)
    - Exposing its defects so that they can be fixed
- Secondary goals:
  - ♦ Provide adequate confidence based on sufficient objective evidence regarding the SUT's:
    - Quality  
A system's quality is not just its lack of defects or its correctness (in terms of meeting its requirements). A system must also have the necessary levels of relevant quality characteristics and attributes; for example, availability, capacity, extensibility, maintainability, performance, portability, reliability, robustness, safety, security, and usability.
    - Fitness for purpose
    - Readiness for shipping, deployment, or being placed into operation

## 1.2 Testing and the V Models

Figure 1.1 illustrates a common way of modeling system engineering: the traditional V Model of system engineering activities.<sup>1</sup> On the left side of the V are the analysis activities that decompose the users' problem into small, manageable pieces. Similarly, the right side of the V shows the synthesis activities that aggregate (and test) these pieces into the system that solves the users' problem.

While useful, the traditional V model does not really represent system engineering from the tester's viewpoint. The next three figures show three increasingly detailed V models that better capture the testing-specific aspects of system engineering.

Figure 1.2 illustrates a V model oriented around work products rather than activities. Specifically, these are the major executable work products because testing involves the execution of work products. In this case, the left side of the V illustrates the analysis of ever more detailed executable models, whereas the right side of the V illustrates the corresponding incremental and iterative synthesis of the actual system. This V model shows the executable things that are tested rather than the general system engineering activities that generate them.

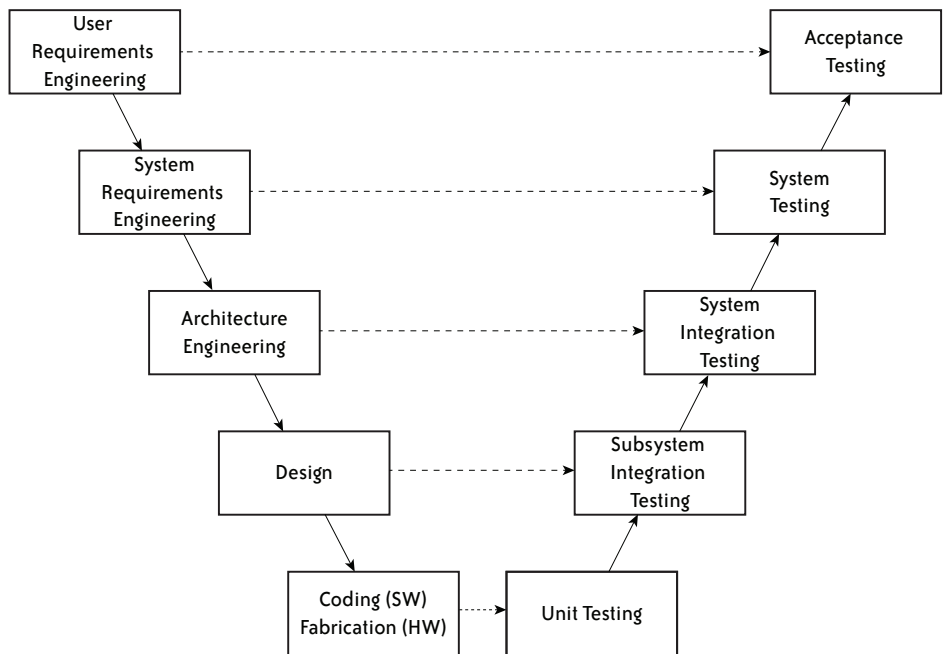


FIGURE 1.1 Traditional Single V model of system engineering activities

1. V stands for both validation and verification.

Figure 1.3 illustrates the Double-V model, which adds the corresponding tests to the Single V Model [Feiler 2012]. The key ideas to take away from this model are:

- Every executable work product should be tested. Testing need not, and in fact should not, be restricted to the implemented system and its parts. It is also important to test any executable requirements, architecture, and design. In this way, associated defects are found and fixed before they can migrate to the actual system and its parts. This typically involves testing executable requirements, architecture, or design models of the system under test (SUT) that are implemented in modeling languages (typically state-based and sufficiently formal) such as SpecTRM-RL, Architecture Analysis and Design Language (AADL), and Program Design Language (PDL); simulations of the SUT; or executable prototypes of the SUT.
- Tests should be created and performed as the corresponding work products are created. The short arrows with two arrowheads are used to show that (1) the executable work products can be developed first and used to drive the

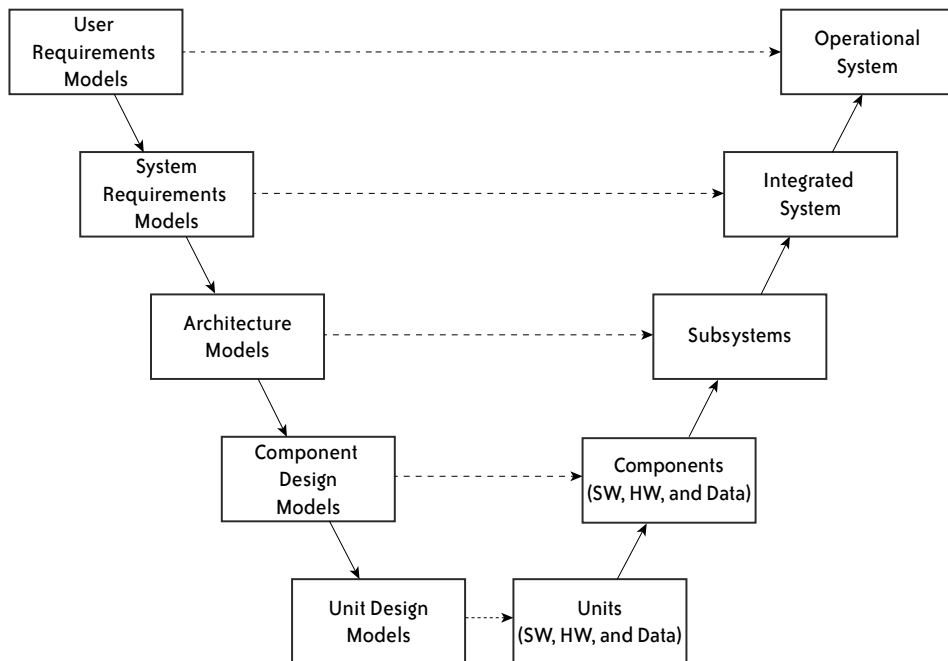


FIGURE 1.2 The Single V model of testable work products

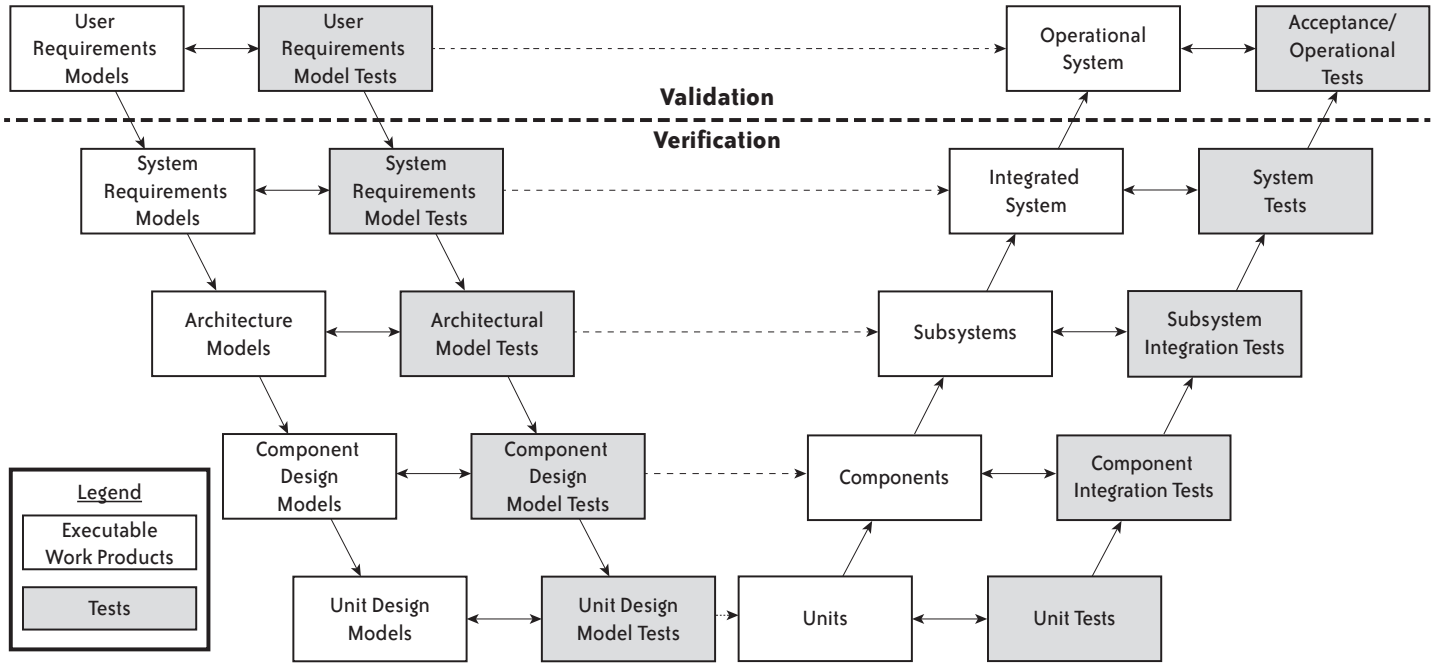


FIGURE 1.3 The Double V model of testable work products and corresponding tests

creation of the tests or (2) Test Driven Development (TDD) can be used, in which case the tests are developed before the work product they test.

- The top row of the model uses testing to *validate* that the system meets the needs of its stakeholders (that is, that the correct system is built). Conversely, the bottom four rows of the model use testing to *verify* that the system is built correctly (that is, architecture conforms to requirements, design conforms to architecture, implementation conforms to design, and so on).
- Finally, in practice, the two sides of the bottom row typically are combined so that the unit design models are incorporated into the units and so that the programming language is used as a program design language (PDL). Similarly, the unit design model tests are incorporated into the unit tests so that the same unit tests verify both the unit design and its implementation.

Figure 1.4 documents the Triple-V model, in which additional verification activities have been added to verify that the testing activities were performed properly. This provides evidence that testing is sufficiently complete and that it will not produce numerous false-positive and false-negative results.

Although the V models appear to show a sequential waterfall development cycle, they also can be used to illustrate an evolutionary (that is, incremental, iterative, and concurrent) development cycle that incorporates many small, potentially overlapping V models. However, when applying a V model to the agile development of a large, complex system, there are some potential complications that require more than a simple collection of small V models, such as:

- The architecturally significant requirements and the associated architecture need to be firmed up as rapidly as is practical because all subsequent increments depend on the architecture, which is difficult and expensive to modify once the initial increment(s) have been based on it.
- Multiple, cross-functional agile teams will be working on different components and subsystems simultaneously, so their increments must be coordinated across teams to produce consistent, testable components and subsystems that can be integrated and released.

Finally, it is interesting to note that these V models are applicable not just to the system under development but also to the development of the system's test environments or test beds and its test laboratories or facilities.

---

### 1.3 What Is a Defect?

A system *defect* (informally known as a bug) is a flaw or weakness in the system or one of its components that could cause it to behave in an unintended, unwanted manner or to exhibit an unintended, unwanted property. Defects are related to, but are different from:

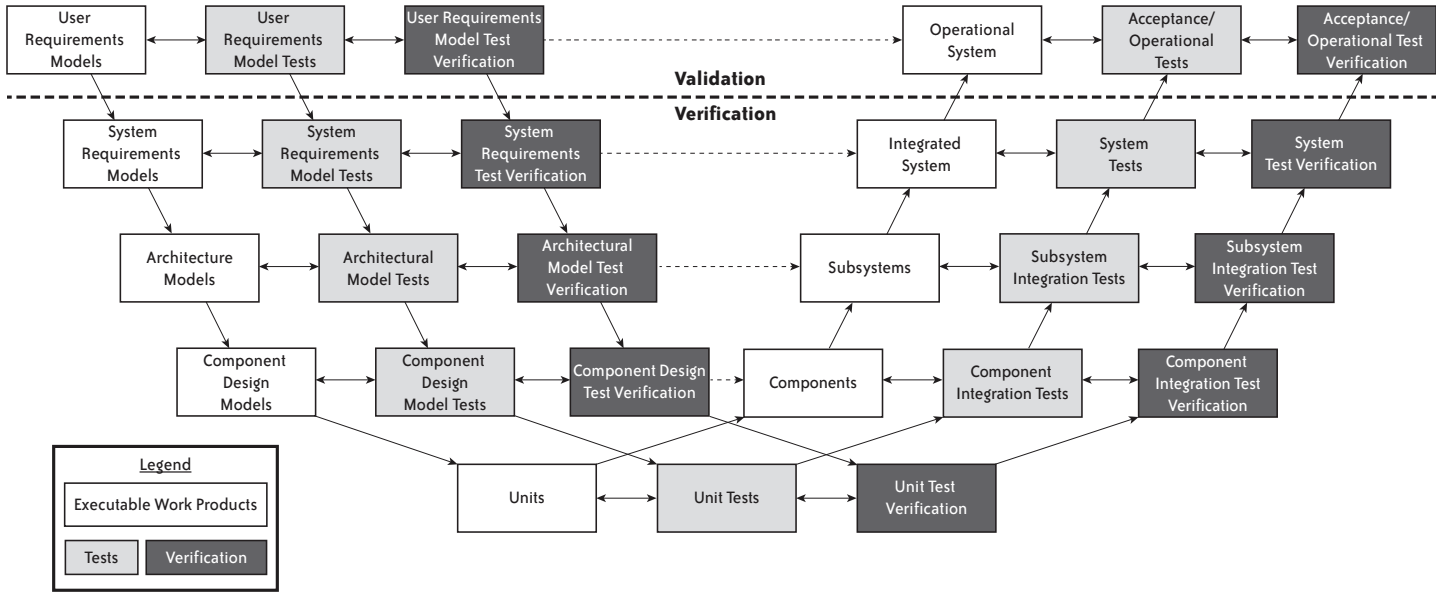


FIGURE 1.4 The Triple V model of work products, tests, and test verification

- **Errors** Human mistakes that cause the defect (for example, making a programming mistake or inputting incorrect data)
- **Faults** Incorrect conditions that are system-internal and not directly visible from outside the system's boundary (for example, the system stores incorrect data or is in an incorrect mode or state)
- **Failures** Events or conditions in which the system visibly behaves incorrectly or has incorrect properties (that is, one or more of its behaviors or properties are different from what its stakeholders can reasonably expect)

Common examples of defects include the following flaws or weaknesses:

- Defects can cause the SUT to violate specified (or unspecified) requirements, including:
  - ♦ Functional requirements
  - ♦ Data requirements
  - ♦ Interface requirements
  - ♦ Quality requirements
  - ♦ Architecture, design, implementation, and configuration constraints
- Defects can also result when the SUT conforms to incorrect or unnecessary requirements.
- Defects can cause the SUT to:
  - ♦ Fail to behave as it should
  - ♦ Be missing characteristics that it should have
  - ♦ Behave as it should not behave
  - ♦ Have characteristics that it should not have
- Defects can cause the SUT to be inconsistent with its architecture or design.
- Defects can result from incorrect or inappropriate architecture, design, implementation, or configuration decisions.
- Defects can violate design guidelines or coding standards.
- Defects can be safety or security vulnerabilities (for example, using inherently unsafe language features or failure to verify input data).

