Common Testing Problems: Pitfalls to Prevent and Mitigate – Checklists of Symptoms, Consequences, and Recommendations

Donald G. Firesmith

12 November 2012

Acquisition Support Program
Unlimited distribution subject to the copyright.

http://www.sei.cmu.edu
This document was prepared for the
SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the
interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally
funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2012 Carnegie Mellon University.

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 report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for
internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions
and derivative works.

External use. This document 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 external
and/or commercial 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.

For information about SEI publications, please visit the library on the SEI website (www.sei.cmu.edu/library).
Table of Contents

1 Introduction ............................................................................................................................ 6
  1.1 Checklists ........................................................................................................................................ 6
  1.2 Checklist Interpretation ............................................................................................................. 7
2 Checklists ................................................................................................................................ 7
  2.1 General Testing Problems .................................................................................................... 7
    2.1.1 GEN-1 Wrong Testing Mindset .................................................................................. 8
    2.1.2 GEN-2 Unrealistic Testing Expectations / False Sense of Security ...................... 9
    2.1.3 GEN-3 Inadequate Whitebox Testing .......................................................................... 10
    2.1.4 GEN-4 Too Immature for Testing ............................................................................. 11
    2.1.5 GEN-5 Testing is Postponed ..................................................................................... 11
    2.1.6 GEN-6 Inadequate Testing Expertise .......................................................................... 12
    2.1.7 GEN-7 Inadequate Test Schedule ............................................................................. 13
    2.1.8 GEN-8 Testing Process Not Integrate Into Engineering Process ....................... 14
    2.1.9 GEN-9 Inadequate Test Documentation .................................................................. 15
    2.1.10 GEN-10 Inadequate Test Evaluations ........................................................................ 16
    2.1.11 GEN-11 Inadequate Test Metrics .............................................................................. 17
    2.1.12 GEN-12 Inadequate Test-related Risk Management ........................................ 18
    2.1.13 GEN-13 Tests not Delivered .................................................................................... 18
    2.1.14 GEN-14 Inadequate Test Maintenance .................................................................... 19
    2.1.15 GEN-15 Inadequate Test Prioritization .................................................................... 19
    2.1.16 GEN-16 Inadequate Test Configuration Management (CM) ............................ 20
    2.1.17 GEN-17 Lack of Requirements Trace ...................................................................... 20
    2.1.18 GEN-18 Over-reliance on COTS Testing Tools ..................................................... 21
    2.1.19 GEN-19 Inappropriate External Influences .......................................................... 22
    2.1.20 GEN-20 Inadequate Communication Concerning Testing ................................ 23
    2.1.21 GEN-21 Test Lessons Learned are Ignored ............................................................ 24
  2.2 Test Planning Problems ...................................................................................................... 24
    2.2.1 PLN-1 No Separate Test Plan .................................................................................. 24

© 2012 by Carnegie Mellon University
2.2.2 PLN-2 Incomplete Test Planning ................................................................. 25
2.2.3 PLN-3 Unclear Testing Responsibilities ......................................................... 26
2.2.4 PLN-4 One-Size-Fits-All Test Planning ......................................................... 27
2.2.5 PLN-5 Inadequate Test Resources Planned .................................................... 27

2.3 Requirements-Related Problems ........................................................................ 28
2.3.1 REQ-1 Ambiguous Requirements ................................................................. 28
2.3.2 REQ-2 Missing Requirements ........................................................................ 29
2.3.3 REQ-3 Incomplete Requirements .................................................................... 30
2.3.4 REQ-4 Incorrect Requirements ....................................................................... 30
2.3.5 REQ-5 Unstable Requirements ....................................................................... 31
2.3.6 REQ-6 Poorly Derived Requirements .............................................................. 31
2.3.7 REQ-7 Verification Methods Not Specified ..................................................... 31

2.4 Unit Testing Problems .......................................................................................... 32
2.4.1 UNT-1 Unstable Design .................................................................................. 32
2.4.2 UNT-2 Inadequate Design Detail ...................................................................... 33
2.4.3 UNT-3 Poor Fidelity of Test Environment ....................................................... 33

2.5 Integration Testing Problems ............................................................................... 33
2.5.1 INT-1 Defect Localization ............................................................................... 34
2.5.2 INT-2 Insufficient Test Environments .............................................................. 34
2.5.3 INT-3 Unavailable Components ...................................................................... 35
2.5.4 INT-4 Inadequate Test Bed Quality ................................................................. 35
2.5.5 INT-5 Inadequate Self-Test ............................................................................. 36

2.6 Specialty Engineering Testing Problems .............................................................. 36
2.6.1 SPC-1 Inadequate Capacity Testing ................................................................. 37
2.6.2 SPC-2 Inadequate Concurrency Testing .......................................................... 37
2.6.3 SPC-3 Inadequate Performance Testing ............................................................ 38
2.6.4 SPC-4 Inadequate Reliability Testing ............................................................... 39
2.6.5 SPC-5 Inadequate Robustness Testing ............................................................. 39
2.6.6 SPC-6 Inadequate Safety Testing .................................................................... 40
Common Testing Problems: Pitfalls to Prevent and Mitigate
Checklists of Symptoms, Consequences, and Recommendations

2.6.7 SPC-7 Inadequate Security Testing ................................................................. 41
2.6.8 SPC-8 Inadequate Usability Testing ................................................................. 41

2.7 System Testing Problems ...................................................................................... 42
2.7.1 SYS-1 Testing Robustness Requirements is Difficult ...................................... 42
2.7.2 SYS-2 Lack of Test Hooks ............................................................................... 43
2.7.3 SYS-3 Testing Code Coverage is Difficult ...................................................... 43

2.8 System of Systems (SoS) Testing Problems .......................................................... 44
2.8.1 SoS-1 Inadequate SoS Planning ................................................................. 44
2.8.2 SoS-2 Poor or Missing SoS Requirements ...................................................... 45
2.8.3 SoS-3 Unclear SoS Testing Responsibilities .................................................. 45
2.8.4 SoS-4 Inadequate Funding for SoS Testing ..................................................... 45
2.8.5 SoS-5 SoS Testing not Properly Scheduled ...................................................... 46
2.8.6 SoS-6 Inadequate Test Support from Individual Systems ............................... 46
2.8.7 SoS-7 Inadequate Defect Tracking Across Projects ...................................... 46
2.8.8 SoS-8 Finger-Pointing .................................................................................... 47

2.9 Regression Testing Problems ............................................................................. 47
2.9.1 REG-1 Insufficient Regression Test Automation ............................................ 47
2.9.2 REG-2 Regression Tests Not Rerun .............................................................. 48
2.9.3 REG-3 Inadequate Scope of Regression Testing ............................................. 48
2.9.4 REG-4 Only Low-Level Regression Tests ..................................................... 49
2.9.5 REG-5 Disagreement over Maintenance Test Resources ............................... 49

3 Conclusion ........................................................................................................... 50
3.1 Testing Problems .................................................................................................. 50
3.2 Common Consequences ....................................................................................... 50
3.3 Common Solutions ............................................................................................... 51
3.4 Potential Future Work .......................................................................................... 52

4 Acknowledgements ............................................................................................... 52
1 Introduction

There are many testing problems that 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 problems, most projects will suffer several of them. Similarly, while exhibiting these testing problems does not guarantee failure, these problems definitely pose risks that need to be managed.

In this document, the problems involving how testing is performed in practice have been grouped into the following categories:

1. **General Testing Problems** – 23 problems of a general nature not restricted to a specific type or scope of testing.
2. **Test Planning Problems** – 5 problems that occur due to inadequate test planning
3. **Requirements-related Problems** – 7 testing problems due to poor requirements
4. **Unit Testing Problems** – 3 problems specific to unit testing
5. **Integration Testing Problems** – 5 problems specific to integration testing
6. **Specialty Engineering Testing Problems** – 8 problems specific to the specialty-engineering testing of quality requirements
7. **System Testing Problems** – 3 problems specific to the testing of complete systems
8. **System of System Testing Problems** – 8 problems specific to the testing of systems of systems
9. **Regression Testing Problems** – 5 problems specific to the performance of regression testing including testing during maintenance

1.1 Checklists

Each of the above nine sets of commonly occurring testing problems has been turned into a checklist, which can be used during the development and review of test plans and test process documents as well the testing sections of system engineering management plans (SEMPs) and software development plans (SDPs). These checklists can also be used during the oversight and evaluation of the actual testing. While the checklist results are intended to be used to help identify possible testing risks and thus the probable need to fix the specific problems found, these results are not intended for use as input to some quantitative scoring scheme.

