



Common Testing Problems: Pitfalls to Prevent and Mitigate

11 October 2012

Donald Firesmith
Software Engineering Institute (SEI)
Carnegie Mellon University
Pittsburgh, PA 15213



Clarification and Caveat

Common – The following testing problems are common in the sense that they occur with such frequency that several are typically observed on most if not all programs.

Not Prioritized – Given that all of these problems should be avoided or their negative consequences should be mitigated if they do occur, no attempt has been made to prioritize them in terms of either frequency or severity.

Experienced Based – The list is not based on any rigorous study of specific programs. Rather, it is a compendium of observations from system/software engineers and testers based on both performing and assessing testing on real programs.



Topics

Goal and Objectives

General Testing Problems

Problems by Source:

- Test Planning Problems
- Requirements and Testing Problems

Problems by Type of Testing:

- Unit Testing Problems
- Integration Testing Problems
- Specialty-Engineering Testing Problems
- System Testing Problems
- System of System (SoS) Testing Problems
- Regression and Maintenance Testing Problems

Recommendations



Goal and Objectives

Overall Goal:

- To provide a **high-level summary** of commonly-occurring testing problems

Supporting Objectives:

- To **list** the different types of commonly-occurring testing problems in an organized manner
- To finish by making highly **general recommendations** as to how to avoid/mitigate these problems
- To provide the information needed to generate testing oversight and assessment **checklists**
 - A general purpose checklist with associated recommendations for preventing, detecting, and reacting to these problems has been created is available from the presenter.



General Testing Problems – 1

Problems	Symptoms
Wrong Test Mindset (“Happy Path Testing”)	Testing being used to prove that system/software works rather than to show where and how it fails. Only nominal (“sunny day”) behavior rather than off-nominal behavior is being tested. Test input only includes middle of the road values rather than boundary values and corner cases.
Defects Discovered Late	Large numbers of requirements, architecture, and design defects are being found that should have been discovered (during reviews) and fixed prior to current testing. Defects that should have been discovered during lower-level testing are slipping through until higher-level testing or even after delivery.
Inadequate Testing Expertise	Contractor testers or Government representatives have inadequate testing expertise. Little or no training in testing has taken place.
Inadequate Schedule	Testing is inadequately incomplete because there is insufficient time allocated in the schedule for all appropriate tests to be performed. For example, an agile (i.e., iterative, incremental, and concurrent) development/life cycle greatly increases the amount of regression testing needed.



General Testing Problems – 2

Problems	Symptoms
Inadequate Test Metrics	There are insufficient test metrics being produced, analyzed, and reported. The primary test metrics (e.g., number of tests passed, number of defects found) show neither the productivity of the testers nor their effectiveness at finding defects (e.g., defects found per test or per day). The number of latent <i>undiscovered</i> defects remaining is not estimated (e.g., using COQUALMO).
Inadequate Test-related Risk Management	There are <i>no</i> test-related risks identified in the project's official risk repository. The number of test-related risks is unrealistically low. The identified test-related risks have inappropriately low priorities.
Unrealistic Expectations	Non-testers (e.g., managers and customer representatives) falsely believe that (1) testing detects all (or the majority of) defects, (2) testing can be effectively exhaustive, and (3) testing <i>proves</i> that the system works. Testing is being relied upon for <i>all</i> verification.



General Testing Problems – 3

Problems	Symptoms
False Sense of Security	Non-testers expect that passing testing <i>proves</i> that there are no remaining defects. Non-testers do <i>not</i> realize that a passed test could result from a weak test rather than a lack of defects. Non-testers do <i>not</i> understand that a truly successful test is one that finds one or more defects.
Over-reliance on COTS Testing Tools	Testers place too much reliance on testing tools and the automation of test case creation. Testers are relying on the tool as their test oracle (to determine the correct test result). Testers let the tool drive the test methodology rather than the other way around. Testers are using the tool to automate test case selection and completion (“coverage”) criteria.
Inadequate Documentation	Testing assets (e.g., test documents, environments, and test cases) are not sufficiently documented to be useful during maintenance and to be reusable (e.g., within product lines).