---

## 1.4 Why Is Testing Critical?

A National Institute of Standards & Technology (NIST) report [\[NIST 2002\]](#) states that inadequate testing methods and tools cost the US economy between \$22.2 billion and \$59.5 billion annually, with roughly half of these costs borne by software developers, in the form of extra testing, and half by software users, in the form of failure avoidance and mitigation efforts. The

same study notes that between 25% and 90% of software development budgets are often spent on testing.

Testing is currently the most important of the standard verification and validation methods used during system development and maintenance. This is not because testing is necessarily the most effective and efficient way to verify that the system behaves as it should; it is not. (See Table 1.1, below.) Rather, it is because far more effort, funding, and time are expended on testing than on all other types of verification put together.

According to Capers Jones, most forms of testing find only about 35% of the code defects [Jones 2013b]. Similarly, on average, individual programmers find less than half the defects in their own software.

For example, Capers Jones analyzed data regarding defect identification effectiveness from projects that were completed in early 2013 and produced the results summarized in Table 1.1 [Jones 2013a]. Thus, the use of requirements inspections identified 87% of requirements defects and 25.6% of all defects in the software and its documentation. Similarly, static analysis of the code identified 87% of the code defects and 33.2% of all defects. Finally, a project that used all of these static verification methods identified 95% of all defects.

As Table 1.2 shows, static verification methods are cumulatively more effective at identifying defects except, surprisingly, documentation defects.

TABLE 1.1 Average Percentage of Defects Found as a Function of Static Verification Method and Defect Type

Verification Method	Defect Type (Location)					Total Effectiveness
	Requirements	Architecture	Design	Code	Documentation	
<i>Requirements</i>						
Inspection	87%	5%	10%	5%	8.5%	25.6%
<i>Architecture</i>						
Inspection	10%	85%	10%	2.5%	12%	14.9%
<i>Design</i>						
Inspection	14%	10%	87%	7%	16%	37.3%
<i>Code</i>						
Inspection	15%	12.5%	20%	85%	10%	70.1%
Static Analysis	2%	2%	7%	87%	3%	33.2%
IV&V	12%	10%	23%	7%	18%	16.5%
SQA Review	17%	10%	17%	12%	12.4%	28.1%
<b>Total</b>	<b>95.2%</b>	<b>92.7%</b>	<b>96.1%</b>	<b>99.1%</b>	<b>58.8%</b>	<b>95.0%</b>

Source: Jones 2013a

TABLE 1.2 Cumulative Effectiveness at Finding Defects by Static Verification Methods, Testing, and Both

Verification Method	Defect Type (Location)					Total Effectiveness
	Requirements	Architecture	Design	Code	Documentation	
Static	95.2%	92.7%	96.1%	99.1%	58.8%	95.0%
Testing	72.3%	74.0%	87.6%	93.4%	95.5%	85.7%
Total	98.11%	98.68%	99.52%	99.94%	98.13%	99.27%

Source: Jones 2013a

## 1.5 The Limitations of Testing

In spite of its critical nature, testing has a number of pitfalls that make it far less effective and efficient than it should be. Testing is relatively ineffective in the sense that a significant number of residual defects remain in a completed system when it is placed into operation. Testing is also relatively inefficient when you consider the large amount of effort, funding, and time that is currently spent to find defects.

According to Capers Jones, most types of testing find only about 35% of the software defects [Jones 2013]. This is consistent with the following, more detailed analysis of defect detection rates as a function of test type and test capabilities, as shown in Table 1.3 [McConnell 2004].

As Table 1.4 shows, no single type of testing is very effective at uncovering defects, regardless of defect type. Even when all of these testing methods are used on an average project, they still only identify four out of five of the code defects.

TABLE 1.3 Defect Detection Rate

Test Type	Defect Detection Rates		
	Lowest	Mode	Highest
Unit Test	15%	30%	50%
Component Test	20%	30%	35%
Integration Test	25%	35%	40%
System Test	25%	40%	55%
Regression Test	15%	25%	30%
Low-volume Beta Test	25%	35%	40%
High-volume Beta Test	60%	75%	85%

Source: McConnell 2004

TABLE 1.4 Defect Detection Rate

<i>Static Verification</i>	<i>Project Defect Detection Rate</i>		
	<i>Worst</i>	<i>Average</i>	<i>Best</i>
<i>Desk Checking</i>	23%	25%	27%
<i>Static Analysis</i>	0%	55%	55%
<i>Inspection</i>	0%	0%	93%
<i>Static Subtotal</i>	19%	64%	98%

<i>Testing</i>	<i>Project Defect Detection Rate</i>		
	<i>Worst</i>	<i>Average</i>	<i>Best</i>
<i>Unit Test</i>	28%	30%	32%
<i>Function Test</i>	31%	33%	35%
<i>Regression Test</i>	10%	12%	14%
<i>Component Test</i>	28%	30%	32%
<i>Performance Test</i>	6%	10%	14%
<i>System Test</i>	32%	34%	36%
<i>Acceptance Test</i>	13%	15%	17%
<i>Testing Subtotal</i>	72%	81%	87%

<i>Cumulative Total</i>	81.1%	95.6%	99.96%
-------------------------	-------	-------	--------

Source: Jones 2013b

## 1.6 What Is a Testing Pitfall?

A testing pitfall is any decision, mindset, action, or failure to act that unnecessarily and, potentially unexpectedly, causes testing to be less effective, less efficient, or more frustrating to perform. Basically, a testing pitfall is a commonly occurring way to screw up testing, and projects fall into pitfalls when testers, managers, requirements engineers, and other testing stakeholders make testing-related mistakes that can have unintended negative consequences.

In a sense, the description of a testing pitfall constitutes a testing anti-pattern. However, the term pitfall was specifically chosen to evoke the image of a hidden or not easily identified trap for the unwary or uninitiated. As with any trap, it is better to avoid a testing pitfall than it is to have to dig one's self and one's project out of it after having fallen in.

---

## 1.7 Categorizing Pitfalls

Many testing pitfalls can occur during the development or maintenance of software-reliant systems and software applications. While no project is likely to be so poorly managed and executed as to experience the majority of these pitfalls, most projects will suffer several of them. Similarly, although these testing pitfalls do not guarantee failure, they definitely pose serious risks that need to be managed.

This book documents 92 pitfalls that have been observed to commonly occur during testing. These pitfalls are categorized as follows:

- [General Testing Pitfalls](#)
  - ♦ [Test Planning and Scheduling Pitfalls](#)
  - ♦ [Stakeholder Involvement and Commitment Pitfalls](#)
  - ♦ [Management-Related Testing Pitfalls](#)
  - ♦ [Staffing Pitfalls](#)
  - ♦ [Test Process Pitfalls](#)
  - ♦ [Test Tools and Environments Pitfalls](#)
  - ♦ [Test Communication Pitfalls](#)
  - ♦ [Requirements-Related Testing Pitfalls](#)
- [Test-Type-Specific Pitfalls](#)
  - ♦ [Unit Testing Pitfalls](#)
  - ♦ [Integration Testing Pitfalls](#)
  - ♦ [Specialty Engineering Testing Pitfalls](#)
  - ♦ [System Testing Pitfalls](#)
  - ♦ [System of Systems \(SoS\) Testing Pitfalls](#)
  - ♦ [Regression Testing Pitfalls](#)

Although each of these testing pitfalls has been observed on multiple projects, it is entirely possible that you might have testing pitfalls that are not addressed by this document. Please notify me of any new testing pitfalls you stumble across or any additional recommended changes to the current pitfalls so that I can incorporate them into future editions of this book.

---

## 1.8 Pitfall Specifications

Chapter 2 gives high-level descriptions of the different pitfalls, while Chapter 3 documents each testing pitfall with the following detailed information:

- **Title** A short, descriptive name of the pitfall
- **Description** A brief definition of the pitfall
- **Potential Applicability** The context in which the pitfall may be applicable
- **Characteristic Symptoms (or, How You Will Know)** Symptoms that indicate the possible existence of the pitfall
- **Potential Negative Consequences (Why You Should Care)** Potential negative consequences to expect if the pitfall is not avoided or mitigated[3]
- **Potential Causes** Potential root and proximate causes of the pitfall[4]
- **Recommendations (What You Should Do)** Recommended actions (prepare, enable, perform, and verify) to take to avoid or mitigate the pitfall[5]
- **Related Pitfalls** A list of other related testing pitfalls

A few words on word choice and grammar are probably appropriate before you start reading about the individual pitfalls:

- **Potential Applicability** You may fall into these pitfalls on your project, but then again you may not. Some pitfalls will be more probable and therefore more relevant than others. Of course, if you have already fallen into a given pitfall, it ceases to be potentially applicable and is now absolutely applicable. Because potential applicability currently exists, it is described in the present tense.
- **Characteristic Symptoms** You may have observed these symptoms in the past, and you may well be observing them now. They may even be waiting for you in the future. To save me from having to write all three tenses and, more importantly, to save you from having to read them all, I have listed all symptoms in present tense.
- **Potential Negative Consequences** Once again, you may have suffered these consequences in the past, or they may be happening now. These consequences might still be in the future and avoidable (or subject to mitigation) if you follow the appropriate recommendations now. These consequences are also listed in the present tense.

Note that sometimes the first symptom(s) of a pitfall are the negative consequence(s) you are suffering from because you fell into it. Therefore, it is not always obvious whether something should be listed under symptoms, consequences, or both. To avoid listing the same negative event or situation twice for the same pitfall, I have endeavored to include it only once under the most obvious heading.

- **Potential Causes** Finally, the causes may also lie in your past, your present, or your future. However, they seem to sound best when written in the past tense, for they must by their very nature precede the pitfall's symptoms and consequences.

*This page intentionally left blank*

# INDEX

## A

AADL models. *See* Architecture Analysis and Design Language (AADL) models

abnormal behavior  
residual defects, 45  
testing, 41

abnormal use case paths, 193, 196

A/B testing, 243

abuse cases, 200, 202

acceptance testing, 31, 104  
defects causing failures, 59–60  
system testing as, 105

accidental regression testing, 104

accreditation testing, 115

acquirer representatives, 57

acquisition organizations, 141

acronym list, 33, 145–146

acronyms  
listing, 253–254  
undefined, 147

agile development, 4, 127–128  
consulting, 95  
cross-functional development teams, 71  
denigrating documentation in favor of verbal communication, 133  
developers' testing expertise, 72  
integrated components, 87  
project planning documentation, 96  
schedules, 96  
testing documentation, 136, 137  
testing performed by non-testers, 73  
test planning documentation, 38, 96  
training, 95  
working on different components and subsystems simultaneously, 4

ALM tool repositories. *See* Application Lifecycle Management (ALM) tool repositories

alpha testing, 100

alternative use case paths, 103, 151

Ambiguous Requirements (GEN-REQ-1), 19, 144–147, 220  
characteristic symptoms, 144  
description, 144  
potential applicability, 144  
potential causes, 145  
potential negative consequences, 145  
recommendations, 146–147  
related pitfalls, 147

analysis, 243

Anti-Tamper Testing, 200

application domain, 70

Application Lifecycle Management (ALM) tool repositories  
acquiring, 30  
testing, xix  
test planning information stored in, 29

apps, 120

architects, 141  
insufficient architecture or design documentation, 131–133  
testing executable architectural models, 167

architecturally significant requirements, 4

architecture  
inadequate, 18  
testing, 203  
tests driving development, 165–166  
undocumented changes to, 139

Architecture Analysis and Design Language (AADL) models, 3

architecture documents, 139–140

architecture engineering process, 160

architecture models, testing, 3

architecture teams, testers excluded from, 76

as-performed testing process  
evaluating, xix  
metrics-driven, 61  
quality assurance evaluations, 92

*Assessment and Control of Software Risks* (Jones), xiv

asset analysis, 199, 202

attack surface identification, 202

attack trees, 200, 202

automated testing, 31, 70–71, 107–108, 166  
difficulty developing, 18, 129–130  
expertise required, 48  
Proof of Concept study, 107  
relying too heavily on, 108–110  
return on investment (ROI), 107  
test cases, 47

automating regression testing, 113, 226–227, 229–231

## B

backward-looking testing metrics, 60

behaviors  
exceptional, 104  
failing to test, 17, 98–100  
normal versus abnormal, 31, 104  
testing documentation, 78

best practices, 70

beta testing, 100, 243

BIST. *See* Built-In Self Test

BIT. *See* built-in test

black-box system  
little time for testing, 87  
no metrics collected, 87  
overemphasizing, 16, 85–86  
testing, 31, 38, 86–88, 104

Black-Box System Testing Overemphasized (GEN-PRO-5)  
characteristic symptoms, 85  
description, 85  
potential applicability, 85  
potential causes, 85  
potential negative consequences, 85  
recommendations, 86  
related pitfalls, 86