Although each of these testing problems has been observed on multiple projects, there is no guarantee that the set is exhaustive. It is entirely possible that you may have testing problems not addressed by this document.

In the following checklists, each testing problem is documented with the following information:

- **Title** – a short descriptive name that identifies the problem
Common Testing Problems: Pitfalls to Prevent and Mitigate
Checklists of Symptoms, Consequences, and Recommendations

- **Description** – a brief descriptive definition of the problem
- **Symptoms** (how you will know) – a list of symptoms that the problem may exist
- **Potential Consequences** (why you should care) – a list of the potential negative consequences to expect if the problem exists
- **Recommendations** (what you should do) – a list of recommended actions to take to help solve the problem

### 1.2 Checklist Interpretation

The goal of testing is not to prove that something works, but rather to demonstrate that it does not.⁴ A good tester assumes that there are always defects (an extremely safe assumption), and it is the tester’s responsibility to uncover them. Thus, a good test is one that causes the thing being tested to fail so that the underlying defect(s) can be found and fixed.

Note that defects are not restricted to violations of specified (or unspecified) requirements. Some of the other important types of defects are:

- inconsistencies between the implementation and either the architecture or the design
- violations of coding standards
- the unnecessary inclusion of safety or security vulnerabilities (e.g., the use of inherently unsafe language features or lack of verification of input data)

Given that testers are looking for problems, it thus seems fitting that these testing checklists are designed to help identify testing problems rather than to show that no such testing problems exist. A “yes” result for symptoms or consequences signifies that a potential problem has been found, not that the absence of a problem has been shown. Just as a failed test should not be viewed negatively but rather as a positive indication of a defect that can now be fixed, a “yes” result on the checklists should also be viewed as a “positive” result in the following sense: a previously unknown problem is now known to exist and can therefore be fixed, which is surely a positive step forward.

### 2 Checklists

#### 2.1 General Testing Problems

The following testing problems are quite general and commonly observed regardless of the type of testing being performed:

---

⁴ Although positive testing results are often used as evidence that the system (or subsystem) under test meets its (derived and allocated) requirements, testing can almost never be exhaustive and cannot “prove” that the requirements are being met. Similarly, testing (especially system and operational testing) can also provide evidence that the system under test is “fit for purpose” and therefore ready to be placed into operation. For example, certain testing provides evidence that can be used for safety and security accreditation and certification. Nevertheless, a tester must exhibit a “show it fails” rather than a “show it works” mindset to be effective.
2.1.1 GEN-1 Wrong Testing Mindset

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

**Symptoms:** Some testers and other testing stakeholders assume that the system/software works. Testers assume or are told that their job is to verify or “prove” that the system/software works. Testing is being used to demonstrate that the system/software works properly rather than to determine where and how it fails. Only normal (“sunny day” or “happy path”) behavior is being tested. There is little or no testing of exceptional or fault/failure tolerant (“rainy day”) behavior. There is no testing of input data (e.g., range testing of the handling of invalid input values). Test input includes

---

2 Using testing to “prove” that their software works is most likely to become a problem when developers test their own software (e.g., with unit testing and with small cross-functional or agile teams).
only middle of the road values rather than boundary values and corner cases.

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

Recommendations: Explicitly state in the project test plan that the primary goal of testing is to find defects by causing the system to fail (i.e., to break the system) rather than to demonstrate that there are no defects (i.e., to show that it works). Provide test training that emphasizes the proper testing mindset. In addition to test cases that verify all nominal behavior, emphasize looking for defects where they are most likely to hide (e.g., boundary values and corner cases).

2.1.2 GEN-2 Unrealistic Testing Expectations / False Sense of Security

Description: Testers and other testing stakeholders have unrealistic testing expectations that generate a false sense of security.

Symptoms: Testers and other testing stakeholders (e.g., managers and customer representatives) falsely believe that:

- Testing detects all (or even the majority of) defects. Testing proves that there are no remaining defects and that the system therefore works as intended.
- Testing can be, for all practical purposes, exhaustive.
- Testing can be relied on for all verification. (Note that some requirements are better verified via analysis, demonstration, certification, and inspection)
- Testing (if it is automated) will guarantee the quality of the tests and reduce the testing effort.

Managers and other testing stakeholders do not understand that:

---

3 Whereas tests that verify nominal behavior are essential, testers must keep in mind that there are typically many more ways for the system/software under test to fail than to work properly. Also, nominal tests must remain part of the regression test suite even after all known defects are fixed because changes could introduce new defects that cause nominal behavior to fail.

4 Testing typically finds less than half of all latent defects and is not the most efficient way of detecting many defects.

5 This depends on the development cycle and the volatility of the system’s requirements, architecture, design, and implementation.
Test automation requires specialized expertise and needs to be budgeted for the effort required to develop, verify, and maintain the automated tests.

- A passed test could result from a weak/incorrect test rather than a lack of defects.
- A truly successful/useful test is one that finds one or more defects, whereas a passed test only shows that the system worked in that single specific instance.

**Potential Consequences:** Testers and other testing stakeholders will have a false sense of security that the system or software will function properly on delivery and deployment. Non-testing forms of verification (e.g., analysis, demonstration, inspection, and simulation) will not be given adequate emphasis.

**Recommendations:** Ensure via training and consulting that managers, customer representatives, testers, and other test stakeholders understand that:

- Testing will not detect all (or even a majority of) defects
- No testing is truly exhaustive
- Testing cannot prove (or demonstrate) that the system works under all combinations of preconditions and trigger events.
- A passed test could result from a weak test rather than a lack of defects.
- A truly successful test is one that finds one or more defects.

Do not rely on testing for the verification of all requirements, especially architecturally-significant quality requirements.

### GEN-3 Inadequate Whitebox Testing

**Description:** The amount of unit- and integration-level whitebox testing is inadequate to prevent large numbers of residual defects from slipping past to blackbox system testing.

**Symptoms:** System testing is identifying significant numbers of defects that should have been found during unit and integration-testing. Similar residual defects are also causing faults and failures after the system has been delivered and placed into operation.

**Potential Consequences:** System testing is unlikely to be completed on schedule. Testers and developers will have a harder time localizing the defects that the system tests reveal. Testers will need to work excessively long hours. The system or software may be delivered late with an unnecessarily large number of residual defects. The costs of finding and fixing
defects will rise when defects are discovered later than necessary during the testing process.

**Recommendations:** Increase the amount and effectiveness of unit testing and integration testing. Review the test plans and process documents to ensure that they adequately address lower-level testing. When appropriate, improve the test plans and process documents with regard to unit and integration testing. Measure the number of defects slipping past unit and integration testing.

### 2.1.4 GEN-4 Too Immature for Testing

**Description:** Some of the products being tested are immature, containing too many defects.

**Symptoms:** Large numbers of requirements, architecture, and design defects are being found that should have been discovered (during reviews) and fixed prior to current testing. The product is delivered for testing when it is not ready for testing because:

- Schedule pressures causes corners to be cut during earlier testing.
- Test readiness criteria do not exist or are not enforced.
- Management, customer/user representatives, and developers do not understand the impact on testing of immature products.

**Potential Consequences:** Testing will find many defects that should have been detected during previous levels of testing. Encapsulation due to integration will make it unnecessarily difficult to localize the defect that caused the test failure. Testing may not be completed on schedule.

