Common Testing Problems: Pitfalls to Prevent and Mitigate

AIAA Case Conference
12 September 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
- Modern Lifecycles Testing 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 Testing Problems
- Maintenance Testing Problems

Recommendations

Goals 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**

Source – General Testing Problems – 1

Problem	Description
Wrong test mindset	Prove that system/software works rather than show how it fails => • Test only nominal ("sunny day") behavior rather than off-nominal behavior • Test middle of the road rather than boundary values and corner cases
Inadequate testing expertise	• Both contractors and Government • There is more to testing than listing the different types of testing.
Over-reliance on COTS testing tools	Inadequate support for: • Test case **selection** criteria • Test case **completion** criteria • Oracle validity
Inadequate CM	Requirements, design, implementation, ***and*** testing work products

Source – General Testing Problems – 2

Problem	Description
Political definition of defect	Administration of the defect database (trouble reports, etc.) can be political: • What constitutes a bug or a defect worth reporting? • Developers and testers are often at odds about this.
Defects not detected prior to testing	Excessive requirements, architecture, and design defects are not detected and fixed prior to testing.
Unrealistic expectations	Testing is not the only verification method.
Over reliance on testing	• Testing only finds a fixed percentage of existing defects. • Testing is not the most effective way of minimizing defects.
False sense of security	• Positive test results may mean good software or poor tests. • A truly successful test run finds a high percentage of the remaining significant defects.

Source – General Testing Problems – 3

Problem	Description
Testing not prioritized	Some testing my need to be sacrificed because the program is behind schedule and over budget. Tests that would find the most important bugs may be deleted.
Inadequate testing not identified as a risk	Inadequate testing is rarely an official risk in the program risk repository.

Poor communication	• Large programs with many teams
	• Geographically distributed programs
	• Contractually separated teams (prime vs. subcontractor, system of systems)
	• Between testers and:
	  • Other developers (requirements engineers, architects, designers, and implementers)
	  • Other testers
	  • Customers and subject matter experts (SMEs)

Source – Test Planning Problems – 1

Problem	Description
No test plan	There is no separate Test and Evaluation Master Plan (TEMP): only incomplete high-level overviews in System Engineering Master Plans (SEMPs) and Software Development Plans (SDPs).
What, not how	Test plans do not specify test processes. They only list and briefly describe types of testing, not how to perform this testing.
Incomplete test plans	Test plans have no clear specific: • Test objectives/no acceptance criteria • Testing techniques (testing is ad hoc) • Test case selection criteria (e.g., single test case vs. boundary value testing) • Test completion criteria including nominal and off-nominal testing (except possibly at the unit test level)

Source – Test Planning Problems – 2

Problem	Description
Inadequate planned test resources	The test plans provide inadequate test resources: • Inadequate test time in schedule with inadequate schedule reserves • Insufficient and inadequately trained and experienced testers and reviewers • Inadequate funding • Insufficient test environments (test beds) for integration testing
One size fits all	Mission-, safety-, and security-critical software is not tested more completely and rigorously (except possibly unit testing).

Source – Modern Life Cycle Problems – 1

Problem	Description
Evolutionary development cycle • Iterative (fixing requirements, architecture, design, and implementation defects) • Incremental (additional capabilities with each new increment) • Parallel (requirements, architecture, design, implementation, integration, testing activities) • Time-boxed (limited time in which to perform activities including testing)	• Regression Testing is Inadequate (amount needed is greatly increased). • Automation of regression testing is inadequate. • Regression tests are not developed iteratively and incrementally. • Refactoring may inadvertently obsolete tests. • Refactoring may delete test hooks.
Long operational lifespans	• Testing assets (test documents, environments, and test cases) are not properly documented (Agile). • Testing assets are not adequately maintained (fixed and updated).

Source – Requirements & Testing Problems – 1

Problem	Description
Ambiguous requirements	Ambiguous requirements cause incorrect test cases as testers misinterpret requirement's intent.
Missing requirements	Missing requirements cause incomplete testing (missing test cases): • Over emphasis on nominal over off-nominal behavior • Normal (sunny day) as opposed to fault tolerant and failure (rainy day) use case paths • Lack of fault and failure detection and reaction requirements
Poor quality "requirements"	Vague, high-level architecturally-significant goals rather than verifiable requirements cause missing or inadequate specialty-engineering testing (discussed later).

Source – Requirements & Testing Problems – 2

Problems	Description
Poor safety requirements	Inadequate derivation of system/software requirements from combined process/product standards as DO-178C and DO-254 can lead to missing requirements that are then not properly tested.
Unstable requirements	Unstable requirements cause inadequate regression testing. This is worsened by inadequate automation of regression tests.
Inadequate test oracles	Testing assumes the correctness of test oracles (i.e., requirements, architecture, and design). Building tests based on inadequate oracles provides false data.

Test Type – Unit Testing Problems

Problem	Description
Inadequate low-level requirements	The derived requirement merely restates its parent requirement. Allocated requirements are not derived to be at the proper level of abstraction.
Inadequate detailed design	There is insufficient detail to drive testing. An unstable design leads to obsolete testing.
Wrong test mindset	If the tester is the developer, testing will often be to show correctness, not find defects.
Lack of pre-test peer review	There is no pre-test peer review of the test input, preconditions (pre-test state), and test oracle (expected test outputs and postconditions).
Inadequate regression testing before integration	Modified code (bug fixes and changed design) is not retested prior to integration testing.
Different test environment	Unit is not compiled using the same compiler (version) and toolset as the actual product software.