- Black-Box System Testing Underemphasized (GEN-PRO-6)
    - characteristic symptoms, 86–87
    - description, 86
    - potential applicability, 86
    - potential causes, 87
    - potential negative consequences, 87
    - recommendations, 87–88
    - related pitfalls, 88
  - black-box testing, 109, 243
    - incorrect, 145
    - requirements, 146, 149, 152
    - safety requirements, 199
    - security requirements, 202
    - test cases, 154
  - boundary values, 46, 103
  - boundary-value testing, 103, 195, 243
  - branch coverage, 243–244
  - budgets, 29, 51–52, 63
  - bug reports. *See* defect reports
  - bugs, 4, 54
  - build-test-fix cycle, 166
  - Built-In Self Test (BIST), 171
  - built-in test (BIT), 171, 172, 196, 209, 244
- C**
- capacity requirements, 178–180
  - capacity testing, performing little or none, 21, 178–180
  - CCB. *See* change control board
  - certification testing, 115
  - change control board (CCB), 136, 222, 225
  - change-impact analysis, 230, 232, 233
  - change-request impact, 39, 41
  - characteristics, failing to test, 17
  - chief system engineer, 77
  - classified networks, prohibiting access to, 121
  - class under test, 129
  - cloud testing, 118, 244
    - Testing as a Service (TaaS), 117
  - CM. *See* configuration management
  - code
    - coverage, 244
    - identifying defects, 8
  - collaboration websites, 29
  - commercial off-the-shelf (COTS) software, 32, 108, 121
    - not tested, 99
  - commercial off-the-shelf (COTS) testing tools, 17, 124
  - communication inadequate regarding testing, 19, 140–143
  - completed system, no operational testing on, 100–102
  - components
    - concurrent, 181–183
    - difficulty testing encapsulated, 20
    - failing to test, 17
    - not certified, 122
    - unavailable, 20, 173–175
  - computational environments, 114
  - concurrency faults and failures, 21, 181
  - concurrency testing, little or none performed, 21, 181–183
  - concurrent development cycle, 244
  - concurrent testing, 95
  - condition coverage, 244
  - configuration management (CM), 93
    - inadequate, 18
    - test products not under, 126–128
  - conflict of interest, 45
  - Conflict of Interest (TTS-UNT-2), 20, 166–168
  - constructive quality model, 59
  - consulting, 95
  - context modeling, 151
  - contractors
    - proposals, xix
    - versus subcontractors, 141
  - contracts, 27, 132
    - test assets, 125
  - contractual relationships, 29
  - corner cases, 46
  - COTS. *See* commercial off-the-shelf (COTS) testing tools
  - countermeasure analysis, 202
  - credible exception “rainy-day” use case paths, 103
  - critical defects
    - not discovered until system is in operation, 80
    - not finding, 60
  - cross-functional engineering teams, 76
  - crowdsource testing, 244
  - current release scope, 41
  - customer representatives, 45, 56, 141
- D**
- data
    - inadequate, 17
    - poor fidelity, 120
    - requirements not verified, 102
  - databases
    - imperfect replica of data, 118
    - test data not migrated to new versions, 93
    - updating, 94
  - data/control interfaces, 186
    - protocols, 186
  - data requirements, 83, 150
  - date-based testing, 32
  - DBC. *See* design by contract
  - deadlock, 181, 182
  - decision coverage, 244
  - decision makers, 68
  - defect reports, 18, 134–136
  - defects, 244–245
    - assigning responsibility for finding and fixing to other projects, 224–225
  - boundary values, 46
  - corner cases, 46
  - cost per defect, 59
  - counting, 59
  - decision makers and, 68
  - defining, 4, 7
  - deployed system, 78
  - effectiveness, 8
  - effectiveness finding, 59
  - finding with testing, 8
  - functionality-related, 24
  - identification and reporting, 55
  - importance of, 60
  - inadequate tracking across projects, 23, 222–224
  - incomplete requirements, 153
  - increasing, 55
  - input type, 46
  - insufficient time to correct, 42, 55
  - latent remaining, 59
  - measuring and reporting, 90
  - missed by testing, 9
  - missing requirements, 150
  - mission-critical, 60
  - negative impact of execution, 100
  - not fixing, 62, 80
  - not found, 91
  - priority and severity, 56, 134
  - quality attributes, 59–60
  - range verification, 46
  - residual defects delivered, 26
  - safety-critical, 7, 60
  - security-critical, 7, 60
  - severity, 134
  - slipping through prior testing, 59
  - system under test (SUT), 7
  - violating design guidelines or coding standards, 7
- deliverable software
    - architectural and design models and documentation, 132
    - target environments, 116
  - delivered system
    - evaluating, 91
    - failing to meet system requirements, 87
    - failing to remove test hooks, 206–208
    - residual defects, 45, 60
  - demonstrations, 245
  - derived requirements, 159, 245
  - design
    - no completion criteria, 89
    - testing not driving, 20
    - tests driving development, 165–166
    - undocumented changes to, 139
  - design by contract (DBC), 103
  - design documentation, 18
    - not maintaining, 139–140
  - design-driven testing, 109
  - designers, 141
    - insufficient architecture or design documentation, 131–133
    - testing executable design models, 167
  - design models
    - defects, 97
    - testing, 3
  - developed in-house software, 32
  - developers
    - bottleneck lowering productivity, 74
    - build-test-fix cycle, 166
    - components working properly with-out integration, 87
    - conflict of interest, 20, 72, 167–168
    - evolutionary development cycle, 95
    - false sense of security, 45
    - ignoring testability, 18
    - low-level testing training, 75
    - negative relationships, 55

- not considering testing, 129-130
- passing on units with defects, 89
- performing testing, 87
- system behaviors, 72
- test case development, 96, 97
- testing responsibilities, 74
- testing their own work products, 20, 45, 74
- unit testing, 75
- what constitutes bug/defect, 54
- withholding information about negative tests, 72
- Developers Ignore Testability (GEN-TTE-10)
  - characteristic symptoms, 129
  - description, 129
  - potential applicability, 129
  - potential causes, 129-130
  - potential negative consequences, 129
  - recommendations, 130
  - related pitfalls, 130
- Developers Responsible for All Testing (GEN-STF-4), 72-73, 75, 102, 177
  - characteristic symptoms, 72
  - description, 72
  - potential applicability, 72
  - potential causes, 73
  - potential negative consequences, 72
  - recommendations, 73
  - related pitfalls, 73
- developmental testing, 31, 245
- development cycle
  - defects identified late in, 95
  - evolutionary, 4
  - short-duration, iterative increments, 157
  - testing performed late in, 14, 42-43
  - waterfall, 4
- development manager, 66
- development organizations, 141
- development process documents, 79
- development teams, 143
  - geographical distribution of, 29
  - responsibility for all testing, 15
  - sharing responsibility for testing, 72-73
- documentation
  - evaluating, 27
  - identifying defects, 8
  - rejecting inadequate, 27
  - system of systems (SoS) test planning, 212-213
  - test planning documentation, 13
  - updating for test process, 28
- domain-specific glossary, 145, 146
- Double-V model, 3-4, 5, 43
- dynamic testing, 32
- E**
  - edge cases, 103
  - electrical interfaces, 186
  - emulator, 245
  - encapsulated components, difficulty testing, 20, 172-173
  - end-to-end mission thread modeling, 152
  - end-to-end mission threads, 209-211
  - end-to-end system of systems (SoS) testing
    - failure to define and document responsibilities for, 213-215
    - little or none performed, 216
  - end-to-end testing, 22
  - end-user testing, 104
  - engineering process, 76
    - not integrated with testing process, 16
  - entry/exit coverage, 245
  - environmental conditions, detecting and reacting to dangerous, 193-196
  - environmental-tolerance testing, 193
  - environment and poor fidelity, 120
  - erroneous input, 245
  - error reporting, 172, 173
  - errors, 7, 245
  - error tolerance, 45, 152, 194, 195, 245
  - error-tolerance testing, 193
  - error-tolerant path, 103, 151, 152
  - ETA. *See* Event Tree Analysis
  - event schedulability, 188-190
  - Event Tree Analysis (ETA), 193, 194, 196, 197, 199
  - evidence-based cost/effort models, 53-54
  - evolutionary development cycle, 4, 43, 95-96
    - requirements not frozen, 157
    - testing requirements, 149
  - exceptional behavior, 44, 104
  - executable architecture defects, 97
  - executable work products, 3-4
  - expected test outputs, 92
  - external pressures, inappropriate, 15
- F**
  - fail-safe behavior, 197
  - fail-secure behavior, 202
  - failure
    - detecting and reacting to, 193-196
    - negative impact of, 100
  - Failure Modes Effects and Criticality Analysis (FMECA), 193, 194, 196, 199
  - failure paths, 151
  - failures, 7, 134, 245
  - failure tolerance, 245
  - failure-tolerance testing, 193
  - failure-tolerant behavior, 44
  - failure-tolerant use case paths, 103
  - false-negative test results, 70, 90, 93, 118, 119, 122, 127, 145, 153, 154-156, 165, 245
  - false-positive test results, 70, 90, 118, 119, 122, 127, 145, 153, 154-156, 165, 246
  - fault logging, 196
  - faults, 7, 134, 246
    - detecting and reacting to, 193-196
  - fault tolerance, 172, 246
  - fault-tolerance testing, 193
  - fault-tolerant behavior, 44
  - fault-tolerant paths, 151
  - fault-tolerant use case paths, 103
  - Fault Tree Analysis (FTA), 193, 194, 196, 197, 199
  - Finger-Pointing (TTS-SoS-8)
    - characteristic symptoms, 224
    - description, 224
    - potential applicability, 224
    - potential causes, 224-225
    - potential negative consequences, 224
    - recommendations, 225
    - related pitfalls, 225
  - flight certified components, 122
  - FMECA. *See* Failure Modes Effects and Criticality Analysis
  - foreign characters, 183
  - forward-looking testing metrics, 60
  - FTA. *See* Fault Tree Analysis
  - functionality-related defects, 24, 237-239
  - functionality testing, overemphasizing, 16, 82-84
  - Functionality Testing Overemphasized (GEN-PRO-4), 82-84, 104
    - characteristic symptoms, 82-83
    - description, 82
    - potential applicability, 82
    - potential causes, 83-84
    - potential negative consequences, 83
    - recommendations, 84
    - related pitfalls, 84
  - functional requirements, 78, 152, 246
  - functional testing, 246
  - function coverage, 246
  - fuzz testing, 246
- G**
  - general-purpose hardware access ports, 112
  - general-purpose platforms, 114
  - general testing pitfalls, 11
    - Inadequate Test Schedule (GEN-TPS-5), 39-41
    - Incomplete Test Planning (GEN-TPS-2), 13, 31-35
    - management-related testing pitfalls, 14-15, 51-65
    - No Separate Test Planning Documentation (GEN-TPS-1), 13, 28-31
    - requirements-related testing pitfalls, 19-20, 143-164
    - staffing pitfalls, 15-16, 65-75
    - stakeholder involvement and commitment pitfalls, 14, 44-51
    - Test-Case Documents as Test Plans (GEN-TPS-4), 37-39
    - test communication pitfalls, 18-19, 131-143
    - Testing at the End (GEN-TPS-6), 42-43
    - test planning and scheduling pitfalls, 13-14, 28
    - Test Plans Ignored (GEN-TPS-3), 35-37
    - test process pitfalls, 16-17, 75-106
    - test tools and environments pitfalls, 17-18, 106-130
  - GFE. *See* government furnished equipment
  - golden path, 103, 152, 249
  - GOTS. *See* governmental-off-the-shelf

- government furnished equipment (GFE), 32
- Government-off-the-Shelf (GOTS) software, 32, 108
- graphical user interface (GUI), 129
- gray-box testing, 31, 86, 246
- GUI. *See* graphical user interface
- H**
- habitability requirements and architecture, 204
- hacker tools, 112
- hallway intercept testing, 246
- happy path, 103
- hardware, 110
  - poor fidelity, 119–120
  - test environments, 116, 117
  - unavailable components, 173–175
- hardware emulation, 114
- hazard analysis, 199
- hazard testing, 197
- HFE. *See* human factors engineering (HFE) and architecture
- high-level regression testing, 234–235
- high-level testing, 73
- high-level testing metrics, 60
- home-grown software, 108
- homegrown testing tools, 17
- how-to books on testing, xvii
- human factors engineering (HFE) and architecture, 203–206
- hydraulic interfaces, 186
- I**
- IBIT. *See* Interrupt-driven BIT
- implementation
  - no completion criteria, 89
  - testing, 165
  - testing not driving, 20
  - tests driving development, 165–166
  - undocumented changes to, 139
- implementors, 141
- Improperly Derived Requirements (GEN-REQ-7), 19–20, 159–160
  - characteristic symptoms, 159
  - description, 159
  - potential applicability, 159
  - potential causes, 160
  - potential negative consequences, 159
  - recommendations, 160
  - related pitfalls, 160
- IMS, 93
- Inadequate Architecture or Design Documentation (GEN-COM-1), 100, 131–133, 140, 164
  - characteristic symptoms, 131
  - description, 131
  - potential applicability, 131
  - potential causes, 132–133
  - potential negative consequences, 132
  - recommendations, 133
  - related pitfalls, 133
- Inadequate Capacity Testing (TTS-SPC-1), 21, 35, 178–180, 239
  - characteristic symptoms, 178
  - description, 178
  - potential applicability, 178
  - potential causes, 179–180
  - potential negative consequences, 179
  - recommendations, 180
  - related pitfalls, 180
- Inadequate Communication Concerning Testing (GEN-COM-5), 19, 47, 49, 51, 77, 79, 130, 141–143, 147, 149, 152, 154, 158, 162
  - characteristic symptoms, 141
  - description, 140
  - potential applicability, 140
  - potential causes, 142
  - potential negative consequences, 141–142
  - recommendations, 142–143
  - related pitfalls, 143
- Inadequate Concurrency Testing (TTS-SPC-2), 181–183, 190, 239
  - characteristic symptoms, 181
  - description, 181
  - potential applicability, 181
  - potential causes, 182
  - potential negative consequences, 181–182
  - recommendations, 182–183
  - related pitfalls, 183
- Inadequate Defect Reports (GEN-COM-2)
  - characteristic symptoms, 134–135
  - description, 134
  - potential applicability, 134
  - potential causes, 135
  - potential negative consequences, 135
  - recommendations, 135–136
  - related pitfalls, 136
- Inadequate Defect Tracking Across Projects (TTS-SoS-7), 23, 223–224
  - characteristic symptoms, 222
  - description, 222
  - potential applicability, 222
  - potential causes, 223
  - potential negative consequences, 223
  - recommendations, 223–224
  - related pitfalls, 224
- Inadequate End-to-End Testing (TTS-SYS-3), 22, 100, 209–211
  - characteristic symptoms, 209
  - description, 209
  - potential applicability, 209
  - potential causes, 210
  - potential negative consequences, 210
  - recommendations, 210–211
  - related pitfalls, 211
- Inadequate Evaluations of Test Assets (GEN-PRO-8), 35, 90–92
  - characteristic symptoms, 90
  - description, 90
  - potential applicability, 90
  - potential causes, 91
  - potential negative consequences, 90–91
  - recommendations, 91–92
  - related pitfalls, 92
- Inadequate Internationalization Testing (TTS-SPC-3), 21, 35, 183–185, 239
  - characteristic symptoms, 183
  - description, 183
  - potential applicability, 183
  - potential causes, 184–185
  - potential negative consequences, 184
  - recommendations, 185
  - related pitfalls, 185
- Inadequate Interoperability Testing (TTS-SPC-4), 21, 185–188
  - characteristic symptoms, 186
  - description, 185
  - potential applicability, 185
  - potential causes, 187
  - potential negative consequences, 186–187
  - recommendations, 187–188
  - related pitfalls, 188
- Inadequate Maintenance of Test Assets (GEN-PRO-9), 92–94, 138, 164, 228, 231, 234
  - characteristic symptoms, 92
  - description, 92
  - potential applicability, 92
  - potential causes, 93
  - potential negative consequences, 93
  - recommendations, 93–94
  - related pitfalls, 94
- Inadequate Performance Testing (TTS-SPC-5), 21, 35, 183, 188–190, 239
  - characteristic symptoms, 188
  - description, 188
  - potential applicability, 188
  - potential causes, 189
  - potential negative consequences, 188–189
  - recommendations, 189–190
  - related pitfalls, 190
- Inadequate Regression Test Automation (TTS-REG-1), 23, 108, 225–228
  - characteristic symptoms, 226
  - description, 225
  - potential applicability, 225
  - potential causes, 226–227
  - recommendations, 227–228
  - related pitfalls, 228
- Inadequate Reliability Testing (TTS-SPC-6), 21, 35, 104, 190–193, 239
  - characteristic symptoms, 191
  - description, 190
  - potential applicability, 190
  - potential causes, 191–192
  - potential negative consequences, 191
  - recommendations, 192–193
  - related pitfalls, 193
- Inadequate Resources for SoS Testing (TTS-SoS-3), 23, 211, 215–217
  - characteristic symptoms, 215–216
  - description, 215
  - potential applicability, 215
  - potential causes, 216
  - potential negative consequences, 216
  - recommendations, 216–217