**Recommendations:** Set and enforce reasonable criteria for test readiness. Increase the amount of earlier verification of the requirements, architecture, and design (e.g., with peer-level reviews and inspections). Improve the effectiveness of earlier disciplines and types of testing (e.g., by improving methods and providing training). Measure the number of defects slipping through multiple disciplines and types of testing (e.g., where the defect was introduced and where it was found).

### 2.1.5 GEN-5 Testing is Postponed

**Description:** Testing is postponed until late in the development schedule.

**Symptoms:** Testing is scheduled to be performed late in the development cycle on the project master schedule. Little or no unit or integration testing is planned or is being performed during the early and middle stages of the
Common Testing Problems: Pitfalls to Prevent and Mitigate
Checklists of Symptoms, Consequences, and Recommendations

<table>
<thead>
<tr>
<th>Development cycle.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Potential Consequences</strong>: It is very difficult to find and localize defects that remain hidden within the internals of the system. It is difficult to show the required degree of test coverage. There is no time left in the schedule to correct any major defects found.(^6)</td>
</tr>
<tr>
<td><strong>Recommendations</strong>: Plan and schedule testing to be performed in an iterative, incremental, and parallel manner during the vast majority of the development cycle. Verify the proper performance of this testing during major project milestones. Evaluate test metrics during the entire development process.</td>
</tr>
<tr>
<td>Observed</td>
</tr>
</tbody>
</table>

### 2.1.6 GEN-6 Inadequate Testing Expertise

<table>
<thead>
<tr>
<th>Description: Too many people have inadequate testing expertise, experience, and training.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms</strong>: Testers and/or those who oversee them (e.g., managers and customer representatives) have inadequate testing expertise, experience, or training. Developers who are not professional testers are tasked to perform testing. Little or no classroom or on-the-job training in testing has taken place. Testing is ad hoc without any proper process. Best practices are not being followed.</td>
</tr>
<tr>
<td>Observed</td>
</tr>
<tr>
<td><strong>Potential Consequences</strong>: Testing will not be effective in detecting defects, especially the less obvious ones. The productivity of the testers will be needlessly low. There is a high probability that the system or software will be delivered late with an unnecessarily large number of residual defects. During development, managers, developers, and customer representatives will have a false sense of security that the system functions properly(^7).</td>
</tr>
<tr>
<td>Observed</td>
</tr>
<tr>
<td><strong>Recommendations</strong>: Hire full time (i.e., professional) testers who have sufficient expertise and experience in testing. Obtain independent support for those overseeing testing. Provide appropriate amounts of test training (both classroom and on-the-job) for both testers and those overseeing testing. Provide proper test processes including procedures, standards, guidelines,</td>
</tr>
<tr>
<td>Implemented</td>
</tr>
</tbody>
</table>

---

\(^6\) An interesting example of this is the Hubble telescope. Testing of the mirror’s focusing was postponed until after launch, resulting in an incredibly expensive repair mission.

\(^7\) This false sense of security is likely to be replaced by a sense of panic when the system begins to frequently fail operational testing or real-world usage after deployment.
and templates for On-The-Job training. Ensure that testers automating testing have the necessary specialized expertise and training.\(^8\) Use an independent test organization staffed with experienced trained testers for system/acceptance testing, whereby the head of this organization is at the same (or higher) level as the project manager.

### 2.1.7 GEN-7 Inadequate Test Schedule

| **Description:** The testing schedule is inadequate to permit proper testing. |
| **Symptoms:** Testing is incomplete because there is insufficient time allocated in the project master schedule to perform all test activities (e.g., automating testing, configuring test environments, and developing test data and test scripts/drivers and test stubs) and to perform all appropriate tests (e.g., abnormal behavior, quality requirements, regression testing).\(^9\) Testers are working excessively and unsustainably long hours and days per week in an attempt to meet schedule deadlines. |
| **Potential Consequences:** Testers will become exhausted and will make an unnecessarily large number of mistakes. Tester productivity (e.g., importance of defects found and number of defects found per unit time) will decrease. Customer representatives, managers, and developers will have a false sense of security that the system functions properly. There is a significant probability that the system or software will be delivered late with an unnecessarily large number of residual defects. |
| **Recommendations:** Ensure that adequate time for testing is included in the program master schedule and test team schedules including the testing of abnormal behavior and the specialty engineering testing of quality requirements.\(^10\) Deliver inputs to the testing process (e.g., requirements, architecture, design, and implementation) earlier and more often (e.g., as part of an incremental, iterative, parallel—agile—development cycle). Provide adequate time for testing in change request estimates. Provide evidence-based estimates of the amount of testing and associated test effort. |

---

\(^8\) Note that these recommendations apply, regardless of whether the project uses separate testing teams or cross functional teams including testers.

\(^9\) Note that an agile (i.e., iterative, incremental, and concurrent) development/life cycle greatly increases the amount of regression testing needed (although this increase in testing can be largely offset by highly automating regression tests). Although testing can never be exhaustive, more time is typically needed for adequate testing unless testing can be made more efficient. For example, fewer defects could be produced and these defects can be found and fixed earlier and thereby be prevented from reaching the current testing.

\(^10\) Also integrate the testing process into the software development process.
Common Testing Problems: Pitfalls to Prevent and Mitigate
Checklists of Symptoms, Consequences, and Recommendations

<table>
<thead>
<tr>
<th>that will be needed. Provide sufficient test resources (e.g., number of testers, test environments, and test tools). Automate as much of the regression testing as is practical.¹¹ Do not reduce the testing effort in order to meet a delivery deadline.</th>
</tr>
</thead>
</table>

### 2.1.8 GEN-8 Testing Process Not Integrated Into Engineering Process

**Description:** The testing process is not integrated into the overall system/software engineering process.

**Symptoms:** There is little or no discussion of testing in the system/software engineering process documentation. All or most of testing is done as a completely independent activity performed by staff members who are not part of the project engineering team. Testing is treated as a separate specialty-engineering activity with only limited interfaces with the primary engineering activities. Testers are not included in the requirements teams, architecture teams, and any cross functional engineering teams.

**Potential Consequences:** There is inadequate communication between testers and other system/software engineers (e.g., requirements engineers, architects, designers, and implementers). Few testing outsiders will understand the scope, complexity, and importance of testing. Testers will not understand the work being performed by other engineers. There will be incompatibilities between outputs and associated inputs at the interfaces between testers and other engineers. Testing will be less effective and take longer than necessary.

**Recommendations:** In addition to being in test plans such as the Test and Evaluation Master Plan (TEMP) or Software Test Plan (STP) as well as in other process documents, provide high-level overviews of testing in System Engineering Master Plans (SEMPs) and Software Development Plans (SDPs). Incorporate testing into the Master Project Schedule. Incorporate testing into the project work breakdown structure (WBS). Have test subject matter experts and project testers collaborate closely with the project chief engineer / technical lead and process engineer when they develop the engineering process descriptions and associated process documents. Document how

---

¹¹ When there is insufficient time to perform manual testing, it may be difficult to justify the automation of these tests. However, automating regression testing is not just a maintenance issue. Even during initial development, there should typically be a large amount of regression testing, especially if an iterative and incremental development cycle is used. Thus, ignoring the automation of regression testing is often a case of being penny wise and pound foolish.
testing is integrated into the system/software development/life cycle (regardless of whether it is traditional waterfall or agile (iterative, incremental, and parallel), or anything in between).

### 2.1.9 GEN-9 Inadequate Test Documentation

**Description**: Test documentation is inadequate for defect identification and analysis, regression testing, test automation, reuse, and quality assurance of the testing process.\(^\text{12}\)

**Symptoms**: Test (a.k.a., defect/bug/trouble) reports do not contain sufficient detail to enable developers to reproduce to faults/failures and thereby identify the underlying defects. Different testers and test teams use different test report templates. Testing assets (e.g., test documents, environments, and test cases) are not sufficiently documented to be used by:

- testers to drive test automation
- testers to perform regression testing, either during initial development or during maintenance
- quality assurance personnel and customer representatives during evaluation and oversight of the testing process
- testers other than the original test developer (e.g., by those performing integration, system, system of system, and maintenance testing)
- test teams from other projects developing/maintaining related systems within a product family or product line

Test cases do not completely describe test preconditions, test trigger events, test input data, mandatory/expected test outputs (data and commands), and mandatory/expected system post-conditions.

