# Common System and Software Testing Pitfalls

**Exploring Software Testing: Strategies and innovation**
**18 September 2014**

**Donald Firesmith**, Principle Engineer
Software Solutions Division
Software Engineering Institute (SEI)
Carnegie Mellon University
Pittsburgh, PA 15213

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.



This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

# Topics

Testing:

- Definition
- Software vs. System Testing
- Goals

Testing Pitfalls:

- Taxonomy / Ontology
- Testing Challenges
- Addressing these Challenges
- Goals and Potential Uses
- Example Pitfall

Taxonomy of Common Testing Pitfalls (lists of pitfalls by category):

- General Pitfalls
- Test-Type-Specific Pitfalls

Remaining Limitations and Questions

Future Work

# Testing Definition

Testing

> The execution of a system or application, subsystem, or model under specific preconditions (for example, pretest mode, states, stored data, or external conditions) with specific inputs so that its actual behavior (outputs and postconditions) can be compared with its expected or required behavior

Notes:

- Not just software
- Requires execution (do not confuse testing with QE or with other means of verification such as analysis, certification, demonstration, inspection, …)
- Requires ***controllability*** to set up preconditions and provide inputs
- Requires ***observability*** to determine outputs and postconditions

# *Software* vs. *System* Testing

These testing pitfalls occur when testing:

- Software applications
- Systems (hardware, software, data, personnel, facilities, equipment, etc.)
- Executable models (requirements, architecture, design)

Most pitfalls apply to ***both*** software and system testing.

The vast majority of ***software*** testers ***must*** address ***system*** testing issues:

- Software executes on hardware, and how well it executes depends on:
  - That hardware
  - Other software running on the same hardware
- Software communicates over:
  - "External" networks (Internet, NIPRNet, SIPRNet, WAN, LAN, MAN, etc.)
  - Data-center-internal networks connecting servers and data libraries (e.g., SAN)
  - Busses within systems (embedded software)
- Software must meet quality requirements (thresholds of relevant quality characteristics and attributes) that are actually system-level, not software-level.

# Goals of Testing

In decreasing order of importance:

- **Uncover Defects** in the system/software under test (SUT) by causing it to behave incorrectly (e.g., to fail or enter a faulty state) so that these underlying defects can be identified and fixed and the SUT can thereby be improved.
- **Provide Evidence** that can be used to determine the SUT's:
  - Quality
  - Fitness for purpose
  - Readiness for shipping, deployment, or being placed into operation
- **Prevent Defects** by:
  - Testing executable requirements and architecture models so that defects in the models are fixed before they can result in defects in the system/software.
  - Developing test cases and then using these test cases to drive development (design and implementation) – Test Driven Development (TDD)

Passing [initial] tests is not necessarily a good thing and may provide:

- Evidence that product does not have certain types of defects
- Evidence that the tests is ineffectual at uncovering defects

# Testing Pitfalls Definitions

Testing Pitfall

- Any decision, mindset, action, or failure to act that unnecessarily and, potentially unexpectedly, causes testing to be:
  - Less effective at uncovering defects
  - Less efficient in terms of effort expended
  - More frustrating to perform
- A way to screw up testing (i.e., a human mistake)

***Common*** Testing Pitfall

- Observed numerous times on different projects
- Having sufficient frequency (and consequences ) to be a significant risk