- related pitfalls, 217
- Inadequate Robustness Testing (TTS-SPC-7), 22, 35, 104, 193–196, 239
  - characteristic symptoms, 193–194
  - description, 193
  - potential applicability, 193
  - potential causes, 194–195
  - potential negative consequences, 194
  - recommendations, 195–196
  - related pitfalls, 196
- Inadequate Safety Testing (TTS-SPC-8), 22, 35, 104, 197–199, 203, 239
  - characteristic symptoms, 197
  - description, 197
  - potential applicability, 197
  - potential causes, 198
  - potential negative consequences, 197
  - recommendations, 198–199
  - related pitfalls, 199
- Inadequate Scope of Regression Testing (TTS-REG-3), 23, 231–234, 235
  - characteristic symptoms, 231
  - description, 231
  - potential applicability, 231
  - potential causes, 232
  - potential negative consequences, 232
  - recommendations, 233–234
  - related pitfalls, 234
- Inadequate Security Testing (TTS-SPC-9), 22, 35, 104, 199, 200–203, 239
  - characteristic symptoms, 200
  - description, 200
  - potential applicability, 200
  - potential causes, 201
  - potential negative consequences, 200–201
  - recommendations, 201–203
  - related pitfalls, 203
- Inadequate Self-Monitoring (TTS-INT-2), 20, 172–173
  - characteristic symptoms, 172
  - description, 172
  - potential applicability, 172
  - potential causes, 173
  - potential negative consequences, 172–173
  - recommendations, 173
  - related pitfalls, 173
- Inadequate SoS Planning (TTS-SoS-1), 23, 35, 211–213
  - characteristic symptoms, 212
  - description, 212
  - potential applicability, 212
  - potential causes, 212
  - recommendations, 213
  - related pitfalls, 213
- Inadequate SoS Requirements (TTS-SoS-5), 23, 147, 149, 152, 156, 220–222
  - characteristic symptoms, 219
  - description, 219
  - potential applicability, 219
  - potential causes, 220
  - potential negative consequences, 219–220
  - recommendations, 220
  - related pitfalls, 220
- Inadequate Support from Individual System Projects (TTS-SoS-6), 23, 51, 221–222, 225
  - characteristic symptoms, 221
  - description, 220
  - potential applicability, 220
  - potential causes, 221
  - potential negative consequences, 221
  - recommendations, 221–222
  - related pitfalls, 222
- Inadequate Test Configuration Management (GEN-TTE-9), 117, 124, 126–128, 138, 140, 228, 231, 234
  - characteristic symptoms, 126
  - description, 126
  - potential applicability, 126
  - potential causes, 127–128
  - potential negative consequences, 126–127
  - recommendations, 128
  - related pitfalls, 128
- Inadequate Test Data (GEN-PRO-14)
  - characteristic symptoms, 102
  - description, 102
  - potential causes, 103
  - potential negative consequences, 102
  - recommendations, 103–104
  - related pitfalls, 104
- Inadequate Test Documentation (GEN-COM-3), 19, 31, 69, 128, 136–138, 143
  - characteristic symptoms, 136
  - description, 136
  - potential applicability, 136
  - potential causes, 137
  - potential negative consequences, 136–137
  - recommendations, 137–138
  - related pitfalls, 138
- Inadequate Test Environment Quality (GEN-TTE-7), 117, 122–124
  - characteristic symptoms, 122
  - description, 122
  - potential applicability, 122
  - potential causes, 122–123
  - potential negative consequences, 122–123
  - recommendations, 123–124
  - related pitfalls, 124
- Inadequate Test Environments (GEN-TTE-5), 102, 112, 114–117, 128
  - characteristic symptoms, 114–115
  - description, 114
  - potential applicability, 114
  - potential causes, 116
  - potential negative consequences, 115–116
  - recommendations, 116–117
  - related pitfalls, 117
- Inadequate Testing Expertise (GEN-STF-3), 54, 69–71, 73, 96, 100, 104, 106, 166, 177
  - characteristic symptoms, 69–70
  - description, 69
  - potential applicability, 69
  - potential causes, 70–71
  - potential negative consequences, 70
  - recommendations, 71
  - related pitfalls, 71
- Inadequate Test Metrics (GEN-MGMT-4), 15, 59–61
  - characteristic symptoms, 59–60
  - description, 59
  - potential applicability, 59
  - potential causes, 60
  - potential negative consequences, 60
  - recommendations, 60–61
  - related pitfalls, 60–61
- Inadequate Test Prioritization (GEN-PRO-3), 16, 79, 80–82, 100, 112, 128
  - characteristic symptoms, 80
  - description, 80
  - potential applicability, 80
  - potential causes, 81
  - potential negative consequences, 80
  - recommendations, 81–82
  - related pitfalls, 82
- Inadequate Test-Related Risk Management (GEN-MGMT-3)
  - characteristic symptoms, 57
  - description, 57
  - potential applicability, 57
  - potential causes, 58
  - potential negative consequences, 57
  - recommendations, 58
  - related pitfalls, 58
- Inadequate Test Resources (GEN-MGMT-1), 51, 52–54, 71, 100, 102, 112, 128, 177, 228, 231, 234
  - characteristic symptoms, 52
  - description, 52
  - potential applicability, 52
  - potential causes, 53
  - potential negative consequences, 52–53
  - recommendations, 53–54
  - related pitfalls, 54
- Inadequate Test Schedule (GEN-TPS-5), 13, 39–41, 51, 98, 100, 106, 177, 228, 231, 234
  - characteristic symptoms, 39–40
  - description, 39
  - potential applicability, 39
  - potential causes, 40
  - potential negative consequences, 40
  - recommendations, 40–41
  - related pitfalls, 41
- Inadequate Usability Testing (TTS-SPC-10), 22, 35, 203–206, 239
  - characteristic symptoms, 203–204
  - description, 203
  - potential applicability, 203
  - potential causes, 204–205
  - potential negative consequences, 204
  - recommendations, 205–206
  - related pitfalls, 206

- Inappropriate External Pressures (GEN-MGMT-2), 15, 47, 49, 51, 54–56, 67, 73, 206
    - characteristic symptoms, 54–55
    - description, 54
    - potential applicability, 54
    - potential causes, 55–56
    - potential negative consequences, 55
    - recommendations, 56
    - related pitfalls, 56
  - inch pebble, 61
  - incomplete requirements, 152–154
  - Incomplete Requirements (GEN-REQ-4), 19, 147, 149, 152–154,
    - characteristic symptoms, 152–153
    - description, 152
    - potential applicability, 152
    - potential causes, 153
    - potential negative consequences, 153
    - recommendations, 153–154
    - related pitfalls, 154
  - Incomplete Testing (GEN-PRO-12), 17, 98–100, 106
    - characteristic symptoms, 98
    - description, 98
    - potential applicability, 98
    - potential causes, 98–99
    - potential negative consequences, 98
    - recommendations, 99–100
    - related pitfalls, 100
  - Incomplete Test Planning (GEN-TPS-2), 13, 31–35, 36, 38, 39, 69, 92, 94, 100, 102, 104, 106, 112, 126, 128, 166, 177, 180, 183, 185, 188, 190, 193, 196, 199, 203, 206, 211, 213, 228, 231, 234, 237
    - characteristic symptoms, 31–33
    - description, 31
    - organizations, 32
    - potential applicability, 31
    - potential causes, 34
    - potential negative consequences, 33
    - recommendations, 34–35
    - related pitfalls, 35
    - scope of testing, 31
    - test goals and objectives, 31
    - test levels, 31
    - test process, 32
    - test types, 31–32
  - Inconvenient Test Results Ignored (GEN-MGMT-5), 15, 61–63, 206
    - characteristic symptoms, 61–62
    - description, 61
    - potential applicability, 61
    - potential causes, 62–63
    - potential negative consequences, 62
    - recommendations, 63
    - related pitfalls, 63
  - incorrect requirements, 154–156
  - Incorrect Requirements (GEN-REQ-5), 19, 149, 154–156, 220
    - characteristic symptoms, 155
    - description, 154
    - potential applicability, 154
    - potential causes, 155
    - potential negative consequences, 155
    - recommendations, 155–156
    - related pitfalls, 156
  - incremental, iterative testing, 43
  - incremental development cycle, 246
  - incremental testing, 95
  - independent product teams (IPTs), 189, 192
  - Independent Research and Development (IRAD) funding, 123
  - independent technical assessments (ITAs), xvii
    - independent testing, 66, 74
  - independent test team, 63, 179, 184, 189, 192, 195, 198, 201, 205
  - independent verification and validation (IV&V) program, 66, 67
  - independent verification and validation (IV&V) testing, 115
  - industry best practices, 105
  - infinite loops, unintentional, 181
  - information gathering, 201
  - information hiding, 22, 208–209
  - input data, 44
  - input errors, 193–196
  - inputs, 46, 227
  - inspection, 246
  - integrated product teams (IPTs), 143, 179, 184, 195, 198, 201, 205
  - integrated test system, 114
  - integration, 80, 169–172
  - Integration Decreases Testability Ignored (TTS-INT-1), 20, 130, 169–173
    - characteristic symptoms, 169
    - description, 169
    - potential applicability, 169
    - potential causes, 170–171
    - potential negative consequences, 169–170
    - recommendations, 171–172
    - related pitfalls, 172
  - integration-level testing, 59, 86
  - integration testing, 60, 86, 172, 246
    - insufficient architecture or design documentation, 18, 131–133
    - interface defects, 59
    - little or none, 85
    - repeating unit-level tests, 104
    - system-level testing as, 20, 175–177
    - unavailable components, 173–175
  - integration testing pitfalls, 11, 20–21, 169–177
  - Inadequate Self-Monitoring (TTS-INT-2), 20, 172–173
  - Integration Decreases Testability Ignored (TTS-INT-1), 20, 169–172
  - System Testing as Integration Testing (TTS-INT-4), 20, 175–177
  - Unavailable Components (TTS-INT-3), 20, 173–175
  - interactions, little or none tested, 176
  - interface defects, 59
  - interface requirements, 83, 102, 150, 172
  - interface testing, 243
  - internationalization testing, 21, 183–185
  - International Software Certifications Board (ISCB), 70
  - International Software Testing Qualifications Board (ISTQB), 70
  - international symbols, 183
  - Internet, 118
  - interoperability, 246–247
  - interoperability testing, 21, 185–188
  - Interrupt-driven BIT (IBIT), 171, 172
  - invariants, 103
  - IPTs. *See* integrated product teams
  - IRAD funding. *See* Independent Research and Development (IRAD) funding
  - ISCB. *See* International Software Certifications Board
  - ISTQB. *See* International Software Testing Qualifications Board
  - iterative development cycle, 247
  - iterative testing, 95
  - IV&V program. *See* independent verification and validation (IV&V) program
- J**
- jailbreaking, 112
  - jitter, 188–190
  - Jones, Capers, 8, 9, 73
- L**
- Lack of Independence (GEN-STF-1), 15, 56, 66–67
    - characteristic symptoms, 66
    - description, 66
    - potential applicability, 66
    - potential causes, 66–67
    - potential negative consequences, 66
    - recommendations, 67
    - related pitfalls, 67
  - Lack of Requirements Trace (GEN-REQ-9), 20, 160, 162–164
    - characteristic symptoms, 163
    - description, 162
    - potential applicability, 162
    - potential causes, 163
    - potential negative consequences, 163
    - recommendations, 164
    - related pitfalls, 164
  - Lack of Stakeholder Commitment to Testing (GEN-SIC-3), 14, 41, 49–51, 54, 65, 130, 156, 217, 222
    - characteristic symptoms, 49–50
    - description, 49
    - potential applicability, 49
    - potential causes, 50–51
    - potential negative consequences, 50
    - recommendations, 51
    - related pitfalls, 51
  - Lack of Test Hooks (TTS-SYS-2), 22, 114, 208–209
    - characteristic symptoms, 208
    - description, 208
    - potential applicability, 208
    - potential causes, 208