General Testing Problems – 4

Problems	Symptoms
Inadequate Maintenance	Testing assets (e.g., test software and documents such as test cases, test procedures, test drivers, and test stubs) are not being adequately maintained as defects are found and system changes are introduced (e.g., due to refactoring). The testing assets are no longer consistent with the current requirements, architecture, design, and implementation. Regression test assets are not updated when an agile development cycle with numerous increments is used.
Inadequate Prioritization	All types of testing are given the same priority. All test cases for the system or a subsystem are given the same priority. The most important tests of a given type are not being performed first. Difficult but important testing is postponed until late in the schedule.
Inadequate CM	Test plans, procedures, test cases, and other testing work products are <i>not</i> being placed under configuration control.



General Testing Problems – 5

Problems	Symptoms
Lack of Review	No [peer-level] review of the test assets (e.g., test inputs, preconditions (pre-test state), and test oracle including expected test outputs and postconditions) is being performed prior to actual testing.
Poor Communication	There is inadequate testing-related communication between: <ul style="list-style-type: none">• Teams within large or geographically-distributed programs• Contractually separated teams (prime vs. subcontractor, system of systems)• Between testers and:<ul style="list-style-type: none">○ Other developers (requirements engineers, architects, designers, and implementers)○ Other testers○ Customers, user representatives, and subject matter experts (SMEs)



General Testing Problems – 6

Problems	Symptoms
External Influences	Managers or developers are dictating to the testers what constitutes a bug or a defect <i>worth reporting</i> . Managerial pressure exists to not find defects (e.g., until after delivery because the project so far behind schedule that there is no time to fix any defects found).



Test Planning Problems – 1

Problems	Symptoms
No Separate Test Plan	There is <i>no</i> separate Test and Evaluation Master Plan (TEMP) or Software Test Plan (STP). There are only incomplete high-level overviews of testing in System Engineering Master Plans (SEMPs) and Software Development Plans (SDPs).
Incomplete Test Planning	The test planning documents <i>lack</i> clear and specific: <ul style="list-style-type: none">• test objectives• testing methods and techniques (e.g., testing is ad hoc, and planning documents merely list the different types of testing rather than state how the testing will be performed)• test case selection criteria (e.g., single nominal test case vs. boundary value testing)• test completion criteria (e.g., does it include both nominal and off-nominal testing)



Test Planning Problems – 2

Problems	Symptoms
One-Size-Fits-All Test Planning	The test planning documents contain only generic boilerplate rather than appropriate system-specific information. Are mission-, safety-, and security-critical software are not required to be tested more completely and rigorously than other less-critical software?
Inadequate Resources Planned	The test planning documents and schedules <i>fail</i> to provide for adequate test resources such as: <ul style="list-style-type: none">• test time in schedule with inadequate schedule reserves• trained and experienced testers and reviewers• funding• test tools and environments (e.g., integration test beds)



Requirements-Related Testing Problems – 1

Problems	Symptoms
Ambiguous Requirements	Testers are misinterpreting requirements that are ambiguous due to the use of: <ul style="list-style-type: none">• inherently ambiguous words• undefined technical terms and acronyms• quantities without associated units of measure• synonyms
Missing Requirements	Testing is incomplete (e.g., missing test cases) because of missing requirements: <ul style="list-style-type: none">• Use case analysis primarily addressed normal (sunny day) paths as opposed to fault tolerant and failure (rainy day) paths.• Requirements for off-nominal behavior (e.g., fault and failure detection and reaction) are missing.• Quality requirements (e.g., availability, interoperability, maintainability, performance, portability, reliability, robustness, safety, security, and usability) are missing.• Data requirements are missing.



Requirements-Related Testing Problems – 2

Problems	Symptoms
Incomplete Requirements	Testing is incomplete or incorrect due to incomplete requirements that lack: <ul style="list-style-type: none">• Preconditions and trigger events• Quantitative thresholds• Postconditions
Requirements Lack Good Characteristics	Testing is difficult or impossible because many requirements lack the characteristics of good requirements such as being complete, consistent, correct, feasible, mandatory, testable and unambiguous.
Unstable Requirements	Test cases (test inputs, preconditions, and expected test outputs) and automated regression tests are being obsoleted because the requirements are constantly changing.



Requirements-Related Testing Problems – 3

Problems	Symptoms
Poorly Derived Requirements	Testing is difficult because derived requirements merely restate their parent requirement and newly allocated requirements are not at the proper level of abstraction.