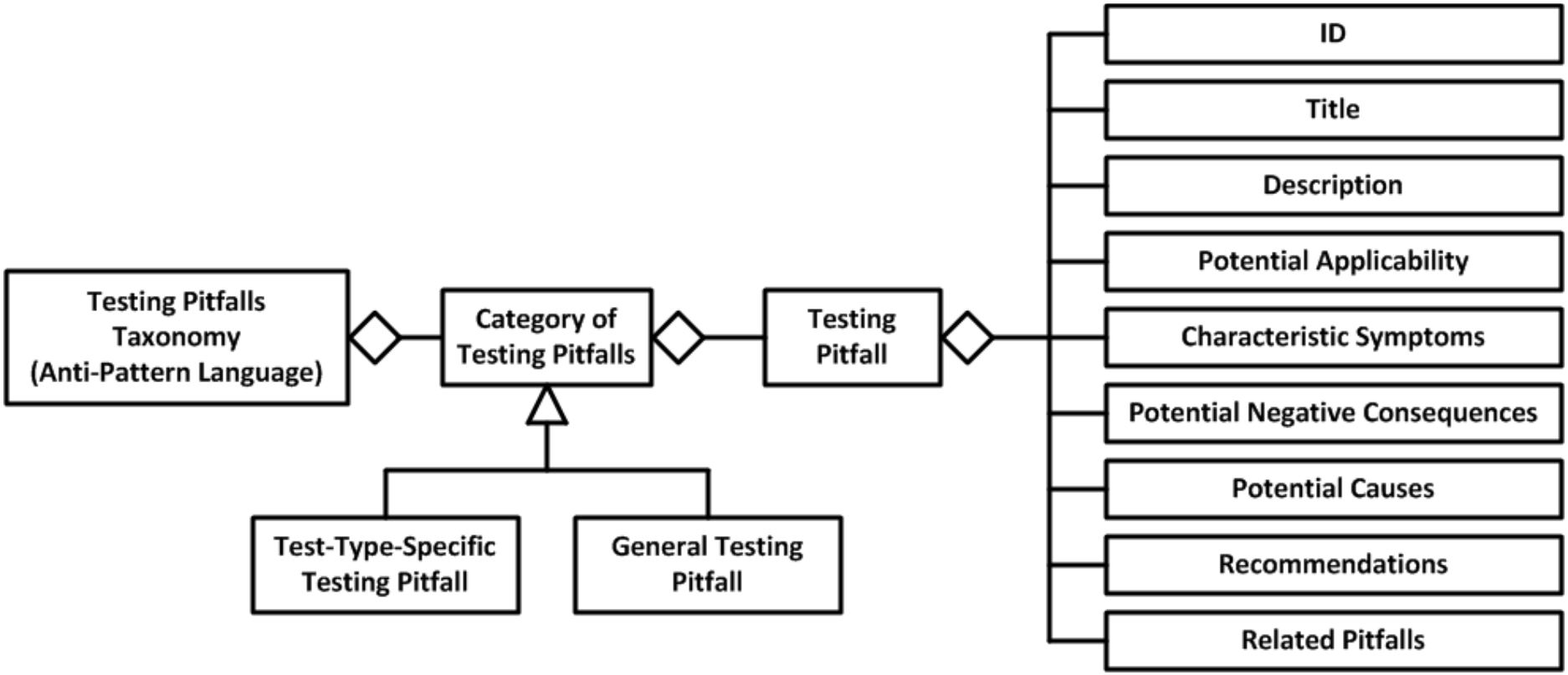
Testing Pitfall ***Taxonomy***

A hierarchical classification of testing pitfalls into categories

Testing Pitfall ***Ontology***

A hierarchy of concepts concerning testing pitfalls, using a shared vocabulary to denote the types, properties and interrelationships of these concepts

# Testing Pitfall Taxonomy and Ontology



# Testing Challenges

A great many different ways exist to screw up testing.

Multiple testing pitfalls are observed on just about every project.

Different programs often exhibit different testing pitfalls.

In spite of many excellent how-to testing books, we see projects falling into these same testing pitfalls over and over again.

# Using Testing Pitfalls to Address these Challenges

Testing Pitfalls Taxonomy and Ontology

Anti-Pattern Language of how-not-to do testing

*Common System and Software Testing Pitfalls* (Addison-Wesley, 2014) (Note: 35% conference discount):

- ***92*** pitfalls classified into ***14*** categories
- Technically reviewed by 47 testing international SMEs

Current Taxonomy with new pitfalls and pitfall categories:

- ***128*** pitfalls classified into ***18*** categories
- http://sites.google.com/a/firesmith.net/donald-firesmith/home/common-testing-pitfalls [Work in progress - new content is draft and may be incomplete]