- potential negative consequences, 208
    - recommendations, 209
    - related pitfalls, 209
  - latency, 188–190
  - lessons learned, 64, 65
  - line replaceable unit (LRU), 134
  - livelock, 181
  - load testing, 247
  - long-duration reliability testing, 190–193
  - loop coverage, 247
  - lower-level requirements, 159
  - low-level integration testing, 74–75, 89–90
  - low-level regression testing, 234–235
  - low-priority defects, 81
  - Low Rate of Initial Production (LRIP) phase, 175
  - Low Rate of Initial Production (LRIP) system components, 121
  - LRIP components. *See* Low Rate of Initial Production (LRIP) system components
  - LRU. *See* line replaceable unit
- M**
- maintenance master schedule, 140
  - maintenance organizations, 141, 236–237
  - maintenance plans, 139
  - maintenance projects, 224–225
  - major defects, 42
  - management, 52–55
    - conflict of interest, 63
    - external pressures, 15
    - funding to fix defects, 62
    - “go to market” attitude, 56
    - highly risk averse, 56, 58
    - ignoring test lessons learned, 15
    - independent test team, 63
    - measuring test program productivity, 60
    - meeting schedule, budget, and functionality targets, 62
    - negative relationships, 55
    - operational testing, 101
    - professional testers, 70, 73
    - risk, 57
    - risk repository, 57, 58
    - scapegoats, 62
    - scope and complexity of testing, 70
    - sequential development cycle, 95
    - staffing testing team, 43, 73
    - system testing, 43
    - testing as cost center, 53
    - test metrics, 15, 60
    - test-related risk management, 15
    - unimportance of testing, 52
  - management chains, 66–67
  - management-related testing pitfalls, 11, 14–15
    - Inadequate Test Metrics (GEN-MGMT-4), 15, 59–61
    - Inadequate Test-Related Risk Management (GEN-MGMT-3), 57–58
    - Inadequate Test Resources (GEN-MGMT-1), 14, 52–54
    - Inappropriate External Pressures (GEN-MGT-2), 54–56
    - Inconvenient Test Results Ignored (GEN-MGMT-5), 61–63
    - Test Lessons Learned Ignored (GEN-MGMT-6), 64–65
  - managerial independence, 67
  - managers
    - decreasing testing effort, 137
    - false sense of security, 45
    - not believing bad test results, 62
    - testing occurring late in life cycle, 97
  - mandatory behaviors, 153
  - mandatory postconditions, 153
  - mandatory quantitative thresholds, 153
  - manpower requirements, 203
  - manual regression testing, 226
  - manual testing, 31, 109
    - over-reliance on, 17, 106–108
  - master plan, 40
  - master schedule
    - adequate time for testing, 41
    - automated regression testing, 227–228
    - evaluating test assets, 91
    - incorrect requirements, 156
    - obsolete requirements, 149
    - operational testing, 101
    - regression testing, 230
    - system of systems (SoS) testing, 217–219
    - test asset maintenance, 93
  - McCabe’s Complexity, 129, 133
  - memorandum of understanding (MOU), 236, 237
  - message-oriented middleware (MOM), 111
  - messages, 176
  - metadata, 247
  - middleware, 110–111
  - milestone reviews, 39
  - Military-off-the-Shelf (MOTS) software, 32, 108
  - mishap analysis, 199
  - mishap testing, 197
  - missing requirements, 150–152
  - Missing Requirements (GEN-REQ-3), 19, 100, 149, 150–152, 220
    - characteristic symptoms, 150
    - description, 150
    - potential applicability, 150
    - potential causes, 151
    - potential negative consequences, 150
    - recommendations, 151–152
    - related pitfalls, 152
  - mission analysis, 81
  - mission-critical defects, 60, 80
  - mission-critical software, 78–79, 123
  - mission-critical system components, 78
  - mission-thread modeling, 210–211
  - misuse analysis, 202
  - misuse cases, 200
  - misuser analysis, 199, 202
  - mobile devices and apps, 120
  - mode-based testing, 31
  - modeling system engineering, 2–4
  - models, not maintaining, 139–140
  - MOM. *See* message-oriented middleware
  - MOTS. *See* Military-off-the-Shelf (MOTS) software
  - MOU. *See* memorandum of understanding
  - multiple target platforms, 110–112
- N**
- National Institute of Standards & Technology (NIST), 7–8
  - negative test results, 61–63
  - network scanning, 201
  - network simulation and testing, 121
  - NIPRNet. *See* Non-classified Internet Protocol Router Network
  - Non-classified Internet Protocol Router Network (NIPRNet), 118
  - nonfunctional requirements, 83–84
  - No Operational Testing (GEN-PRO-13) characteristic symptoms, 100
    - description, 100
    - potential applicability, 100
    - potential causes, 101
    - potential negative consequences, 100
    - recommendations, 101–102
    - related pitfalls, 102
  - normal behavior, 44, 46, 104
  - normal paths, 151
  - normal use case paths, 45
  - No Separate Test Planning Documentation (GEN-TPS-1), 13, 29–31, 94, 211, 213, 228, 231, 234
    - characteristic symptoms, 29
    - description, 28–29
    - potential applicability, 29
    - potential causes, 30
    - potential negative consequences, 29–30
    - recommendations, 30–31
    - related pitfalls, 31
- O**
- object request brokers (ORBS), 111
  - Obsolete Requirements (GEN-REQ-2) characteristic symptoms, 147
    - description, 147
    - potential applicability, 147
    - potential causes, 148
    - potential negative consequences, 148
    - recommendations, 148–149
    - related pitfalls, 149
  - OJT. *See* On-the-Job Training
  - on-device testing, 121
  - One-Size-Fits-All Testing (GEN-PRO-2), 16, 77–79, 82, 104
  - Only Functional Regression Testing (TTS-REG-6), 24, 237–239
    - characteristic symptoms, 238
    - description, 237
    - potential applicability, 237
    - potential causes, 238
    - potential negative consequences, 238
    - recommendations, 238–239
    - related pitfalls, 239

- Only Low-Level Regression Tests (TTS-REG-4), 24, 234-235
  - characteristic symptoms, 234
  - description, 234
  - potential applicability, 234
  - potential causes, 234-235
  - potential negative consequences, 234
  - recommendations, 235
  - related pitfalls, 235
- On-the-Job Training (OJT), 71
- open source software, 32, 108
- Open Systems Interconnection (OSI) model, 186
- operating systems, 110-111
- operational acceptance testing, 104
- operational environment, 118
- operationally relevant testing, 247
- operational testing, 247
- operational testing (OT), 31, 100-102, 121
  - defects causing failures to meet quality attributes, 59-60
  - target environment, 114
- Oracle, 93
- ORBS. *See* object request brokers
- organizations
  - incomplete test planning, 32
  - success-oriented culture, 45
- OSI model. *See* Open Systems Interconnection (OSI) model
- outputs, 145, 227
- outsourcing and testing, 69, 76
- Over-Reliance on Manual Testing (GEN-TTE-1), 17, 106-108, 110, 228, 231, 234
  - characteristic symptoms, 106-107
  - description, 106
  - potential applicability, 106
  - potential causes, 107
  - recommendations, 108
  - related pitfalls, 108
- Over-Reliance on Testing Tools (GEN-TTE-2), 17, 104, 108-110
  - characteristic symptoms, 108-109
  - description, 108
  - potential applicability, 108
  - potential causes, 109
  - potential negative consequences, 109
  - recommendations, 109-110
  - related pitfalls, 110
- P**
  - parallel development process, 127-128
  - password cracking, 201
  - path coverage, 247
  - PBIT. *See* Periodic BIT
  - PDL. *See* program design language
  - PDL models. *See* Program Design Language (PDL) models
  - penetration testing, 200, 247
  - performance testing, 21, 188-190
  - Periodic BIT (PBIT), 171, 172
  - personnel requirements, 203
  - PHM. *See* prognostics and health management (PHM) function
  - physical environment, 120
  - pitfalls, xiv, xviii, 10
    - categorizing, xix, 11
    - characteristic symptoms, 12
    - common negative consequences, 25-26
    - communication about, xix
    - descriptions, xiv, 12
    - detection recommendations, 27
    - first symptoms of, 12
    - general recommendations, 26-28
    - general testing pitfalls, 11, 13-20
    - management-related testing pitfalls, 14-15
    - metrics, 27
    - prevention recommendations, 26-27
    - reaction recommendations, 27-28
    - related, 12
    - repeating, 64
    - reporting occurrence, 28
    - residual defects delivered, 26
    - severity of consequences, 26
    - specifications, 11-12
    - staffing pitfalls, 15-16
    - testing process, 26
    - test planning and scheduling pitfalls, 13-14
    - test process documentation, 28
    - test-process pitfalls, 16-17
    - test-type-specific pitfalls, 11, 20-24
    - training, 27
    - training on, 27
    - treating as risks, 27, 28
  - platform-specific defects, 110
  - platform-specific test tools, 111
  - PMS. *See* Project Master Schedule
  - poor fidelity, 118-122
  - Poor Fidelity of Test Environments (GEN-TTE-6), 112, 114, 118-122, 124, 175
    - characteristic symptoms, 118
    - description, 118
    - potential applicability, 118
    - potential causes, 119-121
    - potential negative consequences, 119
    - recommendations, 121-122
    - related pitfalls, 122
  - port scanning, 201
  - postconditions, 1, 32, 90, 92, 103, 153, 227, 247, 249, 250, 262, 267
  - potential schedule overruns, 26
  - PowerUp BIT (PupBIT), 171, 172
  - preconditions, 1, 32, 90, 92, 103, 153, 227, 247, 249, 250, 262, 267
  - predicate coverage, 244
  - primary test metrics, 59
  - primary use case paths, 44, 103
  - priority inversion, 181
  - private network, 118
  - privilege escalation, 201
  - process documentation
    - black-box system testing, 87, 88
    - testing nonfunctional requirements, 83
  - process engineer, 77
  - process requirements
    - black-box system testing, 87
    - testing critical and non-critical components, 78
    - unit and integration testing, 85
  - product line component developers, 141
  - products
    - not under configuration management (CM), 126-128
    - testing before they are ready to test, 88-90
    - too immature for testing (GEN-PRO-7), 16
  - professional testers, 70-71, 73
  - Prognostics and Health Management (PHM) functions or subsystems, 171, 172, 173, 209
  - Program Design Language (PDL), 4, 133
  - programmers, 73
  - programming languages
    - as program design language (PDL), 4
    - testability guidelines, 130
  - project chief engineer, 77
  - project-level risks, 57
  - project manager, 66
  - Project Master Schedule (PMS), 76
    - early involvement of testers, 96
    - incorporating testing, 77
    - testing as phase, 94
    - testing late in development cycle, 42
  - project planning documentation
    - agile development, 96
    - statement of testing goals, 45-46
  - projects
    - budget, schedule, and staff, 29, 55
    - canceling, 55
    - contractual relationships, 29
    - inadequate defect tracking, 23
    - master schedule, 39
    - responsibility for finding and fixing defects, 224-225
    - system of system (SoS) testing, 23
    - testers in initial staffing, 77
    - test-related risks, 57-58
  - project-specific glossary, 145, 146
  - project-wide test-case-selection criteria, 103
  - proprietary test tools, 108
  - prototypes, 121
  - public network, 118
  - PupBIT. *See* PowerUp BIT
- Q**
  - quality, 62-63, 248
    - characteristics and attributes, 151
    - little or no testing on, 83
  - quality assurance, xix, 58
    - as-performed testing process, 92
  - quality-assurance plans, 91
  - quality attributes, 59-60, 248
  - quality-based testing, 31-32
  - quality control, xix, 58
  - quality-engineering plans, 91
  - quality model, 152
  - quality requirements, 78, 150, 151, 248
    - little or no testing, 87, 107
    - relevant, 103, 104