Test Type – Integration Testing Problems – 1

Problem	Description
Insufficient integration test environments (test beds)	Lack of sufficient test beds due to cost or hardware needed elsewhere causes competition for limited resources among test teams.
Insufficient schedule	Insufficient time scheduled for adequate testing (especially regression testing).
Poor test bed fidelity	Insufficient software, hardware, and system fidelity to actual system (e.g., due to inadequate software simulations/drivers, prototype or wrong version).
Poor test bed quality	Test beds contain too many defects, slowing testing.
Unavailable components	Not all the components to be integrated are ready at the proper time.
Poor code quality	Code contains many defects that should have been found during lower-level testing, which unnecessarily slows integration testing.

Test Type – Integration Testing Problems – 2

Problem	Description
Insufficient/inadequate test metrics	Should be more than just defects found and closed. Also need estimated defects remaining.
Schedule conflicts	Inadequate scheduling of limited test environments.
Inadequate BIT and PHM	Inadequate Built-In Test (BIT) and Prognostics and Health Management (PHM) makes localizing defects more difficult.

Test Type – Specialty Engineering Testing

Problem	Description
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.
Inadequate **robustness** testing	There is little or no testing of environmental, error, fault, and failure tolerance based on robustness analysis (fault, degraded mode, and failure use case paths as well as ETA, FTA, FMECA, RBD, etc.).
Inadequate **safety** testing	There is little or no testing of safeguards (e.g., interlocks) and fail-safe behavior based on safety analysis (mishap cases).
Inadequate **security** testing	There is little or no penetration testing, security features, or testing of security controls and fail-secure behavior based on security analysis (e.g., attack trees, misuse cases).
Inadequate **usability** testing	There is little or no explicit usability testing of human interfaces for user friendliness, learnability, etc.

Test Type – System Testing Problems

Problem	Description
Ambiguous requirements	Ambiguous requirements are easily misunderstood and difficult or impossible to verify.
Incomplete requirements	Many requirements are incomplete (e.g., missing preconditions and trigger events) making them difficult to verify.
Missing requirements	Many requirements are missing (e.g., quality requirements and requirements for off-nominal behavior) and therefore unverifiable.
System testing is performed too late	Testing is performed just before delivery when it is too late to provide adequate time for defect removal and regression testing.
Poor code quality	Code contains many defects that should have been found during lower-level (unit and integration) testing, which unnecessarily slows system testing.

Test Type – SoS Testing Problems – 1

Problem	Description
Inadequate SoS planning	Test planning often does not exist at the SoS level. There are no clear test completion/acceptance criteria.
Poor or missing SoS requirements	Requirements only exist at the system level, leaving no clear and explicit requirements to be verified at the SoS level.
Unclear testing responsibilities	No program is explicitly tasked with testing end-to-end SoS behavior.
SoS not funded	No program is funded to perform end-to-end SoS testing.
SoS not properly scheduled	SoS testing is not in the system integrated master schedules and depends on uncoordinated system schedules.
Inadequate test resources	Hard to obtain test resources (people, test beds) from individual programs

Test Type – SoS Testing Problems – 2

Problem	Description
Poor code quality	Code contains many defects that should have been found during lower-level (unit, integration, and system) testing, which unnecessarily slows SoS testing.
Poor bug tracking across systems	Poor coordination of bug tracking and associated regression testing across multiple programs.
Finger-pointing	Makes it hard to improve SoS testing process

Test Type – Regression Testing Problems

Problem	Description
Insufficient test automation	Sufficient regression testing will only be performed if it is relatively quick and painless.
Insufficient schedule	Iterative, incremental, and concurrent development/life cycle greatly increases amount of regression testing.
Regression tests not updated	Regression test suite (test inputs, test drivers, test stubs, oracles) is not updated for new/changed capabilities.
Documentation not maintained	Requirements, user documents, test plans, entry/exit criteria and errata lists are not updated to reflect changes.
Regression tests not rerun	“There was only one minor code change, so we didn’t do the regression testing.” “There isn’t enough time.”
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.

Test Type – Maintenance Testing Problems

Problem	Description
Regression tests not maintained	Regression test suite (test inputs, test drivers, test stubs, oracles) is not updated for new/changed capabilities.
Test documentation not maintained	Test plans, test procedures, and other documentation are not maintained as changes occur.
Test work products not under CM	The test documentation and software are not under configuration control
Test work products inconsistent with SW	The test work products become out of synch with the SW as it evolves over time.
Disagreements over funding	It is unclear where the funding 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
- Consultants

Use Common Testing Problems as a checklist:

- Require the prevention/mitigation of these problems in the RFPs
- Evaluate proposed solutions to these problems in contractor proposals
- Evaluate test documentation for solutions to these problems:
  - Test program plans
  - Testing sections of SEMPs and SDPs

Do *not* accept inadequate contractor/subcontractor testing documents

Evaluate contractor/subcontractor implementations of test plans and processes

Push back against testing shortcuts late in the program.

Contact Information Slide Format

Donald Firesmith
Senior Member Technical Staff
Acquisition Support Program
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

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.