Potential 2nd Edition of Pitfalls Book or Supplement to 1st Edition

Potential Testing Pitfalls Wiki

# Goals and Potential Uses of Testing Pitfalls

Goals:

- To become the de facto industry-standard taxonomy of testing pitfalls
- To reduce the incidence of testing pitfalls and thereby improve testing effectiveness and efficiency
- To improve the quality of systems and software

Potential Uses:

- Training materials for testers and testing stakeholders
- Standard terminology regarding commonly occurring testing pitfalls
- Checklists for use when:
  - Producing test strategies/plans and related documentations
  - Evaluating contractor proposals
  - Evaluating test strategies/plans and related documentation (quality control)
  - Evaluating as-performed test process (quality assurance)
  - Identifying test-related risks and their mitigation approaches
- Categorization of pitfalls for test metrics collection, analysis, and reporting

# Example – Wrong Testing Mindset (GEN-SIC-1)

**Description:**
Some of the testers and other testing stakeholders have an incorrect testing mindset.

**Potential Applicability:**

- This pitfall is always potentially applicable.

# Example – Wrong Testing Mindset (GEN-SIC-1)

**Characteristic Symptoms:**

- Some testers and other testing stakeholders believe that ***the purpose of testing*** is to ***demonstrate*** that the system works properly rather than to determine where and how it fails.[13]
- Testers believe that it is their job (***responsibility***) to verify or "***prove***" that the system works, rather than identify where and when it doesn't work.[14]
- Some testers and other testing stakeholders begin testing assuming that the system or software works, so testing is only performed to show this.
- Managers and acquirers believe that testing is a cost center (that is, an expense rather than an investment) because they do not see the value of the products produced (for example, test documentation, test cases, and test environments) or the costs avoided due to testing.
- Testers believe that the purpose of testing is to find out how the system actually behaves, without considering how it should or must behave. In this mindset, testing is unrelated to requirements, and defects are subjective and all in the "eye of the beholder."
- Only normal (primary and alternative) behavior is tested.
- There is little or no testing of:
  - Exceptional (error-, fault-, or failure-tolerant) behavior
  - Input data:
    - There is no testing to identify incorrect handling of invalid input values.
    - Test inputs include only middle-of-the-road values rather than boundary values and corner cases.

# Example – Wrong Testing Mindset (GEN-SIC-1)

**Potential Consequences:**


- There is a high probability that:
  - the delivered system or software will contain a significant number of residual defects, especially related to abnormal behavior (e.g., exceptional use case paths)
  - these defects will unacceptably reduce its reliability and robustness (e.g., error, fault, and failure tolerance)
- Customer representatives, managers, and developers have a false sense of security that the system functions properly.

# Example – Wrong Testing Mindset (GEN-SIC-1)

**Potential Causes:**

- Testers were taught or explicitly told that their job is to verify or "prove" that the system/software works. [15]
- Developers are testing their own software [16] so that there is a "conflict of interest" (i.e., build software that works and show that their software does not work). This is especially a problem with small, cross-functional development organizations/teams that "cannot afford" to have separate testers (i.e., professional testers who specialize in testing).
- There was insufficient schedule allocated for testing so that there is only sufficient time to test the normal behavior (e.g., use case paths).
- The organizational culture is very success oriented so that looking "too hard" for problems is (implicitly) discouraged.
- Management gave the testers the strong impression that they do not want to hear any "bad" news (i.e., that there are any significant defects being found in the system).

# Example – Wrong Testing Mindset (GEN-SIC-1)

## Recommendations:

- **Prepare:**
  - Explicitly state in the project test plan that the primary goals of testing is to:
    - break the system by causing system faults and failures in order to identify residual defects so that they can be fixed
    - thereby determine the
    - quality of the system
    - system's fitness for purpose
    - system's readiness for shipping, deployment, and/or operation
- **Enable:**
  - Provide test training that emphasizes uncovering defects by causing faults or failures.
  - Provide sufficient time in the schedule for testing beyond the basic success paths.
  - Hire new testers who exhibit a strong "destructive" mindset to testing.
  - When relevant, identify this pitfall as a risk in the project risk repository.
- **Perform:**
  - In addition to test cases that verify all normal behavior, emphasize looking for defects where they are most likely to hide (e.g., boundary values, corner cases, and input type/range verification). [17]
  - Incentivize testers based more on the number of significant defects they uncover than merely on the number requirements "verified" or test cases ran. [18]
  - Foster a healthy competition between developers (who seek to avoid inserting defects) and testers (who seek to find those defects).