- testing, 41, 83, 179
  - verifying, 83
- R**
- race conditions, 181, 182
  - rainy-day path, 31, 103, 152, 167, 263
  - range verification, 46
  - Red Teaming, 200
  - redundant testing, 53
  - regression testers and non-testing as-sets, 124
  - regression testing, 31, 107, 248
    - automating, 23, 41, 226–227, 229–231
    - change-impact analysis, 230, 232, 233
    - change-request impact, 39
    - false-negative results, 139
    - functionality-related defects, 24, 237–239
    - high-level, 234–235
    - inadequate scope, 23, 125, 231–234
    - insufficient number of tests for, 225–228
    - manual, 226
    - not performed, 228–231
    - only low-level, 24, 234–235
    - productive decreasing, 93
    - resources unavailable for mainte-nance, 236–237
  - Regression Testing Not Performed (TTS-REG-2), 23, 49, 100, 234
    - characteristic symptoms, 228
    - description, 228
    - potential applicability, 228
    - potential causes, 229–230
    - potential negative consequences, 229
    - recommendations, 230–231
    - related pitfalls, 231
  - regression testing pitfalls, 11, 23–24, 225–239
    - Inadequate Scope of Regression Test-ing (TTS-REG-3), 23, 231–234
    - Inadequate Regression Test Automata-tion (TTS-REG-1), 23, 225–228
    - Only Functional Regression Testing (TTS-REG-6), 24, 237–239
    - Only Low-Level Regression Tests (TTS-REG-4), 24, 234–235
    - Regression Testing Not Performed (TTS-REG-2), 23, 228–231
    - Test Resources Not Delivered for Maintenance (TTS-REG-5), 24, 236–237
  - related pitfalls, 12
  - reliability testing
    - inadequate, 21
    - little or none performed, 190–193
    - long-duration, 190–193
  - REP. *See* Requirements Engineering Plan
  - replacement management problem, 63
  - request for proposal (RFP), 27
  - requirements, 248
    - abnormal conditions, 150
    - ambiguous, 19, 144–147
    - architecturally significant, 4
    - black-box testing, 149
    - changing, 148, 157
    - deleted, 148
    - ensuring testability, 97
    - high-level policies as, 145
    - identifying defects, 8
    - improperly derived, 19–20, 159–160
    - inadequate for system of systems (SoS), 23
    - incomplete, 19, 152–154
    - incorrect, 19, 154–156
    - informal changes, 147
    - interfaces, 172
    - lower-level, 159
    - mandatory behaviors, 153
    - mandatory postconditions, 153
    - mandatory quantitative thresholds, 153
    - missing, 19, 150–152
    - not tested, 98
    - not tracing to test assets, 92
    - obsolete, 19, 147–149, 157
    - preconditions, 153
    - properly validated, 156
    - requirements-review checklist, 145
    - stabilizing, 158
    - system behavior during non-opera-tional modes, 150
    - system of systems (SoS) testing, 219–220
    - system under test (SUT), 7
    - system under test (SUT) violating, 7
    - test hooks, 172
    - tracings between tests and, 92
    - tracing to tests or test cases, 20
    - trigger events, 153
    - undocumented, 19, 139, 150–152
    - unstable, 156–158
    - verification methods, 20, 161–162
    - verifying, 87, 88
  - requirements analysis, 103
  - requirements-based system of system (SoS) testing, 219–220
  - requirements-based testing, 150
  - requirements churn, 19, 156–158
  - Requirements Churn (GEN-REQ-6), 149, 156–158
    - characteristic symptoms, 157
    - description, 156
    - potential applicability, 156
    - potential causes, 157
    - potential negative consequences, 157
    - recommendations, 158
    - related pitfalls, 158
  - requirements documents, 149, 152, 154
  - requirements-driven testing, 109
  - Requirements Engineering Plan (REP), 163
  - requirements engineering process, 155, 160, 162
  - requirements engineering team, 151–152
    - requirements repository, 148
    - senior test engineer, 145
  - requirements engineers, 141, 167
  - requirements-level tests, 149
  - requirements management tool, 248
  - requirements metadata, 248
  - requirements models, 3, 152
  - requirements prototypes, 3
  - requirements-related testing pitfalls, 11, 19–20, 143–164
    - Ambiguous Requirements (GEN-REQ-1), 19, 144–147
    - Improperly Derived Requirements (GEN-REQ-7), 19–20, 159–160
    - Incomplete Requirements (GEN-REQ-4), 19, 152–154
    - Incorrect Requirements (GEN-REQ-5), 19, 154–156
    - Lack of Requirements Trace (GEN-REQ-9), 20, 162–164
    - Missing Requirements (GEN-REQ-3), 19, 150–152
    - Obsolete Requirements (GEN-REQ-2), 19, 147–149
    - Requirements Churn (GEN-REQ-6), 19, 156–158
    - Verification Methods Not Properly Specified (GEN-REQ-8), 20, 161–162
  - requirements repository, 146
    - missing requirements, 152
    - not updating, 145, 148
    - reviewing, 149, 154
    - test cases, 163
    - updating, 151, 153, 155, 160
    - verification methods, 161–162
  - requirements-review checklist, 153
  - requirements specification documents, 162
  - requirements specifications, 139–140
  - requirements teams, 76, 143
  - requirements trace, 248
  - residual defects, 45, 48, 85
    - delivered system, 60
    - faults and failures during opera-tional usage, 100
    - percentage found per unit time, 61
    - unacceptably large number of, 70, 123
  - resources
    - best-case scenarios, 53
    - estimates for testing, 53
    - insufficient, 80, 89
    - most-likely scenarios, 53
    - system of systems (SoS) testing, 23, 215–217
    - unavailable for maintenance, 236–237
    - worst-case scenarios, 53
  - response time, 188–190
  - return on investment (ROI)
    - automated testing, 107
    - testing, 60
  - reverse engineering, 201
  - RFP. *See* request for proposal
  - risk management, 58
  - risk management plan, 57
  - risk mitigation approaches, xix
  - risk repository, 57–58
  - risks, 57–58
    - identifying, xix
    - treating pitfalls as, 27, 28

- robustness testing, 22, 193–196  
 ROI. *See* return on investment  
 root kits, 112
- S**
- safeguard analysis, 199  
 safeguards, 197, 199  
 safety certified components, 122  
 safety-critical defects, 60, 80  
 safety-critical software, 78–79, 123  
 safety-critical system components, 78  
 safety (hazard) analysis, 81  
 safety risk analysis, 199  
 safety-specific testing, 197  
 safety testing, 22, 197–199  
 SBIT. *See* Shutdown BIT  
 schedules, 63
  - agile development, 96
  - evolutionary development cycle, 96
  - inadequate estimates, 51
  - pressures causing corners to be cut, 89
  - size, 29
  - system of systems (SoS) testing, 217–219
  - test resources, 52, 54
 scheduling documents, 96  
 Scrum sprint, 157  
 SDC. *See* System Development Cycle  
 SDP. *See* Software Development Plan  
 SDP. *See* System Development Plan  
 Secure Internet Protocol Router Network (SIPRNet), 118, 121  
 security analysis, 200–203  
 security certified components, 122  
 security-critical defects, 60, 80  
 security-critical software, 78–79, 123  
 security-critical system components, 78  
 security-related requirements, 201–202  
 security risk analysis, 202  
 security testing, 22, 200–203  
 security (threat) analysis, 81  
 self-documenting software, 133  
 self-monitoring, 20  
 self-tests, 172–173  
 SEMP. *See* System Engineering Management Plan  
 senior test engineer, 145  
 sensor drift, 120, 122, 248  
 sequential waterfall development cycle, 4, 43
  - management, 95
  - testing as phase late in, 17, 94–96
 shared memory, 181  
 SharePoint sites, xix  
 short-duration increment, 157  
 Shutdown BIT (SBIT), 171, 172, 244  
 simulator, 248  
 Single V model, 3, 42–43  
 SIPRNet. *See* Secure Internet Protocol Router Network  
 SLE. *See* System Level Exerciser  
 SMEs. *See* subject-matter experts  
 soak tests, 192  
 software
  - concurrency faults and failures, 181–183
  - current increment, 80
  - defect removal, xv
  - degrading as capacity limits approach, 178–180
  - error, fault, failure, and environmental tolerance, 193–196
  - failure meet testability requirements, 169
  - functioning without failure, 190–193
  - identifying defects, 8
  - incorrect versioning, 126–127
  - insufficient resources, 78
  - internationalization testing, 183–185
  - interoperability testing, 185–188
  - McCabe's complexity, 129
  - multiple people responsible for testing, 68–69
  - not tested, 98, 127
  - performance quality attributes, 188–190
  - removing test hooks and interfaces, 130
  - safe from causing accidental harm, 197–199
  - secure from causing or suffering malicious harm, 200–203
  - self-documenting, 133
  - test assets, 124–126
  - unavailable components, 173–175
 software developers, 167–168  
 software development, 8, 79
  - independent technical assessments (ITAs), xvii
 Software Development Plan (SDP), 29, 30, 76
  - capacity testing, 180
  - high-level overviews of testing, 77
  - integration testing, 177
 Software Engineering Institute (SEI), iv, xiv  
 software engineers, 76  
 software-internal self-tests, 172–173  
 software-only test environments, 114  
 software planning documents, 96  
 software platform and poor fidelity, 119  
 software simulation, 114  
 software under test (SUT)
  - authorized attacks, 200
  - changing, 16
  - critical nature of, 29, 66
  - improving, 46
  - inconsistencies between current versions, 126
  - incremental and iterative updates
    - not tested, 127
  - obsolete requirements, 19, 147–149
  - operational environment, 118
  - poor fidelity of test environments or test beds, 18
  - test drivers, 118
  - test stubs, 118
  - test tools, 118
 solutions, 27  
 SoS. *See* system of systems  
 SoS Development Plan (SoSDP), 212  
 SoSDP. *See* SoS Development Plan  
 SoS-integration-level defects, 214  
 SoS-level requirements, 224–225  
 SoS Testing Not Properly Scheduled (TTS-SoS-4), 41, 217–219
  - characteristic symptoms, 217
  - description, 217
  - potential applicability, 217
  - potential causes, 218
  - potential negative consequences, 218
  - recommendations, 218–219
  - related pitfalls, 219
 source documents, 139–140  
 Source Documents Not Maintained (GEN-COM-4), 19, 133, 139–140
  - characteristic symptoms, 139
  - description, 139
  - potential applicability, 139
  - potential causes, 139
  - potential negative consequences, 139
  - recommendations, 139–140
  - related pitfalls, 140
 SOW. *See* statement of work  
 special engineering performance engineers, 189  
 specialized engineers, 83  
 specialized testing, 78, 83  
 specialty engineering capacity engineers, 179  
 specialty engineering reliability engineers, 192  
 specialty engineering robustness engineers, 195  
 specialty engineering safety engineers, 198  
 specialty engineering security engineers, 201  
 specialty engineering testing, 59, 79  
 specialty engineering testing pitfalls, 11, 21–22, 60, 177–206
  - Inadequate Capacity Testing (TTS-SPC-1), 21, 178–180
  - Inadequate Concurrency Testing (TTS-SPC-2), 21, 181–183
  - Inadequate Internationalization Testing (TTS-SPC-3), 21, 183–185
  - Inadequate Interoperability Testing (TTS-SPC-4), 21, 185–188
  - Inadequate Performance Testing (TTS-SPC-5), 21, 188–190
  - Inadequate Reliability Testing (TTS-SPC-6), 21, 190–193
  - Inadequate Robustness Testing (TTS-SPC-7), 22, 193–196
  - Inadequate Safety Testing (TTS-SPC-8), 22, 197–199
  - Inadequate Security Testing (TTS-SPC-9), 22, 200–203
  - Inadequate Usability Testing (TTS-SPC-10), 22, 203–206
 specialty engineering usability or human factors engineers, 205  
 SpecTRM-RL, 3  
 SQL Server, 93  
 staff, 29, 63  
 staffing pitfalls, 11, 15–16, 65–75

- Developers Responsible for All Testing (GEN-STF-4), 15, 72-73
- Inadequate Testing Expertise (GEN-STF-3), 15, 69-72
- Lack of Independence (GEN-STF-1), 15, 66-67
- Testers Responsible for All Testing (GEN-STF-5), 16, 74-75
- Unclear Testing Responsibilities (GEN-STF-2), 15, 68-69
- stakeholder involvement and commitment pitfalls, 11, 14, 44-51
- Lack of Stakeholder Commitment to Testing (GEN-SIC-3), 49-51
- Unrealistic Testing Expectations (GEN-SIC-2), 47-49
- Wrong Testing Mindset (GEN-SIC-1), 44-47
- stakeholders
  - commitment to testing, 14, 44-51
  - communication regarding testing, 140-143
  - false sense of security, 40, 48, 60, 64, 70
  - ignoring testers and test results, 49-50
  - independent reporting, 67
  - not understanding testing, 50, 53
  - professional testers, 73
  - requirements changes, 149
  - requirements documents, 146
  - resources necessary for testing, 50
  - test assets, 50, 93
  - testing, 99
  - testing expertise, 69-71
  - test team's lack of independence, 66
  - unrealistic testing expectations, 14, 47-49
  - wrong mindset, 44-47
- STAMP. *See* Systems-Theoretic Accident Model and Processes
- starvation, 181
- statecharts, 3
- statement coverage, 248
- statement of work (SOW), 27, 236, 237
  - architectural and design models and documentation, 132
  - test assets, 125
- static testing, 32
- static verification methods, 8-9
- status reports, 33
- STPs. *See* System Test Plans
- stress testing, 248
- structural testing, 248
- subject-matter experts (SMEs), 77, 141
  - requirements documents, 146
  - test assets, 50
- subsystem design documents, 131, 133
- subsystem-level testing, 159
- subsystem requirements, 159
- subsystems
  - containing defect, 134
  - criticality, 80
  - risks, 80
  - testing source, 32
  - unnecessary interfaces and dependencies, 129
  - subsystems developers, 141
  - sunny-day path, 31, 45, 47, 103, 152, 167, 263
  - sunny-day testing, 249
  - sustainment organizations, 141
  - SUT. *See* software under test; system under test
  - Sybase, 93
  - system
    - accreditation testing, 115
    - behavior, 44
    - capacity limits, 178-180
    - causing accidental harm, 197-199
    - causing or suffering malicious harm, 200-203
    - certified, 115
    - concurrency faults and failures, 181-183
    - end-to-end support for missions, 209-211
    - end-to-end testing, 22
    - error, fault, failure, and environmental tolerance, 193-196
    - failing to meet requirements, 87, 159, 169
    - functioning without failure, 21, 190-193
    - inconsistencies between current versions, 126
    - independent testing, 66
    - interfacing and collaborating with other systems, 21
    - internationalization testing, 21, 183-185
    - interoperability testing, 185-188
    - meeting requirements, 97
    - multiple people responsible for testing, 68-69
    - no alpha or beta testing, 100
    - non-functional requirements, 78
    - no operational testing, 17, 100-102
    - obsolete requirements, 19, 147-149
    - operational environment, 118
    - performance quality attributes, 21, 188-190
    - prognostics and health management (PHM) function, 171, 172, 173
    - removing test hooks and interfaces, 130
    - residual defects, 40, 62, 70
    - self-monitoring capabilities, 171-172
    - test assets, 124-126
    - test hooks remaining, 22
    - testing, xviii
    - test metrics, 55
    - unknown defects, 98
    - version or variant, 134
  - system architect, 77
  - system architecture document, 131, 133
  - system defects, 4, 98
  - system developers, 141
  - system development
    - documenting testing, 77
    - project-specific, specialized testing information, 79
    - testers responsibility for testing during, 16
    - verification and validation methods, 8
  - System Development Cycle (SDC), 76
  - System Development Plan (SDP), xix, 27
    - prioritizing tests, 81
    - requirements section, 163
  - system development projects, xvii, 224-225
  - system engineering, 2-4, 76-77
  - system engineering documentation, 76
  - System Engineering Management Plan (SEMP), xix, 27, 29, 30, 76, 249
    - capacity testing, 180
    - high-level overviews of testing, 77
    - integration testing, 177
    - requirements section, 163
    - system of systems (SoS), 212
  - system engineers, 76
  - system-external actors, 118
  - system-external interfaces, 187
  - system integrator developers, 141
  - system-internal interface documents, 131, 133
  - system-internal self-tests, 172-173
  - system-level defects, 95
  - System Level Exerciser (SLE), 190
  - system-level functional testing, 209-211
  - system-level risks, 58
  - system-level testing
    - as integration testing, 175-177
    - training, 88
    - unit defects, 59
  - system maintenance, 8
  - system of systems (SoS), 23
    - developers, 141
    - System Engineering Management Plan (SEMP), 212
    - test planning, 212-213
    - test planning documentation, 213-215
  - system of systems (SoS) testing
    - defects responsibilities, 23
    - end-to-end testing responsibilities, 213-215
    - inadequate defect tracking, 23, 222-224
    - inadequate support for, 220-222
    - master schedules, 217-219
    - requirements, 219-220
    - resources, 23, 215-217
    - responsibility for finding and fixing defects, 224-225
    - system projects inadequate support for, 23
    - tasks, 212-213
    - unclear responsibilities, 23
  - system of systems (SoS) testing pitfalls, 11, 22-23, 211-225
  - Finger-Pointing (TTS-SoS-8), 23, 224-225
  - Inadequate Defect Tracking Across Projects (TTS-SoS-7), 23, 222-224