**Potential Consequences**: Developers will be unable to reproduce some faults/failures. It will take longer to identify and fix some of the underlying defects. Test deadlines will be missed. Maintenance costs will be needlessly high. Insufficient regression testing may be performed. The reuse of testing assets will be needlessly low, thereby unnecessarily increasing the costs, schedule, and effort that will be spent recreating testing assets.

**Recommendations**: Use the contract, test plans, test training, test process documents, and test standards to specify the required test documents and ensure that test work products are adequately documented. Ensure that test

---

\(^{12}\) This is often cause by managers attempting to decrease the testing effort and thereby meet schedule deadlines or by processes developed by people who do not have adequate testing training and experience.
cases completely describe test preconditions, test trigger events, test input data, mandatory/expected test outputs (data and commands), and mandatory/expected system post-conditions. When using an iterative, incremental, and parallel – agile – development cycle in which components under test will frequently change, concentrate on making their associated executable testing work products self-documenting (rather than using separate testing documentation) so that the components and their testing work products are more likely to be changed together and thereby remain consistent. Use common standard templates for test documents (e.g., test plans, test cases, test procedures, and test reports). Use a test documentation tool or database to record test reports. When using a database to store test results, make sure that its schema supports easy searches. Clearly identify the versions of the software, test environment, test cases, etc. to use to ensure consistency.

2.1.10 GEN-10 Inadequate Test Evaluations

<table>
<thead>
<tr>
<th>Description:</th>
<th>The quality of the test assets is not being adequately evaluated prior to their use.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symptoms:</td>
<td>Little or no [peer-level] inspections, walk-throughs, or reviews of the test assets (e.g., test inputs, preconditions, trigger events, expected test outputs and postconditions) are being performed prior to actual testing.</td>
</tr>
<tr>
<td>Potential Consequences:</td>
<td>Test plans, procedures, test cases, and other testing work products will contain defects that could have been found during these evaluations. There will be an increase in false positive and false negative test results. Unnecessary effort will be wasted identifying and fixing problems. Some defects will not be found, and an unnecessary number of these defects may make it through testing and into the deployed system.</td>
</tr>
<tr>
<td>Recommendations:</td>
<td>Incorporate test evaluations into (1) the system/software development process documents, (2) the project schedules (master and team), and the project work breakdown structure (WBS). Ensure that the following test assets are reviewed prior to actual testing: test inputs, preconditions (pre-test state), and test oracle including expected test outputs and postconditions. To the extent practical, ensure that the test evaluation team includes other testers, requirements engineers, user representatives, subject matter experts, architects, and implementers.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observed</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Implemented</th>
</tr>
</thead>
</table>

© 2012 by Carnegie Mellon University
## 2.1.11 GEN-11 Inadequate Test Metrics

<table>
<thead>
<tr>
<th>Description: Insufficient test metrics are being produced, analyzed, and reported.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms:</strong> Insufficient or no test metrics are being produced, analyzed, and reported. The primary test metrics (e.g., number of tests, number of tests needed to meet adequate or required test coverage levels, number of tests passed/failed, 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 undiscovered defects remaining is not estimated (e.g., using COQUALMO). Management measures tester productivity strictly in terms of defects found per unit time, ignoring the importance / severity of the defects found.</td>
</tr>
<tr>
<td><strong>Potential Consequences:</strong> Managers, testers, and other stakeholders in testing will not accurately know the quality of testing, the importance of the defects being found, or the number of residual defects in the delivered system or software. Managers will not know the productivity of the testers and their effectiveness at finding important defects, thereby making it difficult to improve the testing process. Testers may concentrate on finding lots of (unimportant) defects rather than finding critical defects (e.g., those with mission-critical and safety-critical ramifications). Customer representatives, managers, and developers may have a false sense of security that the system functions properly.</td>
</tr>
<tr>
<td><strong>Recommendations:</strong> Incorporate a robust metrics program in the test plan that covers leading indicators. Emphasize the finding of important defects. Some representative examples of useful testing metrics include the:</td>
</tr>
<tr>
<td>• number of defects found per test (test effectiveness metric)</td>
</tr>
<tr>
<td>• number of defects found per tester day (tester productivity metric)</td>
</tr>
<tr>
<td>• number of defects that slip through each verification milestone / inch pebble (e.g., reviews, inspections, tests)</td>
</tr>
<tr>
<td>• estimated number of latent undiscovered defects remaining in the delivered system (e.g., estimated using COQUALMO)</td>
</tr>
</tbody>
</table>

---

13 Note that the number of tests metric does not indicate the effort or complexity of identifying, analyzing, and fixing defects.

14 COQUALMO (COnstructive QUALity Model) is an estimation model that can be used for predicting the number of residual defects/KSLOC (thousands of source lines of code) or defects/FP (Function Point) in a software product.

15 For example, what are the percentages of defects that manage to slip by architecture reviews, design reviews, implementation inspections, unit testing, integration testing, and system testing without being detected?
process to ensure that it does not become metrics-driven. In other words, watch out for signs that testers worry more about looking good (e.g., by concentrating on only the defects that are easy to find) than on finding the most important defects.

### 2.1.12 GEN-12 Inadequate Test-related Risk Management

**Description:** There are too few test-related risks identified in the project’s official risk repository.  

**Symptoms:** Managers treat risk as a “four letter word”. There are little or no 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 probabilities, harm severities, and priorities. The identified test risks have no associated risk mitigation approaches and no one assigned as being responsible for the risk. The test risks are never updated (e.g., additions or modification) over the course of the project. Testing risks are not addressed in either the test plan(s) or the risk management plan.

**Potential Consequences:** Testing risks are not visible so that management and acquirer representatives are unaware of their existence. Test-related risks are not being managed or their management is not given sufficiently high priority.

**Recommendations:** Ensure that test-related risks are identified, incorporated into the official project risk repository. Ensure that test-related risks are provided realistic probabilities, harm severities, and priorities.

### 2.1.13 GEN-13 Tests not Delivered

**Description:** Test assets are not being delivered along with the system / software.

**Symptoms:** The delivery of tests (e.g., test cases, test oracles, test drivers/scripts, test stubs, and test environments) is neither required nor planned. Tests are not delivered along with the system / software.

**Potential Consequences:** It will be unnecessarily difficult to perform testing during maintenance. There will be inadequate regression testing as the

---

16 These potential testing problems can be viewed as generic testing risks.

17 Adding risks to the risk repository is looked on as a symptom of management failure. Therefore, risks (including testing risks) tend to be labeled as issues or concerns so that they need not be treated as an official risk.
delivered system/software is updated. Some post-delivery testing will not be performed so that some post-delivery defects may not being found and fixed.

**Recommendations:** Ensure that the migration to maintenance section of the system development contract or associated list of deliverables includes the delivery of all test work products needed to perform testing after delivery. **Implemented**

### 2.1.14 GEN-14 Inadequate Test Maintenance

<table>
<thead>
<tr>
<th><strong>Description:</strong> Testing assets are not being properly maintained.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms:</strong> Testing assets (e.g., test software and documents such as test cases, test procedures, test drivers, and test stubs) are not being adequately updated and iterated as defects are found and the system software is changed (e.g., due to refactoring or the use of an agile – incremental and iterative development cycle).</td>
</tr>
<tr>
<td><strong>Potential Consequences:</strong> Testing assets are no longer consistent with the current requirements, architecture, design, and implementation. Test productivity will decrease as the number of false negative test results increases (i.e., as tests fail due to test defects). The amount of productive regression testing will decrease as effort is redirect to identifying and fixing test defects.</td>
</tr>
<tr>
<td><strong>Recommendations:</strong> Ensure that testing assets (e.g., test software and documents such as test cases, test procedures, test drivers, and test stubs) are adequately maintained as defects are found and system changes are introduced. Ensure that testing assets remain consistent with the current requirements, architecture, design, and implementation. Ensure that regression test assets are updated as needed. Provide sufficient test resources (e.g., schedule and staffing) to maintain the automated test cases. Ensure that the maintenance testers are adequately trained and experienced. <strong>Implemented</strong></td>
</tr>
</tbody>
</table>

### 2.1.15 GEN-15 Inadequate Test Prioritization

| **Description:** Testing is not being adequately prioritized. |

---

18 While this is useful with regard to any product that undergoes multiple internal or external releases, it is especially a good idea when an agile (iterative and incremental) development cycle produces numerous short duration increments.