# Example – Wrong Testing Mindset (GEN-SIC-1)

- **Verify:**
  - Determine whether the testers exhibit a testing mindset.
  - Determine whether the goals of testing are documented in the test planning documentation.
  - Determine whether any test training covers the proper testing mindset.
  - Determine whether adequate time has been scheduled to enable testing beyond the basic success paths.
  - Determine (for example, via conversation or questioning) whether testing goes beyond "demonstrate that the system works" (sunny-day path testing) to also include "demonstrate that the system does not work" (rainy-day path testing)
  - Determine whether the testers exhibit the correct testing mindset.

**Related Pitfalls:**

- Inappropriate External Pressures (GEN-MGMT-2)
- Inadequate Communication Concerning Testing (GEN-COM-5)
- Unit Testing Considered Unimportant (TTS-UNT-3)

# Categories of Testing Pitfalls – General

1.1 Test Planning and Scheduling Pitfalls

1.2 Stakeholder Involvement and Commitment Pitfalls

1.3 Management-Related Testing Pitfalls

1.4 Staffing Pitfalls

1.5 Test Process Pitfalls

1.6 Test Tools and Environments Pitfalls

1.7 Automated Testing Pitfalls [new pitfall category since book]

1.8 Test Communication Pitfalls

1.9 Requirements-Related Testing Pitfalls

# 1 General Pitfalls –
## 1.1 Test Planning and Scheduling Pitfalls

No Separate Test Planning Documentation (GEN-TPS-1)

Incomplete Test Planning (GEN-TPS-2)

Test Plans Ignored (GEN-TPS-3)

Test-Case Documents as Test Plans (GEN-TPS-4)

Inadequate Test Schedule (GEN-TPS-5)

Testing at the End (GEN-TPS-6)

Independent Test Schedule (GEN-TPS-7) [new pitfall]

# 1. General Pitfalls –
## 1.2 Stakeholder Involvement and Commitment Pitfalls

Wrong Testing Mindset (GEN-SIC-1)

Unrealistic Testing Expectations (GEN-SIC-2)

Lack of Stakeholder Commitment to Testing (GEN-SIC-3)

# General Pitfalls –
## 1.3 Management-related Testing Pitfalls

Inadequate Test Resources (GEN-MGMT-1)

Inappropriate External Pressures (GEN-MGMT-2)

Inadequate Test-Related Risk Management (GEN-MGMT-3)

Inadequate Test Metrics (GEN-MGMT-4)

Inconvenient Test Results Ignored (GEN-MGMT-5)

Test Lessons Learned Ignored (GEN-MGMT-6)

# General Pitfalls –
## 1.4 Staffing Pitfalls

Lack of Independence (GEN-STF-1)

Unclear Testing Responsibilities (GEN-STF-2)

Developers Responsible for All Testing (GEN-STF-3)

Testers Responsible for All Testing (GEN-STF-4)

Users Responsible for Testing (GEN-STF-5) [new pitfall]

Inadequate Testing Expertise (GEN-STF-6)

Inadequate Domain Expertise (GEN-STF-7) [new pitfall]

Adversarial Relationship (GEN-STF-8) [new pitfall]

# General Pitfalls –
## 1.5 Test Process Pitfalls 1

No Testing Process (GEN-PRO-1) [new pitfall]

Essentially No Testing (GEN-PRO-2) [new pitfall]

Incomplete Testing (GEN-PRO-3)

Testing Process Ignored (GEN-PRO-4) [new pitfall]

One-Size-Fits-All Testing (GEN-PRO-5)