- system of systems (SoS) testing pitfalls, (*cont.*)
    - Inadequate Resources for SoS Testing (TTS-SoS-3), 23, 215-217
    - Inadequate SoS Requirements (TTS-SoS-5), 23, 219-220
    - Inadequate SoS Test Planning (TTS-SoS-1), 23, 212-213
    - Inadequate Support from Individual System Projects (TTS-SoS-6), 23, 220-221
    - SoS Testing Not Properly Scheduled (TTS-SoS-4), 23, 217-219
    - Unclear SoS Testing Responsibilities (TTS-SoS-2), 23, 213-215
  - system planning documents, 96
  - system-specific risks, 58
  - Systems-Theoretic Accident Model and Processes (STAMP), 199, 202
  - system testing, 60, 249
    - as acceptance tests, 105
    - defects, 59-60
    - repeating white-box tests, 104
    - requirements verified by, 88
    - unnecessary, 87
  - System Testing as Integration Testing (TTS-INT-4), 20, 175-177
    - characteristic symptoms, 176
    - description, 175
    - potential applicability, 175
    - potential causes, 176-177
    - potential negative consequences, 176
    - recommendations, 177
    - related pitfalls, 177
  - system testing pitfalls, 11, 22, 206-211
    - Inadequate End-to-End Testing (TTS-SYS-3), 22, 209-211
    - Lack of Test Hooks (TTS-SYS-2), 22, 208-209
    - Test Hooks Remain (TTS-SYS-1), 22, 206-208
  - System Test Plans (STPs), xix, 28
  - system under test (SUT), 7
    - changing (GEN-PRO-9), 16
    - critical nature of, 29
    - improving, 1, 46
    - poor fidelity of test environments or test beds (GEN-TTE-6), 18
    - quality, 1
- T**
- TaaS. *See* Testing as a Service
  - tables, 93
  - target environments, 116
  - Target Platform Difficult to Access (GEN-TTE-4)
    - characteristic symptoms, 112
    - description, 112
    - potential applicability, 112
    - potential causes, 113
    - potential negative consequences, 113
    - recommendations, 113-114
    - related pitfalls, 114
  - target platforms, 112-114, 126
    - difficulty accessing, 17
    - software not tested on, 127
    - testing applications executing on multiple, 110-112
  - TDD. *See* Test Driven Development
  - teams
    - as-available or as-needed testing, 71
    - evaluating testing, 91
    - geographically distributed, 142
    - inadequate communication regarding testing, 141
    - technical interdependence, 67
    - technical interchange meetings (TIMs), 143
    - technical lead, 77
  - TEMPs. *See* Test and Evaluation Master Plans
  - testability, 20, 249
  - testability guidelines, 130
  - testable requirements, 98
  - Test and Evaluation Master Plan (TEMP), xix, 28, 250
  - test assets, 249
    - development contract, 125
    - identifying and fixing defects, 91
    - inadequate maintenance, 16, 92-94
    - not adequately evaluated, 16, 90-92
    - not consistent with requirements, 93
    - not delivered, 18
    - not tracing requirements to, 92
    - statement of work (SOW), 125
    - system or software delivered without, 124-126
    - tool support for, 93, 94
    - undocumented, 136-137
    - verifying during testing, 91
  - Test Assets Not Delivered (GEN-TTE-8), 35, 63, 124-126, 228, 231, 234, 237
  - test automation. *See* automating testing
  - test beds, 92, 249
    - insufficient, 18, 114-117
    - poor fidelity, 18, 118-122
    - verification, validation, accreditation, or certification, 90
  - test-case completion criteria, 190, 193, 196, 199, 203
  - test-case documents, 13, 38
  - Test-Case Documents as Test Plans (GEN-TPS-4)
    - characteristic symptoms, 38
    - description, 37
    - potential applicability, 37
    - potential causes, 38
    - potential negative consequences, 38
    - recommendations, 39
  - test cases, 227, 249
    - architecture coverage, 132
    - automating creation, 109, 110
    - code coverage, 132
    - cross cutting requirements, 19-20, 159-160
    - design coverage, 132
    - design method that implies, 103
    - developing, 96, 97, 107
    - false-negative and false-positive results, 19, 154-156
    - finding defects, 90
    - inadequate, 176
    - incomplete documentation, 141
    - incomplete or incorrect, 19, 152-154
    - little or no inspections, walk throughs, or reviews, 90
    - missing, 19-20, 159-160
    - non-stable requirements, 19
    - normal behavior, 46
    - preconditions, 169, 170
    - rainy-day paths, 167
    - repeating same kind of, 17, 104-106
    - requirements analysis, 103
    - requirements coverage, 132
    - stable requirements, 158
    - subject-matter experts (SMEs), 90
    - sunny-day paths, 167
    - system and test failures, 88
    - test-case-selection criteria, 96
    - test-completion criteria, 96
    - test-type-specific, 105
    - tracing requirements, 162-164
    - unstable requirements, 156-158
    - wrong level of abstraction, 19-20, 159-160
  - test-case-selection criteria, 32, 103, 104, 105, 190, 193, 196, 199, 203, 227, 249
  - test communication pitfalls, 11, 18-19, 131-143
    - Inadequate Architecture or Design Documentation (GEN-COM-1), 18, 131-133
    - Inadequate Communication Concerning Testing (GEN-COM-5), 140-143
    - Inadequate Communication Regarding Testing (GEN-COM-5), 19
    - Inadequate Defect Reports (GEN-COM-2), 18, 134-136
    - Inadequate Test Documentation (GEN-COM-3), 19, 136-138
    - Source Documents Not Maintained (GEN-COM-4), 19, 139-140
  - test-completion criteria, 32, 99, 103, 104, 105, 109, 110, 227, 249
    - decision makers, 68
    - lower-level testing, 90
  - test-coverage criteria, 103, 104
  - test coverage levels, 99
  - test data
    - correctness, 104
    - describing, 32
    - developing, 107
    - imperfect replica of actual data, 118
    - inaccessible, 93
    - inadequate, 17
    - incomplete, 102-104
    - incorrect, 102-104
    - invalid, 102-104
    - listing, 32
    - normal "sunny-day" use case paths, 103
    - not migrated to new versions of database, 93
    - responsibility for, 68

- validity, 104
- test data set, 102
- test documentation, 136–138
- Test Driven Development (TDD), 4, 42, 165, 166, 168, 249
- test driver, 250
- test effectiveness metric, 61
- test engineers, 79, 250
- test entrance criteria, 32
- test-environment fidelity, 121
- test environments, 92, 122–123, 134, 250
  - configurations not identified, 126
  - emulating or simulating parts of operational environment, 118
  - evaluating fidelity of behavior, 121
  - excessive number of defects, 122–124
  - funding for, 116
  - hardware, 116, 117
  - inadequate planning for, 116
  - inadequate quality, 18
  - insufficient, 18, 114–117
  - multiple people competing for time on, 115
  - not under configuration control, 126
  - poor fidelity, 18, 118–122
  - software-only, 114
  - unavailable components, 173–175
  - verification, validation, accreditation, or certification, 90
- testers, 250
  - ambiguous requirements, 19
  - defect reports, 135
  - development and bug fixes, 71
  - easy tests, 80
  - engineering and evaluating requirements, 96
  - evidence-based cost/effort models, 53–54
  - expertise, 52
  - external pressures, 54–56
  - failed tests, 63
  - finding defects, 54, 60
  - high-level testing, 73
  - inadequate communication regarding testing, 19, 140–143
  - insufficient resources, 74
  - low productivity, 40, 70
  - manual testing, 17
  - miscommunication with stakeholders, 70
  - morale suffers, 25–26, 55
  - not certified for testing, 70
  - not involved early in project, 17, 96–98
  - On-the-Job Training (OJT), 71
  - process engineer, 77
  - project chief engineer, 77
  - quality, 74
  - relevant changes, 142
  - reporting defects, 45
  - reporting independently of project manager, 56
  - required qualifications, 71
  - requirements engineering, 97
  - resources, 73
  - system failure, 48
  - testing tools, 17
  - training, 70
  - undocumented requirements, 19
  - units and low-level components, 74
  - working long hours, 40, 52
  - wrong mindset, 44–47
- Testers Not Involved Early (GEN-PRO-11), 43, 96–98
  - characteristic symptoms, 96
  - description, 96
  - potential applicability, 96
  - potential causes, 97
  - potential negative consequences, 96–97
  - recommendations, 97
  - related pitfalls, 98
- Testers Responsible for All Testing (GEN-STF-5), 16, 73, 74–75, 130
  - characteristic symptoms, 74
  - description, 74
  - potential applicability, 74
  - potential causes, 74
  - potential negative consequences, 74
  - recommendations, 74–75
  - related pitfalls, 75
- test evaluations, 91
- test-exit criteria, 32
- test facilities, 68
- test hooks, 196, 250
  - failing to remove after testing, 206–208
  - lack of, 22, 208–209
  - requirements, 172
  - security vulnerabilities, 207
- Test Hooks Remain (TTS-SYS-1), 22, 207–209
  - characteristic symptoms, 207
  - description, 206
  - potential applicability, 206
  - potential causes, 207
  - potential negative consequences, 207
  - recommendations, 207–208
  - related pitfalls, 208
- testing, xviii, 250
  - abnormal behavior, 41
  - all performed the same way, 77–79
  - Application Lifecycle Management (ALM) tool repositories, xix
  - applications on multiple target platforms, 110–112
  - architecture models, 3
  - augmenting with other types of verification, 49
  - behavior of system, 44
  - best practices, 70
  - black-box-system, 16
  - bottlenecks, 74
  - completeness and rigor, 32, 79
  - concurrent, 95
  - cost, 60
  - cost center, 53
  - cost savings, 60
  - criticalness of, 7–9
  - defects, 8, 42, 47, 59, 68, 70
  - dependencies, 33
  - design-driven, 109
  - design models, 3
  - development cycle, 42–43
  - development process, 99
  - effectiveness, 25, 76, 89
  - efficiency, 25
  - encapsulated components, 172–173
  - environments, 32–33
  - evaluating, 91
  - evidence-based estimates, 41
  - exceptional behavior, 44
  - execution of work products, 2–3
  - exhaustive, 47
  - expertise and experience, 72
  - failure-tolerant behavior, 44
  - fault-tolerant behavior, 44
  - functionality overemphasized, 16
  - glossary, 33
  - goals, 1, 45–46
  - hardware emulation, 114
  - human error, 107
  - incomplete, 17
  - incremental, 95
  - as independent activity, 76
  - input data, 44
  - iterative, 95
  - iteratively, incrementally, and parallel, 43
  - lessons learned, 64–65
  - levels, 31
  - lower-level types, 86
  - multiple types being performed, 104
  - normal versus abnormal behavior, 31, 44
  - one-size-fits-all, 16
  - on-the-fly, 99
  - outsourcing, 69, 76
  - as phase, 94–96
  - as a phase, 17
  - primary goal, 1
  - priorities, 80
  - prioritizing, 80–82
  - process of, 32
  - product immaturity, 89
  - quality, 60
  - quality requirements, 41
  - redundant, 53
  - relevant documents, 33
  - requirements, 85
  - requirements-driven, 109
  - requirements models, 3
  - residual defects, 48
  - resources, 50, 53, 90
  - responsibilities, 68–69
  - return on investment (ROI), 60
  - schedule, 33
  - scope, 31
  - secondary goals, 1
  - SharePoint sites, xix
  - software projects, xiii
  - software simulation, 114
  - specialty engineering activity, 76
  - stakeholder, 14
  - subsystem source, 32
  - tasks, 32
  - techniques, 32