19 This will help combat the loss of project expertise due to the fact that many/most of the testers who are members of the development staff tend to move after delivery.
### Common Testing Problems: Pitfalls to Prevent and Mitigate

Checklists of Symptoms, Consequences, and Recommendations

<table>
<thead>
<tr>
<th>Symptoms: 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.</th>
<th>Observed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential Consequences: Limited testing resources will be wasted or ineffectively used. Some of the most critical defects (in terms of failure consequences) will not be discovered until after the system/software is delivered and placed into operation. Specifically, defects with mission-, safety-, and security-ramifications may not be found.</td>
<td>Observed</td>
</tr>
<tr>
<td>Recommendations: Prioritize testing according to the criticality (e.g., mission, safety, and security) of the subsystem or software being tested and the degree to which the test is likely to elicit important failures. Perform the highest priority tests of a given type first.</td>
<td>Implemented</td>
</tr>
</tbody>
</table>

#### 2.1.16 GEN-16 Inadequate Test Configuration Management (CM)

**Description:** Testing assets are not being properly placed under configuration control.

<table>
<thead>
<tr>
<th>Symptoms: Test plans, procedures, test cases, and other testing work products are not being placed under configuration control.</th>
<th>Observed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential Consequences: Test plans, test procedures, test cases, and other testing work products will cease to be consistent with the system/software being tested and with each other. It will be much more difficult to know that the correct versions of the system, test environment, and tests are being used when performing regression testing. There will be an increase in false positive and false negative test results. False positive test results due to incorrect version control may lead to incorrect fixes and the resulting insertion of defects into the system/software. Unnecessary effort will be wasted identifying and fixing CM problems. Some defects will not be found, and an unnecessary number of these defects may make it through testing and into the deployed system.</td>
<td>Observed</td>
</tr>
<tr>
<td>Recommendations: Ensure that all test plans, procedures, test cases, and other testing work products are placed under configuration control before they are used.</td>
<td>Implemented</td>
</tr>
</tbody>
</table>

#### 2.1.17 GEN-17 Lack of Requirements Trace

**Description:** The requirements are not traced to the individual test cases.
Common Testing Problems: Pitfalls to Prevent and Mitigate
Checklists of Symptoms, Consequences, and Recommendations

**Symptoms:** There is no requirements traceability matrix from the requirements to the test cases. The mapping from the requirements to the test cases is not stored in the project requirements repository (e.g., database or requirements management tool). There may only be a backwards trace from the individual test cases to the requirement(s) they test. Any tracing that was originally created is not maintained as the requirements change.

**Potential Consequences:** There will not be any easy way to determine if all requirements are being tested. If requirements change, there will be no way of knowing which test cases need to be created, modified, or deleted.

**Recommendations:** Include tracing to requirements in the requirements management tool/repository. If no such tool is being used, create a requirements traceability matrix documenting the trace from requirements to test cases. Include generating and maintaining the tracing from requirements to test cases in the test plan(s). Evaluate the testing process and work products to ensure that this tracing is being properly performed. Allocate time in the project master schedule to perform this training.

**2.1.18 GEN-18 Software Under Test Behaves Differently**

**Description:** The software under test and the operational software behave differently.

**Symptoms:** A fault or failure that occurs during testing is not repeatable during normal operation. Software that behaved correctly during test causes a fault or failure during operation. The software under test contains test software that is either removed (physically or via compiler switch) before being placed in operation.

**Potential Consequences:** Correct behavior due to the existence of integrated test software leads to a false sense of security.

**Recommendations:** Perform blackbox regression testing after removing the test software. Consider incorporating the test software as deliverable built-in-test (BIT) software.

**2.1.19 GEN-19 Over-reliance on Manual Testing**

**Description:** Testers are placing too much reliance on manual testing.

---

Note that it may not be practical (e.g., for performance reasons or code size) or permitted (e.g., for safety or security reasons) to deliver the system with embedded test software. For example, embedded test software could provide an attacker with a back door capability.
### Common Testing Problems: Pitfalls to Prevent and Mitigate

**Checklists of Symptoms, Consequences, and Recommendations**

<table>
<thead>
<tr>
<th><strong>Symptoms</strong></th>
<th><strong>Potential Consequences</strong></th>
<th><strong>Recommendations</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>All or the majority of testing is being performed manually without the support of test tools or test scripts.</td>
<td>Testing will be very labor intensive. Any non-trivial amount of regression testing will likely be impractical.</td>
<td>Limit manual testing to only the testing for which is most appropriate. Automate regression testing. Use test tools and scripts to automate appropriate parts of the testing process (e.g., to ensure that testing provides adequate code coverage).</td>
</tr>
</tbody>
</table>

#### 2.1.20 GEN-20 Over-reliance on COTS Testing Tools

**Description**: Testers and other testing stakeholders are placing too much reliance on testing tools.

**Symptoms**: Testers and other testing stakeholders are relying on testing tools to do far more than to merely generate sufficient whitebox test cases to ensure code coverage. Testers are relying on the tools to automate test case creation including test case selection and completion (“coverage”) criteria. Testers are relying on the test tools as their test oracle (to determine the expected – correct – test result). Testers let the tool drive the test methodology rather than the other way around.

**Potential Consequences**: Testing will emphasize white-box (design-driven) testing and will include inadequate black-box (requirements-driven) testing. Many design defects will not be found during testing and will remain in the delivered system.

**Recommendations**: Ensure that testers (e.g., via training and test planning) understand the limits of testing tools and the automation of test case creation. Ensure that testers need to use the requirements, architecture, and design as the test oracle (to determine the correct test result). Let the test methodology drive tool drive tool selection. Ensure that testers are not relying on test tools to automate test case selection and set the test completion (“coverage”) criteria.

#### 2.1.21 GEN-21 Inappropriate External Pressures

**Description**: Testers are subject to inappropriate external pressures, primarily from managers.

**Symptoms**: Managers (or developers) are dictating to the testers what constitutes a bug or a defect *worth reporting*. Managerial pressure exists to:

---

© 2012 by Carnegie Mellon University
• Inappropriately cut corners (e.g., only perform “sunny day” testing in order to meet schedule deadlines.
• Inappropriately lower the severity and priority of reported defects.
• Not find defects (e.g., until after delivery because the project is so far behind schedule that there is no time to fix any defects found).

**Potential Consequences:** If the testers yield to this pressure, then the test metrics will accurately reflect neither the true state of the system or software nor the status of the testing process. The delivered system or software may contain an unnecessarily large number of residual defects.

**Potential Consequences:**

<table>
<thead>
<tr>
<th>Implemented</th>
</tr>
</thead>
</table>

**Observed**

**Recommendations:** Ensure that trained testers determine what constitutes a bug or a defect *worth reporting*. Establish criteria for determining the priority and severity of reported defects. Support testers when they oppose any inappropriate managerial pressure that would have them violate their professional ethics. Customer representatives must insist on proper testing. Place the manager of the testing organization at the same or higher level as the project manager in the organizational hierarchy (i.e., have the test manager report independently of the project manager).