Unit Testing Problems

Problems	Symptoms
Unstable Design	Test cases must be constantly updated and test hooks are lost due to design changes (e.g., refactoring and new capabilities).
Inadequate Detailed Design	Unit testing is difficult to perform and repeat because there is insufficient design detail to drive the testing.
Poor Fidelity of Test Environment	Unit testing is experiencing many false positives because it is being performed using a: <ul style="list-style-type: none">• different compiler [version] than the delivered code• software test environment with poor hardware simulation



Integration Testing Problems – 1

Problems	Symptoms
Defect Localization	It is difficult to determine the location of the defect. Is the defect in the new or updated operational software under test, the operational hardware under test, in the COTS OS and middleware, in the software test bed (e.g., in software simulations of hardware), in the hardware test beds (e.g., in pre-production hardware), in the tests themselves (e.g., in the test inputs, preconditions, expected outputs, and expected postconditions), or in a configuration/version mismatch among them?
Insufficient Test Environments	There are an insufficient number of test environments. There is an excessive amount of competition between and among the integration testers and other testers for time on the test environment.



Integration Testing Problems – 2

Problems	Symptoms
Schedule Conflicts	Too many test teams are sharing the same test beds. It is difficult to optimally schedule the allocation of test teams to test beds. Too much time is wasted reconfiguring the test bed for the next team's use.
Unavailable Components	The operational software, simulation software, test hardware, and actual hardware components are not available for integration into the test environments prior to scheduled integration testing.
Poor Test Bed Quality	The test environments contain excessive numbers of defects making it difficult to schedule and perform testing.
Inadequate Self-Test	Failures are difficult to cause, reproduce, and localize because the operational software or subsystem does not contain sufficient test hooks, built-in-test (BIT), or prognostics and health management (PHM) software.



Specialty Engineering Testing – 1

Problems	Symptoms
Inadequate Capacity Testing	There is little or no testing to determine performance as capacity limits are approached, reached, and exceeded.
Inadequate Reliability Testing	There is little or no long duration testing under operational profiles to estimate the system's reliability.



Specialty Engineering Testing – 2

Problems	Symptoms
Inadequate Robustness Testing	<p>The testing is not based on robustness analysis such as off-nominal (i.e., fault, degraded mode, and failure) use case paths, Event Tree Analysis (ETA), Fault Tree Analysis (FTA), or Failure Modes Effects Criticality Analysis (FMECA).</p> <p>There is little or no testing <i>Robustness Testing</i>:</p> <ul style="list-style-type: none">• <i>Error Tolerance Testing</i>, the goal of which is to show that system does not detect or react properly to input errors (a subtype of which is <i>Fuzz Testing</i>)• <i>Fault Tolerance Testing</i>, the goal of which is to show that system does not detect or react properly to system faults (bad states)• <i>Failure Tolerance Testing</i>, the goal of which is to show that system does not detect or react properly to system failures• <i>Environmental Tolerance Testing</i>, the goal of which is to show that system does not detect or react properly to dangerous environmental conditions



Specialty Engineering Testing – 3

Problems	Symptoms
Inadequate Safety Testing	<p>There is little or no:</p> <ul style="list-style-type: none">• testing based on safety analysis (e.g., abuse/mishap cases, ETA, or FTA)• testing of safeguards (e.g., interlocks)• fail-safe behavior• safety-specific testing:<ul style="list-style-type: none">○ <i>Vulnerability Testing</i>, the goal of which is to expose a system vulnerability (i.e., defect or weakness)○ <i>Hazard Testing</i>, the goal of which is to make the system cause a hazard to come into existence○ <i>Mishap Testing</i>, the goal of which is to make the system cause an accident or near miss



Specialty Engineering Testing – 4

Problems	Symptoms
Inadequate Security Testing	<p>There is little or no:</p> <ul style="list-style-type: none">• testing based on security analysis (e.g., attack trees or abuse/misuse cases)• testing of security controls (e.g., access control, encryption/decryption, or intrusion detection)• fail-secure behavior• security-specific testing:<ul style="list-style-type: none">○ <i>Penetration Testing</i>, the goal of which is to penetrate the systems defenses○ <i>Fuzz Testing</i>, the goal of which is to cause the system to fail due to random input○ <i>Vulnerability Testing</i>, the goal of which is to expose a system vulnerability (i.e., defect or weakness)
Inadequate Usability Testing	<p>There is little or no explicit usability testing of human interfaces for user friendliness, learnability, etc.</p>