- testing (*cont.*)
  - test-case-selection criteria, 32, 105
  - test entrance criteria, 32
  - test-exit or -completion criteria, 32
  - test-suspension and -resumption criteria, 32
  - training, 46
  - unimportance of, 62
  - unrealistic expectations, 14, 47-49
  - unsophisticated, xiii
  - verifying system, 45
  - V Models, 2-4
  - wikis, xix
  - work products, 32
  - wrong lessons learned, 64
  - wrong mindset, 44-47
- Testing and Engineering Processes Not Integrated (GEN-PRO-1), 16, 31, 43, 76-77, 98, 143, 166
  - characteristic symptoms, 76
  - description, 76
  - potential applicability, 76
  - potential causes, 76
  - potential negative consequences, 76
  - recommendations, 77
  - related pitfalls, 77
- Testing as a Phase (GEN-PRO-10), 43, 94-96, 98
  - characteristic symptoms, 94-95
  - description, 94
  - potential applicability, 94
  - potential causes, 95
  - potential negative consequences, 95
  - recommendations, 95-96
  - related pitfalls, 96
- Testing as a Service (TaaS), 117, 120
- Testing at the End (GEN-TPS-6), 42-43, 96, 98, 100, 147, 166
  - characteristic symptoms, 42
  - description, 42
  - potential applicability, 42
  - potential causes, 43
  - potential negative consequences, 42
  - recommendations, 43
  - related pitfalls, 43
- testing churn, 157
- testing documentation and agile development, 136, 137
- Testing Does *Not* Drive Design and Implementation (TTS-UNT-1), 130, 165-168
  - characteristic symptoms, 165, 167
  - description, 165, 167
  - potential applicability, 165, 167
  - potential causes, 165-166, 167-168
  - potential negative consequences, 165, 167
  - recommendations, 166, 168
  - related pitfalls, 166, 168
- testing effectiveness, 250
- testing method, 250
- testing metrics, 60-61
- testing pitfalls, 10-11
- testing process
  - difficulty improving, 60
  - engineering process, 16
  - evaluation of, 58
  - inputs to, 41
  - lessons learned and, 64-65
  - quality assurance oversight, 58
  - system engineering process, 76-77
  - updating, 26
- testing process documents, 93
- testing programs, 29
- testing risks, 57, 68
- testing stakeholder. *See* stakeholders
- testing tools
  - new, unfamiliar, 69
  - over-reliance on, 17, 108-110
- testing wikis, 29
- test inputs, 92, 102, 250
  - incorrect, 145
- test interface, 207
- test laboratories or facilities, 92
  - insufficient, 18, 114-117
  - verification, validation, accreditation, or certification, 90
- Test Lessons Learned Ignored (GEN-MGMT-6), 15, 51, 64-65
  - characteristic symptoms, 64
  - description, 64
  - potential applicability, 64
  - potential causes, 64-65
  - potential negative consequences, 64
  - recommendations, 65
  - related pitfalls, 65
- test management, 93
- test manager, 56
- test metrics, 27, 33
  - inadequate, 15, 59-61
  - not reflecting true state of system, 55
  - primary, 59
- test oracle, 250
- test organization manager, 66
- test output, 251
- test planning
  - including all testing tasks, 53
  - incomplete, 13, 31-35
  - presentations, 29
  - at project inception, 53
  - project-specific, specialized testing information, 79
  - testing nonfunctional requirements, 83
- test planning and scheduling pitfalls, 11, 13-14, 28
- test planning documentation, 13, 27, 251
  - agile development, 38, 96
  - automated regression testing, 227-228
  - black-box system testing, 87, 88
  - capacity testing, 179, 180
  - concurrency testing, 182, 183
  - defects, 90
  - developers responsibilities, 74
  - early involvement of testers, 96
  - evaluating, xix
  - evolutionary development cycle, 96
  - generic, boilerplate testing guidelines, 78
  - ignoring, 13, 35-37
  - integration testing, 177
  - internationalization testing, 184, 185
  - interoperability testing, 187
  - lessons learned and, 65
  - limitations of encapsulation, 209
  - never updated, 35
  - operational testing, 101
  - performance testing, 189
  - professional testers, 73
  - regression testing, 230, 238-239
  - reliability testing, 192
  - resources, 52, 54
  - responsibilities for testing, 69
  - risks not addressed in, 57
  - robustness testing, 194, 195, 196
  - safety testing, 198
  - security testing, 201
  - system-level testing, 88
  - system of systems (SoS), 213-215
  - test assets, 91, 93
  - testing as phase, 94
  - testing responsibilities, 68-69
  - usability testing, 204, 205, 206
- test planning presentations, xix
- test plans
  - incomplete, 78
  - no templates or content format standards, 38
  - prioritizing tests, 81
  - project- and system-specific information, 79
  - template, 68
  - test-case documents as, 13, 37-39
- testing metrics, 61
- Test Plans Ignored (GEN-TPS-3)
  - characteristic symptoms, 35
  - description, 35
  - potential applicability, 35
  - potential causes, 36
  - potential negative consequences, 35-36
  - recommendations, 37
  - related pitfalls, 37
- test process documentation, 28
  - automated regression testing, 227-228
  - regression testing, 230
- test process pitfalls, 11, 16-17, 75-106
  - Black-Box System Testing Overemphasized (GEN-PRO-5), 16, 85-86
  - Black-Box System Testing Underemphasized (GEN-PRO-6), 86-88
  - Functionality Testing Overemphasized (GEN-PRO-4), 16, 82-84
  - Inadequate Evaluation of Test Assets (GEN-PRO-8), 16, 90-92
  - Inadequate Maintenance of Test Assets (GEN-PRO-9), 16, 92-94
  - Inadequate Test Data (GEN-PRO-14), 17, 102-104
  - Inadequate Test Prioritization (GEN-PRO-3), 16, 80-82
  - Incomplete Testing (GEN-PRO-12), 17, 98-100
  - No Operational Testing (GEN-PRO-13), 17, 100-102
  - One-Size-Fits-All Testing (GEN-PRO-2), 16, 77-79

- Products Too Immature for Testing (GEN-PRO-7), 16, 88-90
  - Testers Not Involved Early (GEN-PRO-11), 17, 96-98
  - Testing and Engineering Processes Not Integrated (GEN-PRO-1), 16, 76-77
  - Testing Treated as a Phase (GEN-PRO-10), 17, 94-96
  - Test-Type Confusion (GEN-PRO-15), 17, 104-106
  - test program, 59-60
  - test readiness criteria, 89
  - Test Readiness Review, 33
  - test-related reviews, 33
  - test-related risk management, 15
  - test-related risks, 57-58
  - test repository, 138
  - Test Resources Not Delivered for Maintenance (TTS-REG-5), 24, 35, 100, 126, 236-237
    - characteristic symptoms, 236
    - description, 236
    - potential applicability, 236
    - potential causes, 236-237
    - potential negative consequences, 236
    - recommendations, 237
    - related pitfalls, 237
  - test results
    - evaluating, xix
    - false-negative, 70, 90, 118, 119, 122, 127, 145, 153, 154-156, 165
    - false-positive, 70, 90, 118, 119, 122, 127, 145, 153, 154-156, 165
    - ignoring inconvenient, 15, 61-63
    - inconclusive or incorrect, 18, 118-122
  - test-resumption criteria, 32
  - tests, 249
    - abandoning broken, 157
    - architecture, design, and implementation, 165-166
    - automating, 166
    - configuration management (CM), 18
    - defects, 9
    - documenting results, 107
    - executing, 107
    - goals and objectives, 31
    - inadequate or excessive, 162-164
    - inconclusive or incorrect results, 119
    - obsolete, 139
    - passed, 48
    - planned code coverage, 59
    - prioritization, 16
    - purpose, 105
    - quality, 47
    - repeating same kind of test cases, 104-106
    - requirements, 92, 148, 162-164
    - resources, 14
    - scope, 105
    - size and complexity, 59
    - test-completion criteria, 105
    - types, 31-32
    - unofficial changes, 148
    - work product creation, 3
  - test schedule, 13, 39-41
  - test scripts, 106-108, 251
  - test strategy, 27, 251
  - Test Strategy Documents (TSDs), xix, 28, 251
  - test stub, 251
  - test-suspension criteria, 32
  - test team
    - budget and schedule, 66
    - defects, 60
    - financial independence, 67
    - geographical distribution of, 29
    - inadequate testing expertise, 15
    - lack of independence, 15, 66-67
    - productivity, 60
    - schedule, 41
    - system-level testing, 85
    - technical independence, 67
    - testing responsibilities unclear, 15
  - test team leader, 66
  - test-team-level risks, 57
  - test-team-wide test-case-selection criteria, 103
  - test tools, 92, 251
    - communicating with software under test (SUT), 118
    - inadequacy with, 70
    - insufficient, 18, 114-117
    - platform-specific, 111
    - proprietary, 108
    - responsibility for, 68
    - as system-external actors, 118
    - verification, validation, accreditation, or certification, 90
    - without adequate support of, 106-108
  - test tools and environments pitfalls, 106-130
    - Developers Ignore Testability (GEN-TTE-10), 18, 129-130
    - Difficulty Accessing Target Platform (GEN-TTE-4), 17
    - Inadequate Test Configuration Management (GEN-TTE-9), 18, 126-128
    - Inadequate Test Environment Quality (GEN-TTE-7), 18, 122-124
    - Inadequate Test Environments (GEN-TTE-5), 18, 114-117
    - Over-Reliance on Manual Testing (GEN-TTE-1), 17, 106-108
    - Over-Reliance on Testing Tools (GEN-TTE-2), 17, 108-110
    - Poor Fidelity of Test Environments (GEN-TTE-6), 18, 118-122
    - Target Platform Difficult to Access (GEN-TTE-4), 112-114
    - Test Assets Not Delivered (GEN-TTE-8), 18, 124-126
    - Too Many Target Platforms (GEN-TTE-3), 17, 110-112
  - Test-Type Confusion (GEN-PRO-15), 17, 104-106
    - characteristic symptoms, 104
    - description, 104
    - potential applicability, 104
    - potential causes, 105
    - potential negative consequences, 105
    - recommendations, 105-106
    - related pitfalls, 106
  - test-type-specific pitfalls, 11, 20-24, 164-239
    - integration testing pitfalls, 20-21, 169-177
    - regression testing pitfalls, 23-24, 225-239
    - specialty engineering testing pitfalls, 21-22, 177-206
    - system of systems (SoS) testing pitfalls, 22-23, 211-225
    - system testing pitfalls, 22, 206-211
    - unit testing pitfalls, 20, 164-168
  - test-type-specific test cases, 105
  - threat analysis, 200-203, 202
  - throughput, 188-190
  - TIMs. *See* technical interchange meetings
  - Too Immature for Testing (GEN-PRO-7), 69, 88-90, 130
    - characteristic symptoms, 88
    - description, 88
    - potential applicability, 88
    - potential causes, 89
    - potential negative consequences, 88-89
    - recommendations, 89-90
    - related pitfalls, 90
  - Too Many Target Platforms (GEN-TTE-3)
    - characteristic symptoms, 110
    - description, 110
    - potential applicability, 110
    - potential causes, 111
    - potential negative consequences, 110-111
    - recommendations, 111-112
    - related pitfalls, 112
  - training
    - agile development, 95
    - ambiguous requirements, 145
    - automated testing, 71
    - evolutionary development cycle, 95
    - incremental, iterative testing, 43
    - pitfalls, 27
    - requirements, 203
    - testing, 46
    - testing metrics, 60
    - test planning presentations, xix
    - verification methods, 49
  - trapdoors, 207
  - trigger events, 153, 251
  - Triple-V model, 4, 6, 43
  - trouble reports. *See* defect reports
  - TSDs. *See* Test Strategy Documents
- ## U
- UBIT. *See* User-initiated BIT
  - UCM. *See* Use Case Map
  - unavailable components, 173-175
  - Unavailable Components (TTS-INT-3), 20, 100, 122, 174-175
    - characteristic symptoms, 174

- Unavailable Components (*cont.*)
    - description, 173
    - potential applicability, 173
    - potential causes, 174–175
    - potential negative consequences, 174
    - recommendations, 175
    - related pitfalls, 175
  - unclassified network, 118
  - Unclear SoS Testing Responsibilities (TTS-SoS-2), 23, 69, 211, 213–215
    - characteristic symptoms, 213
    - description, 213
    - potential applicability, 213
    - potential causes, 214
    - potential negative consequences, 214
    - recommendations, 214–215
    - related pitfalls, 215
  - Unclear Testing Responsibilities (GEN-STF-2), 35, 54, 68–69, 90, 92, 94, 126, 130, 138, 166, 168, 215
    - characteristic symptoms, 68
    - description, 68
    - potential applicability, 68
    - potential causes, 68
    - recommendations, 68–69
    - related pitfalls, 68–69
  - undocumented requirements, 150–152
  - Unintentional Regression Testing, 104
  - unit defects, 59
  - unit-level designs, 165
  - unit-level test cases, 167
  - unit-level testing, 86, 159
  - unit-level tests, 104
  - units, 74, 129
  - unit testing, 60, 86, 251
    - automating, 168
    - developers, 74–75
    - before integration, 80
    - interface defects, 59
    - software developers, 167–168
    - unit defects, 59
  - unit testing pitfalls, 11, 20
    - Conflict of Interest (TTS-UNT-2), 20, 167–168
    - Testing Does Not Drive Design and Implementation (TTS-UNT-1), 20, 165–166
  - Unrealistic Testing Expectations (GEN-SIC-2), 47–49, 58, 228, 231, 234
    - characteristic symptoms, 47
    - description, 47
    - potential applicability, 47
    - potential causes, 48
    - potential negative consequences, 48
    - recommendations, 48–49
    - related pitfalls, 49
  - U.S. Department of Defense Global Information Grid (GIG), 121
  - U.S. Public Health Service, xv
  - usability defects, 62
  - usability requirements, 203
  - usability testing, 22, 203–206
  - use case modeling, 152, 210
  - use case paths, 251
    - abnormal, 193, 196
    - normal, 45
  - use cases, 151, 251
  - user acceptance testing, 104
  - User-initiated BIT (UBIT), 171, 172
  - user interface, 203–206
  - user representatives, 141
- V**
- validation, 251–252
  - verification, 252
  - verification methods, 243
    - non-testing forms ignored, 48
    - not properly specified, 20
    - training, 49
  - Verification Methods Not Properly Specified (GEN-REQ-8)
    - characteristic symptoms, 161
    - description, 161
    - potential applicability, 161
    - potential causes, 161
    - potential negative consequences, 161
    - recommendations, 162
    - related pitfalls, 162
  - verification milestones, 61
  - V Models
    - agile development, 4
    - analysis activities, 2
    - Double-V model, 3–4, 5
    - evolutionary development cycle, 4
    - sequential waterfall development cycle, 4
    - synthesis activities, 2
    - testing, 2–4
    - Triple-V model, 4, 6
    - work products, 2–3
  - vulnerability, 252
  - vulnerability analysis, 199, 202
  - vulnerability scanning, 201
  - Vulnerability Testing, 197, 200, 252
- W**
- waterfall development cycle, 4
    - testing as phase late in, 17, 94–96
  - WBS. *See* Work Breakdown Structure
  - white-box testing, 31, 38, 86, 109, 252
    - insufficient architecture or design documentation, 18, 131–133
    - repeating, 104
    - safeguards, 199
    - security features or subsystems, 202
    - unit and integration testing, 16
    - very little, 85
  - wikis, xix
  - Work Breakdown Structure (WBS), 76, 252
    - automated regression testing, 227–228
    - incorporating testing, 77
    - operational testing, 101
    - regression testing, 230
    - requirements, architecture, and design, 140
    - resources, 54
    - test assets, 91, 93
    - test environments and facilities, 117
  - work environment, 55
  - work products
    - configuration control, 126
    - execution and testing, 2–3
    - little testing of, 58
    - quality control evaluations of testing, 91–92
    - testing, 3, 32
  - V Models, 2–3
  - Wrong Testing Mindset (GEN-SIC-1), 14, 44–47, 56, 133, 168, 172
    - characteristic symptoms, 44
    - description, 44
    - potential applicability, 44
    - potential causes, 45
    - potential negative consequences, 45
    - recommendations, 45–47
    - related pitfalls, 47