### 2.1.22 GEN-22 Inadequate Communication Concerning Testing

**Description:** There is inadequate communication concerning testing among testers and other testing stakeholders.

**Symptoms:** There is inadequate testing-related communication between:

- Teams within large or 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
  - Customer representatives, user representatives, and subject matter experts (SMEs)

**Potential Consequences:** Some of the requirements may not be testable. Some architectural decisions may make certain types of testing more difficult or impossible. Safety and security concerns may not influence the level of testing of safety- and security-critical functionality. Different test teams may have difficulty coordinating their testing and scheduling their use of common test environments.

**Potential Consequences:**

<table>
<thead>
<tr>
<th>Implemented</th>
</tr>
</thead>
</table>

**Observed**
Recommendations: Ensure that there is sufficient testing-related communication between and among the testers and the stakeholders in testing.

### 2.1.23 GEN-23 Test Lessons Learned are Ignored

**Description:** Lessons learned regarding testing are not placed into practice.

**Symptoms:** Lessons learned during previous projects or during the testing of previous increments of the system under test are ignored by management, the test teams, or customer representatives.

**Potential Consequences:** The test processes will not be improved and the same problems will continue to occur. Customer representatives, managers, and developers will have a false sense of security that the system functions properly. The system or software will be delivered with an unnecessarily large number of associated defects.

**Recommendations:** Customer representatives should explicitly request and look for evidence that previous lessons learned are incorporated into the official test plans and process. Capture (and implement) lessons learned as they are learned; do not wait until a project postmortem when project staff member’s memories are fading and they are moving (have moved) on to their next project.

### 2.2 Test Planning Problems

The following testing problems are related to test planning:

- **PLN-1** No Separate Test Plan
- **PLN-2** Incomplete Test Planning
- **PLN-3** Unclear Test Responsibilities
- **PLN-4** One-Size-Fits-All Test Planning
- **PLN-5** Inadequate Test Resources Planned

#### 2.2.1 PLN-1 No Separate Test Plan

**Description:** There is no separate testing-specific plan.

**Symptoms:** There is no 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). The test planning parts of these other documents are faded and they are moving (have moved) on to their next project.
Common Testing Problems: Pitfalls to Prevent and Mitigate
Checklists of Symptoms, Consequences, and Recommendations

<table>
<thead>
<tr>
<th>are not written by testers. Management and developers do not understand the complexity of testing.</th>
</tr>
</thead>
</table>

**Potential Consequences:** Testing will not have been adequately planned, and the test plans will not be adequately documented. It will be difficult or impossible to evaluate the planned testing process. Testing may be inefficiently and ineffectively performed. An unnecessary number of defects may make it through testing and into the deployed system.

<table>
<thead>
<tr>
<th>Recommendations: Ensure that there is a separate Test and Evaluation Master Plan (TEMP) or Software Test Plan (STP). Do not be satisfied with incomplete high-level overviews of testing in System Engineering Master Plans (SEMPs) and Software Development Plans (SDPs). Customer representatives must ensure that test planning documents are included in the contract as deliverable work products. Ensure that the delivery of test planning documents is noted on the project master schedule (e.g., as part of major milestones).</th>
</tr>
</thead>
</table>

### 2.2.2 PLN-2 Incomplete Test Planning

**Description:** The test planning documents are incomplete.

<table>
<thead>
<tr>
<th>Symptons: The test planning documents are incomplete, missing some or all of the:</th>
</tr>
</thead>
</table>

- test objectives
- testing responsibilities (who does what types of testing on what [types of] components)
- test levels (e.g., unit, subsystem integration, system integration, system, and system of systems)
- test types (e.g., the testing of quality requirements\(^{21}\), abnormal behavior, error/fault/failure tolerance, time- and date-specific functionality, and non-operational modes of operation such as system start-up\(^{22}\), degraded mode, training, and system shutdown)
- 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 prioritization (e.g., addition completeness and formality of tests of mission-, safety-, and security-critical subsystems/software)
- test case selection criteria (e.g., single normal test case vs. boundary value

---

\(^{21}\) This includes but is not limited to the testing of availability, capacity, interoperability, performance, reliability, safety, security, and usability requirements.

\(^{22}\) This includes combinations such as the testing of system start-up when hardware/software components fail.

© 2012 by Carnegie Mellon University
### Potential Consequences: Testers and stakeholders in testing may not understand the true objective of testing (i.e., to find defects so that they can be fixed). Some levels and types of tests may fail to be performed, allowing certain types of residual defects to remain in the system. Some testing may be ad hoc and therefore inefficient and ineffectual. Mission-, safety-, and security-critical software may not be sufficiently tested to the appropriate level of rigor.

Certain types of test cases may be ignored, resulting in related residual defects in the tested system. Test completion criteria may be based more on schedule deadlines than on the required degree of freedom from defects.

### Recommendations: Verify during inspections/reviews that all test planning documents are complete and that they include test objectives, testing responsibilities, test levels, test types, testing methods and techniques, test prioritization, and test completion criteria.

---

#### 2.2.3 PLN-3 Unclear Testing Responsibilities

**Description:** It is unclear who performs what testing.

**Symptoms:** The test planning documents do not adequately address testing responsibilities in terms of which organizations and people will perform which types of testing on what [types of] components.

**Potential Consequences:** Certain tests may not be performed, while other tests may be performed redundantly by multiple organizations or people. Incomplete testing may enable some defects to make it through testing and into the deployed system. Redundant testing will waste test resources and may cause testing deadlines to slip.

**Recommendations:** Obtain organizational agreement as to the testing responsibilities. Clearly and completely document the responsibilities for testing in the test plans as well as the charters of the teams who will be performing the tests. Managers should clearly communicate these responsibilities to the relevant organizations and people.
### PLN-4  One-Size-Fits-All Test Planning

<table>
<thead>
<tr>
<th>Description</th>
<th>All testing is to be performed to the same level of rigor, regardless of its criticality.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symptoms</td>
<td>The test planning documents contain only generic boilerplate rather than appropriate system-specific information. Mission-, safety-, and security-critical software are not required to be tested more completely and rigorously than other less-critical software.</td>
</tr>
<tr>
<td>Potential Consequences</td>
<td>Mission-, safety-, and security-critical software may not be adequately tested. Some defects will not be found, and an unnecessary number of these defects may make it through testing and into the deployed system. The system or software may not provide some mission-critical functionality. The system may not be adequately safe or secure.</td>
</tr>
<tr>
<td>Recommendations</td>
<td>Ensure that the test planning documents contain appropriate system-specific information and are not limited to generic boilerplate documents. Ensure that mission-, safety-, and security-critical software are required to be tested more completely and rigorously than other less-critical software.</td>
</tr>
</tbody>
</table>

### PLN-5  Inadequate Test Resources Planned

<table>
<thead>
<tr>
<th>Description</th>
<th>Test plans (and management) allocate an inadequate amount of resources to testing.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symptoms</td>
<td>The test planning documents and schedules fail to provide for adequate test resources such as:</td>
</tr>
<tr>
<td></td>
<td>• test time in schedule with inadequate schedule reserves</td>
</tr>
<tr>
<td></td>
<td>• trained and experienced testers and reviewers</td>
</tr>
<tr>
<td></td>
<td>• funding</td>
</tr>
<tr>
<td></td>
<td>• test tools and environments (e.g., integration test beds)</td>
</tr>
<tr>
<td>Potential Consequences</td>
<td>Adequate test resources will likely not be provided to perform sufficient testing within schedule and budget limitations. An unnecessary number of defects may make it through testing and into the deployed system.</td>
</tr>
<tr>
<td>Recommendations</td>
<td>Begin test planning at project inception (e.g., at contract award). Ensure that the test planning documents, schedules, and project work breakdown structure (WBS) provide for adequate levels of these test resources.</td>
</tr>
</tbody>
</table>
2.3 Requirements-Related Problems

Many requirements lack the characteristics of good requirements such as being complete, consistent, correct, feasible, mandatory, testable and unambiguous. Such poor quality requirements decrease the testability of systems and software. Given poor requirements, testers are forced to rely on structural testing such as path testing.\(^{23}\)

The following testing problems are related to poor requirements:

- REQ-1 Ambiguous Requirements
- REQ-2 Missing Requirements
- REQ-3 Incomplete Requirements
- REQ-4 Incorrect Requirements
- REQ-5 Unstable Requirements
- REQ-6 Poorly Derived Requirements
- REQ-7 Verification Methods Not Specified

### 2.3.1 REQ-1 Ambiguous Requirements

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Testing is problematic due to ambiguous requirements.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms:</strong></td>
<td>Some of the requirements are ambiguous due to the use of:</td>
</tr>
<tr>
<td></td>
<td>• inherently ambiguous words</td>
</tr>
<tr>
<td></td>
<td>• undefined technical jargon (e.g., application domain terminology as well as the difference between such contractual words as “shall”, “should”, “may”, “recommended”, and “optional”) and acronyms</td>
</tr>
<tr>
<td></td>
<td>• required quantities without associated units of measure</td>
</tr>
<tr>
<td></td>
<td>• unclear synonyms</td>
</tr>
<tr>
<td><strong>Potential Consequences:</strong></td>
<td>Testers may misinterpret the requirements, leading to incorrect blackbox testing. Specifically, ambiguous requirements will often give rise to incorrect test inputs and incorrect expected outputs (i.e., the test oracle is incorrect). Testers may have to spend a sizable amount of time meeting with requirements engineers, customer/user representatives, and</td>
</tr>
</tbody>
</table>

\(^{23}\) At least, this will help to get the system to where it will run without crashing and thereby provide a stable system that can be modified when the customer finally determines what the true requirements are.
subject matter experts in order to sufficiently clarify the ambiguities so that testing can proceed.

**Recommendations**: Promote testability by ensuring that requirements are clear and unambiguous. Ensure that one or more testers review the requirements documents and each requirement for verifiability (testability) before it is approved for use. Encourage testers to request clarification for all ambiguous requirements, and encourage that the requirements be updated based on the clarification given. Verify that the requirements do not include words that are inherently ambiguous, undefined technical terms and acronyms, and quantities without associated units of measure. Ensure that the requirements only use technical or subject matter terminology that is defined in the project glossary and do not use synonyms.

<table>
<thead>
<tr>
<th>2.3.2</th>
<th>REQ-2</th>
<th>Missing Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>: Testing is problematic due to missing requirements.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Symptoms</strong>: Some of the requirements are missing:</td>
<td></td>
<td><strong>Implemented</strong></td>
</tr>
<tr>
<td>• Use case analysis primarily addressed normal (sunny day) paths as opposed to fault tolerant and failure (rainy day) paths.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Requirements for abnormal behavior (e.g., error, fault, and failure detection and reaction) are missing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Quality requirements (e.g., availability, interoperability, maintainability, performance, portability, reliability, robustness, safety, security, and usability) are missing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Data requirements are missing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Potential Consequences</strong>: The missing requirements are not tested, thereby causing the testing to be incomplete (e.g., missing test cases). Testing will not determine that the system or software is missing some of the necessary behavior and characteristics. Customer representatives and developers will have a false sense of security that the system will function properly on delivery and deployment. Testers may have to spend a sizable amount of time meeting with requirements engineers, customer/user representatives in order to clarify missing requirements the existence of which was implied during testing.</td>
<td></td>
<td><strong>Observed</strong></td>
</tr>
<tr>
<td><strong>Recommendations</strong>: Promote testability by ensuring that use case analysis adequately addresses error, fault, and failure (i.e., rainy day) tolerant paths as well as normal (sunny day) paths. Ensure that the requirements repository</td>
<td></td>
<td><strong>Implemented</strong></td>
</tr>
</tbody>
</table>
includes a sufficient amount of the quality and data requirements. Ensure that one or more requirements stakeholders (e.g., customer representatives, user representatives, subject matter experts) review the requirements documents and requirements repository contents for missing requirements before they are accepted and approved for use.

### 2.3.3 REQ-3 Incomplete Requirements

**Description:** Testing is problematic due to incomplete requirements.

**Symptoms:** Requirements are incomplete. The requirements lack:

- Preconditions and trigger events
- Quantitative thresholds
- Postconditions

**Consequences:** Testing will be incomplete or may be incorrect. There may be false negative test results. Some defects associated with incomplete requirements will not be found, and an unnecessary number of these defects may make it through testing and into the deployed system.

**Recommendations:** Ensure that the requirements are complete in order to promote complete testing. Ensure that one or more requirements stakeholders review the requirements documents and requirements repository contents for incomplete requirements before they are accepted and approved for use.

### 2.3.4 REQ-4 Incorrect Requirements

**Description:** Some of the requirements are incorrect.

**Symptoms:** Requirements are determined to be incorrect (invalid) after the associated tests have been developed and run. Testing results include many false positive and false negative results.

**Potential Consequences:** The tests associated with incorrect requirements must be modified or replaced and then rerun, potentially from scratch.

**Recommendations:** Ensure that the requirements are sufficiently validated by requirements stakeholders (e.g., customer representatives, user representatives, subject matter experts) before they are accepted, approved for use, and large numbers of associated test cases are development based on them.
2.3.5  **REQ-5  Unstable Requirements**

<table>
<thead>
<tr>
<th><strong>Description:</strong> Testing is problematic due to requirements volatility.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms:</strong> The requirements are continually changing: new requirements are being added and existing requirements are being modified and deleted. The requirements selected for implementation are not frozen, especially during a short duration increment (e.g., Scrum sprint) when using an incremental, iterative, and parallel – agile – development cycle.</td>
<td><strong>Observed</strong></td>
</tr>
<tr>
<td><strong>Potential Consequences:</strong> Test cases (test inputs, preconditions, and expected test outputs) and automated regression tests are being obsoleted because of requirements changes. The system/software is delivered late.</td>
<td><strong>Observed</strong></td>
</tr>
<tr>
<td><strong>Recommendations:</strong> Promote testability by ensuring that requirements are reasonably stable so that test cases (test inputs, preconditions, and expected test outputs) and automated regression tests are not constantly being obsoleted because of requirements changes.</td>
<td><strong>Implemented</strong></td>
</tr>
</tbody>
</table>

2.3.6  **REQ-6 Poorly Derived Requirements**

<table>
<thead>
<tr>
<th><strong>Description:</strong> Testing is problematic due to poorly derived requirements.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms:</strong> Derived requirements merely restate their parent requirement. Newly allocated requirements are not at the proper level of abstraction.</td>
<td><strong>Observed</strong></td>
</tr>
<tr>
<td><strong>Potential Consequences:</strong> It will be difficult to produce tests at the correct level of abstraction. Testing at the unit- and subsystem-level for these derived requirements may be incomplete. Some of the associated lower-level defects may not be detected until system testing.</td>
<td><strong>Observed</strong></td>
</tr>
<tr>
<td><strong>Recommendations:</strong> Promote testability by reviewing the derived and allocated requirements to ensure that they are at the proper level of abstraction and exhibit all of the standard characteristics of good requirements (e.g., completeness, consistency, correctness, feasible, a lack of ambiguity, a lack of unnecessary architecture or design constraints, and verifiable).</td>
<td><strong>Implemented</strong></td>
</tr>
</tbody>
</table>

2.3.7  **REQ-7 Verification Methods Not Specified**

| **Description:** The methods intended to verify individual requirements are not specified in the |  |

---

24 This testing problem is similar to but more general than the preceding problem: Incorrect Requirements because fixing incorrect requirements is one potential reason that the requirements may be volatile. Other reasons may be engineering missing requirements and changing stakeholder needs.
requirements specification or repository.

**Symptoms:** The requirements specifications do not specify how individual requirements should be verified (e.g., analysis, demonstration, inspection, simulation, testing). The requirements repository does not include verification method(s) as requirements metadata.

**Potential Consequences:** Testers and testing stakeholders may incorrectly assume that all requirements must be verified via testing, even though other verification methods may be adequate, be more appropriate, require less effort, and be faster.