Sunny Day Testing Only (GEN-PRO-6) [new pitfall]

Testing and Engineering Processes Not Integrated (GEN-PRO-7)

Inadequate Test Prioritization (GEN-PRO-8)

Test-Type Confusion (GEN-PRO-9)

Functionality Testing Overemphasized (GEN-PRO-10)

# General Pitfalls –
## 1.5 Test Process Pitfalls 2

Black-Box System Testing Overemphasized (GEN-PRO-11)

Black-Box System Testing Underemphasized (GEN-PRO-12)

Test Preconditions Ignored (GEN-PRO-13) [new pitfall]

Too Immature for Testing (GEN-PRO-14)

Inadequate Test Data (GEN-PRO-15)

Inadequate Evaluations of Test Assets (GEN-PRO-16)

Inadequate Maintenance of Test Assets (GEN-PRO-17)

Testing as a Phase (GEN-PRO-18)

Testers Not Involved Early (GEN-PRO-19)

Developmental Testing During Production (GEN-PRO-20) [new pitfall]

# General Pitfalls –
## 1.5 Test Process Pitfalls 3

No Operational Testing (GEN-PRO-21)

Test Oracles Ignore Nondeterministic Behavior (GEN-PRO-22) [new pitfall]

Testing in Quality (GEN-PRO-23) [new pitfall]

# General Pitfalls –
## 1.6 Test Tools and Environments Pitfalls

Over-Reliance on Manual Testing (GEN-TTE-1)

Over-Reliance on Testing Tools (GEN-TTE-2)

Too Many Target Platforms (GEN-TTE-3)

Target Platform Difficult to Access (GEN-TTE-4)

Inadequate Test Environments (GEN-TTE-5)

Poor Fidelity of Test Environments (GEN-TTE-6)

Inadequate Test Environment Quality (GEN-TTE-7)

Test Environments Inadequately Tested (GEN-TTE-8) [new pitfall]

Inadequate Test Configuration Management (GEN-TTE-9)

Developers Ignore Testability (GEN-TTE-10)

Test Assets Not Delivered (GEN-TTE-11) [combined 2 existing pitfalls]

# General Pitfalls –
## 1.7 Automated Testing Pitfalls [new pitfall category]

Over-Reliance on Manual Testing (GEN-AUTO-1)
[moved from Test Tools and Environments Category]

Automated Testing Replaces Manual Testing (GEN-AUTO-2) [new pitfall]

Excessive Number of Automated Tests (GEN-AUTO-3) [new pitfall]

Inappropriate Distribution of Automated Tests (GEN-AUTO-4) [new pitfall]

Inadequate Automated Test Quality (GEN-AUTO-5) [new pitfall]

Automated Tests Excessively Complex (GEN-AUTO-6) [new pitfall]

Automated Tests Not Maintained (GEN-AUTO-7) [new pitfall]

Insufficient Resources Invested (GEN-AUTO-8) [new pitfall]

Automation Tools Not Appropriate (GEN-AUTO-9) [new pitfall]

Stakeholders Ignored (GEN-AUTO-10) [new pitfall]

# General Pitfalls –
## 1.8 Test Communication Pitfalls

Inadequate Source Documentation (GEN-COM-1) [Expanded & Renamed]

Inadequate Defect Reports (GEN-COM-2)

Inadequate Test Documentation (GEN-COM-3)

Source Documents Not Maintained (GEN-COM-4)

Inadequate Communication Concerning Testing (GEN-COM-5)

Inconsistent Testing Terminology (GEN-COM-6) [new pitfall]

# General Pitfalls –
## 1.9 Requirements Testing Pitfalls

Tests as Requirements (GEN-REQ-1) [new pitfall]

Ambiguous Requirements (GEN-REQ-2)

Obsolete Requirements (GEN-REQ-3)

Missing Requirements (GEN-REQ-4)

Incomplete Requirements (GEN-REQ-5)

Incorrect Requirements (GEN-REQ-6)

Requirements Churn (GEN-REQ-7)