System Testing Problems

Problems	Symptoms
Testing Fault Tolerance is Difficult	It is difficult for tests of the integrated system to cause local faults (i.e., internal to a subsystem) in order to test for fault tolerance.
Testing Code Coverage is Difficult	It is difficult for tests of the integrated system to demonstrate code coverage, which is very important for safety and security.
Lack of Test Hooks	It is difficult to test locally implemented requirements because internal test hooks and testing software has been removed.



Systems of Systems Testing Problems – 1

Problems	Symptoms
Inadequate SoS Planning	Little or no planning has occurred for testing above the individual system level. There are no clear test completion/acceptance criteria at the system of systems level.
Poor or Missing SoS Requirements	Requirements-based testing is difficult because little or no requirements exist above the system level so that there are no official approved SoS requirements to verify.
Unclear Testing Responsibilities	No project is explicitly tasked with testing end-to-end SoS behavior.
Testing not Funded	No program is funded to perform end-to-end SoS testing.
Testing not Properly Scheduled	SoS testing is not in the individual systems' integrated master schedules, and there is no SoS master schedule. SoS testing must be fit into the uncoordinated schedules of the individual systems.



Systems of Systems Testing Problems – 2

Problems	Symptoms
Inadequate Test Support from Individual Systems	It is difficult to obtain the necessary test resources (e.g., people and test beds) from individual projects because they are already committed and there is no funding for such support.
Poor Defect Tracking Across Projects	There is little or no coordination of defect tracking and associated regression testing across multiple projects.
Finger-Pointing	There is a significant amount of finger pointing across project boundaries regarding where defects lie (i.e., in which systems and in which project's testing).



Regression/Maintenance Testing Problems – 1

Problems	Symptoms
Insufficient Automation	Testing is not sufficiently automated to decrease the testing effort, especially when an agile (iterative, incremental, and parallel) development cycle results in large numbers of increments that must be retested. Regression testing takes so much time and effort that it is rarely done.
Regression Tests Not Rerun	Regression testing is not being done because: <ul style="list-style-type: none">• There is insufficient time and staffing to perform it.• Managers or developers do not believe that it is necessary because of the minor scope of most changes.• There is insufficient automation of regression tests.
Inadequate Scope of Regression Testing	Only test the changed code because the “change can’t effect the rest of the system.”
Only Low-Level Regression Tests	Only unit tests and some integration tests are rerun. System and/or the SoS tests are not rerun



Regression/Maintenance Testing Problems – 2

Problems	Symptoms
Disagreement over Funding	There is disagreement as to whether the budget for testing during maintenance should come from development or sustainment funding.



Recommendations

Ensure adequate testing expertise and experience within the PO:

- Program Office personnel
- Systems Engineering and Technical Assistance (SETA) contractors
- SEI Acquisition Support Program (ASP) Subject Matter Experts (SMEs)

Use the associated Testing Problems Checklist:

- Problems, Symptoms, Recommendations

Require the prevention/mitigation of these problems in the RFPs.

Evaluate contractor and subcontractor proposed solutions:

- *Proposals*
- *Test [and Evaluation] Plans or Verification and Validation (V&V) Plans*
- Testing sections of *SEMPs* and *SDPs*

Do *not* accept inadequate contractor/subcontractor testing documents.

Evaluate implementations of test plans and processes.

Push back against testing shortcuts late in the program.



Conclusion

The bad news:

- There are an embarrassingly large number of common test problems.

The good news:

- These problems are foreseeable.
- These problems are relatively easy to prevent or fix.



Contact Information Slide Format

Donald Firesmith

Senior Member Technical Staff

Acquisition Support Program

Telephone: +1 412-268-6874

Email: dgf@sei.cmu.edu

Web

www.sei.cmu.edu

www.sei.cmu.edu/contact.cfm

U.S. Mail

Software Engineering Institute

Customer Relations

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

USA

Customer Relations

Email: info@sei.cmu.edu

Telephone: +1 412-268-5800

SEI Phone: +1 412-268-5800

SEI Fax: +1 412-268-6257



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 Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

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.