Improperly Derived Requirements (GEN-REQ-8)

Verification Methods Not Properly Specified (GEN-REQ-9)

Lack of Requirements Trace (GEN-REQ-10)

Titanic Effect of Deferred Requirements (GEN-REQ-11) [new pitfall]

# Test-Type-Specific Pitfalls

2.1 Executable Model Testing Pitfalls [new pitfall category]

2.2 Unit Testing Pitfalls

2.3 Integration Testing Pitfalls

2.4 Specialty Engineering Testing Pitfalls

2.5 System Testing Pitfalls

2.6 User Testing Pitfalls [new pitfall category]

2.7 Acceptance Testing Pitfalls [new pitfall category]

2.8 System of Systems (SoS) Testing Pitfalls

2.9 Regression Testing Pitfalls

# Test Type Specific Pitfalls –
## 2.1 Executable Model Testing Pitfalls

Inadequate Executable Models (TTS-MOD-1) [new pitfall]

Executable Models Not Tested (TTS-MOD-2) [new pitfall]

# Test Type Specific Pitfalls –
## 2.2 Unit Testing Pitfalls

Testing Does Not Drive Design and Implementation (TTS-UNT-1)

Conflict of Interest (TTS-UNT-2)

Brittle Test Cases (TTS-UNT-3) [new pitfall]

# Test Type Specific Pitfalls –
## 2.3 Integration Testing Pitfalls

Integration Decreases Testability Ignored (TTS-INT-1)

Inadequate Self-Testing (TTP-INT-2)

Unavailable Components (TTS-INT-3)

System Testing as Integration Testing (TTS-INT-4)

# Test Type Specific Pitfalls –
## 2.4 Specialty Engineering Testing Pitfalls

Inadequate Capacity Testing (TTS-SPC-1)

Inadequate Concurrency Testing (TTS-SPC-2)

Inadequate Internationalization Testing (TTS-SPC-3)

Inadequate Interface Standards Compliance Testing (TTS-SPC-4) [new pitfall]

Inadequate Interoperability Testing (TTS-SPC-5)

Inadequate Performance Testing (TTS-SPC-6)

Inadequate Reliability Testing (TTS-SPC-7)

Inadequate Robustness Testing (TTS-SPC-8)

Inadequate Safety Testing (TTS-SPC-9)

Inadequate Security Testing (TTS-SPC-10)

Inadequate Usability Testing (TTS-SPC-11)

# Test Type Specific Pitfalls –
## 2.5 System Testing Pitfalls

Test Hooks Remain (TTS-SYS-1)

Lack of Testing Hooks (TTS-SYS-2)

Inadequate End-to-End Testing (TTS-SYS-3)

# Test Type Specific Pitfalls –
## 2.6 User Testing

Inadequate User Involvement (TTS-UT-1 ) [new pitfall]

Unprepared User Representatives (TTS-UT-2 ) [new pitfall]

User Testing Merely Repeats System Testing (TTS-UT-3) [new pitfall]

User Testing is Mistaken for Acceptance Testing (TTS-UT-4) [new pitfall]

Assume Knowledgeable and Careful User (TTS-UT-5) [new pitfall]

# Test Type Specific Pitfalls –
## 2.7 Acceptance Testing Pitfalls

No Clear System Acceptance Criteria (TTS-AT-1) [new pitfall]

# Test Type Specific Pitfalls –
## 2.8 System of System Testing Pitfalls

Inadequate SoS Test Planning (TTS-SoS-1)

Unclear SoS Testing Responsibilities (TTS-SoS-2)

Inadequate Resources for SoS Testing (TTS-SoS-3)

SoS Testing not Properly Scheduled (TTS-SoS-4)

Inadequate SoS Requirements (TTS-SoS-5)

Inadequate Support from Individual System Projects (TTS-SoS-6)

Inadequate Defect Tracking Across Projects (TTS-SoS-7)

Finger-Pointing (TTS-SoS-8)

# General Pitfalls –
## 2.9 Regression Testing Pitfalls

Inadequate Regression Test Automation (GEN-REG-1)

Regression Testing Not Performed (GEN-REG-2)

Inadequate Scope of Regression Testing (GEN-REG-3)

Only Low-Level Regression Tests (GEN-REG-4)

Test Resources Not Delivered for Maintenance (GEN-REG-5)

Only Functional Regression Testing (GEN-REG-6)

Inadequate Retesting of Reused Software (TTS-REG-7) [new pitfall]

# Remaining Limitation and Questions

Current Taxonomy is Experience Based:

- Based on experience testing and assessing testing programs (author, SEI ITAs, technical reviewers)
- Not the result of documentation study or formal academic research

Remaining Questions:

- Which pitfalls occur most often? With what frequency?
- Which pitfalls cause the most harm?
- Which pitfalls have the highest risk (expected harm = harm frequency x harm)?
- What factors (e.g., system size and complexity, application domain, process) influence frequency, harm, and risk?

# Future Work

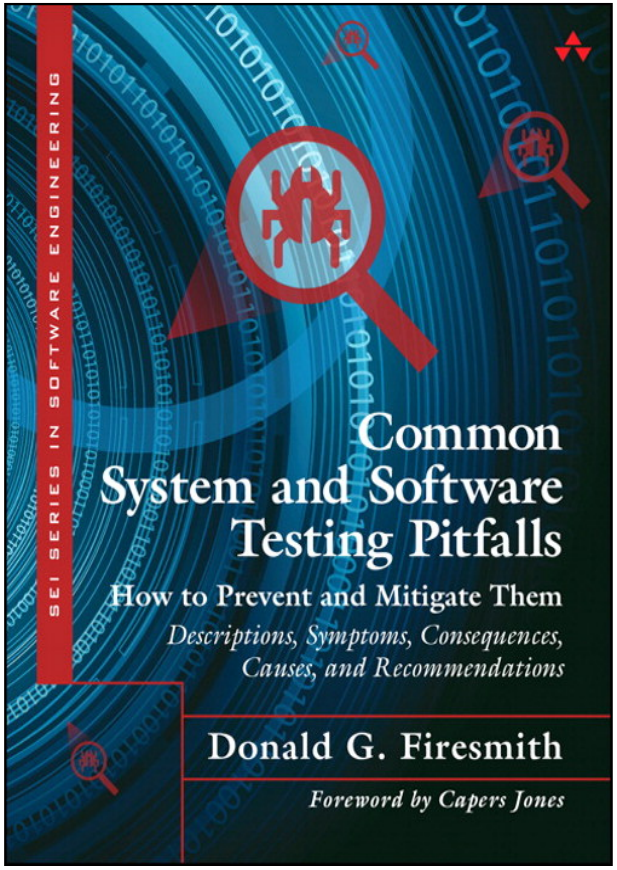
Extensive Technical Review:

- New testing pitfall categories
- New testing pitfalls
- Modified testing pitfalls

Proper Industry Survey:

- How likely are the different testing pitfalls? What are the 10 most common?
- What pitfalls have the worst consequences? What are the 10 worst pitfalls?
- What pitfalls have the highest risk? What are the 10 highest risk pitfalls?
- Do the answers to these questions vary by:
  - System (size, complexity, criticality, application domain, software only vs. HW/SW/people/documentation/facilities/procedures…, system vs. SoS vs. PL)?
  - Project (type, formality, lifecycle scope, schedule, funding, commercial vs. government/military,…)
  - Organization (number, size, type, governance, management/engineering culture,…)

# Save 35%




## informit.com

**Discount code:**

**FIRESMITH550**

- **Available as book & eBook**
- **FREE shipping in the U.S.**
- **Safari Books Online**

Addison Wesley

# Contact Information Slide Format

**Donald Firesmith**
Principle Engineer
Software Support Division (SSD)
Telephone: +1 412-268-6874
Email: dgf@sei.cmu.edu

**U.S. Mail**
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

**Web**
www.sei.cmu.edu
www.sei.cmu.edu/contact.cfm

**Customer Relations**
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257