**Recommendations:** Ensure that each requirement (or set of similar requirements) has one or more appropriate verification methods assigned to it/them. Check the appropriateness of these verification methods during requirements inspections, walk-throughs, and reviews. Ensure that actual verification methods used are consistent with the specified requirements verification methods, updating the requirements specifications and repositories when necessary.

### 2.4 Unit Testing Problems

The following testing problems are related to unit testing:

- **UNT-1 Unstable Design**
- **UNT-2 Inadequate Design Detail**
- **UNT-3 Poor Fidelity of Test Environment**

#### 2.4.1 UNT-1 Unstable Design

**Description:** Unit testing is problematic due to design volatility.

**Symptoms:** Design changes (e.g., refactoring and new capabilities) cause the test cases to be constantly updated and test hooks to be lost.

**Potential Consequences:** Unit tests will be unstable, requiring numerous changes and unit-level regression testing. Unit testing will take an unnecessarily long time to perform.

**Recommendations:** Promote testability by ensuring that the design is reasonably stable so that test cases do not need to be constantly updated and

---

25 Note that because unit testing is typically the responsibility of the developers instead of professional testers, the general problem of inadequate testing expertise, experience, and training often applies.

26 This is especially true with agile development cycles with many short-duration increments and with projects where abnormal behavior is postponed until late increments.
test hooks are not lost due to refactoring and new capabilities.\textsuperscript{27}

### 2.4.2 UNT-2 Inadequate Design Detail

<table>
<thead>
<tr>
<th>Description</th>
<th>Unit testing is problematic due to an inadequate level of design detail.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symptoms</td>
<td>There is insufficient design detail to drive the testing.</td>
</tr>
<tr>
<td>Potential Consequences</td>
<td>Unit testing (especially regression testing during maintenance by someone other than the original developer) will be difficult to perform and repeat. Unit testing will take an unnecessarily long time to perform. Unit-level defects may not be found.</td>
</tr>
<tr>
<td>Recommendations</td>
<td>Ensure that the designers/programmers provide sufficient, well-documented design details to drive the unit testing.</td>
</tr>
</tbody>
</table>

### 2.4.3 UNT-3 Poor Fidelity of Test Environment

<table>
<thead>
<tr>
<th>Description</th>
<th>Unit testing is problematic due to the test environment having poor fidelity related to the operational system/software.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symptoms</td>
<td>Unit testing is being performed using a:</td>
</tr>
<tr>
<td></td>
<td>• different environment (e.g., a different [or different version of the] compiler, class library, operating system, middleware, or database) than that used on the delivered software</td>
</tr>
<tr>
<td></td>
<td>• software test environment with poor hardware simulation</td>
</tr>
<tr>
<td>Potential Consequences</td>
<td>Unit testing will experience many false positives. Unit testing will be difficult to perform and repeat. Unit testing will take an unnecessarily long time to perform. Unit-level defects may not be found.</td>
</tr>
<tr>
<td>Recommendations</td>
<td>Ensure adequate fidelity of the test environment so that unit testing does not experience many false positives due to using a:</td>
</tr>
<tr>
<td></td>
<td>• different compiler [version] than the delivered code</td>
</tr>
<tr>
<td></td>
<td>• software test environment with poor hardware simulation</td>
</tr>
</tbody>
</table>

### 2.5 Integration Testing Problems

The following testing problems are related to poor requirements:

- INT-1 Defect Localization
- INT-2 Insufficient Test Environments
- INT-3 Unavailable Components

\textsuperscript{27} This is especially important with agile development cycles with many short-duration increments and with projects where abnormal behavior is postponed until late increments.
## INT-1 Defect Localization

**Description:** Localizing defects is problematic due to encapsulation caused by integration.

| Symptoms: | It is difficult to determine the location of the defect: in the new or updated operational software under test, in 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. | Observed |

| Potential Consequences: | Defect localization will take an unnecessarily large amount of time and effort to perform. Errors in defect localization may cause the wrong fix (e.g., the wrong changes or changes to the wrong software) to be made. | Observed |

| Recommendations: | Ensure that the architecture and design adequately support testability (i.e., provide the testers with sufficient visibility and control to develop and execute adequate tests). Ensure that the design and implementation (with exception handling, BIT, and test hooks), the tests, and the test tools make it relatively easy to determine the location of defects. Where appropriate, incorporate a test mode that logs information about errors, faults, and failures to support defect identification and localization. | Implemented |

## INT-2 Insufficient Test Environments

**Description:** There is an insufficient number of test environments.

| Symptoms: | 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 environments. | Observed |

| Potential Consequences: | It will be difficult to optimally schedule the allocation of test teams to test environments, resulting in scheduling conflicts. Too much time will be wasted reconfiguring the test environments for the next team’s use. Testing may not be completed on schedule. | Observed |

| Recommendations: | Ensure that there are a sufficient number of test environments of each type so that it is practical to optimally schedule the | Implemented |
allocation of test teams to test environments. For example, this could include in order of increasing fidelity:

- Software only on basic general-purpose platform such as a PC
- Software only on appropriate computational environment (e.g., correct processors, busses, operating system, middleware, databases)
- Software with prototype hardware (e.g., sensors and actuators).
- Software with early/previous version of the hardware.
- Software with actual hardware.

If necessary, port the software to another available environment (with lower fidelity) for initial testing.

### 2.5.3 INT-3 Unavailable Components

**Description:** Integration testing is problematic due to unavailability of needed system, software, or test environment components.

<table>
<thead>
<tr>
<th>Symptoms: The operational software, simulation software, test hardware, and actual hardware components (e.g., sensors, actuators, and network devices) are not available for integration into the test environments prior to scheduled integration testing.</th>
<th>Observed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Potential Consequences:</strong> Testing will not be able to begin until the missing components are available and have been integrated into the test environments. Testing may not be completed on schedule.</td>
<td>Observed</td>
</tr>
<tr>
<td><strong>Recommendations:</strong> Ensure that the operational software, simulation software, test hardware, and actual hardware components are available for integration into the test environments prior to scheduled integration testing. The project budget and schedule need to include the effort and time required to develop and install the simulation software and test hardware. If necessary:</td>
<td>Implemented</td>
</tr>
<tr>
<td>- Obtain components with lower fidelity for initial testing.</td>
<td></td>
</tr>
<tr>
<td>- Develop simulators for the missing components.</td>
<td></td>
</tr>
</tbody>
</table>

### 2.5.4 INT-4 Inadequate Test Bed Quality

**Description:** The quality of the test environments is inadequate.

<table>
<thead>
<tr>
<th>Symptoms: The test environments contain excessive numbers of defects.</th>
<th>Observed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Potential Consequences:</strong> There may be numerous false positive test results. It will be more difficult to determine whether test failures are due to the</td>
<td>Observed</td>
</tr>
</tbody>
</table>
system/software under test or the test environments. Testing will take a needlessly long time to perform. The system may be delivered late and with an unnecessarily large number of residual defects.

**Recommendations**: Ensure that the quality of the test environment is as good as the system/software under test, especially when testing mission-, safety-, or security-critical software. Ensure that the test environments are of sufficient quality (e.g., via good development practices, adequate testing, and careful tool selection).

<table>
<thead>
<tr>
<th>2.5.5 INT-5 Inadequate Self-Test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>: Testing is problematic due to a lack of system- or software-internal self-tests.</td>
</tr>
<tr>
<td><strong>Symptoms</strong>: The operational subsystem or software does not contain sufficient test hooks, built-in-test (BIT), or prognostics and health management (PHM) software.</td>
</tr>
<tr>
<td><strong>Potential Consequences</strong>: Failures will be difficult to cause, reproduce, and localize. Testing will take an unnecessarily long time to perform, potentially exceeding the test schedule.</td>
</tr>
<tr>
<td><strong>Recommendations</strong>: Ensure that the operational software or subsystem contains sufficient test hooks, built-in-test (BIT), or prognostics and health management (PHM) software so that failures are reasonably easy to cause, reproduce, and localize.</td>
</tr>
</tbody>
</table>

**2.6 Specialty Engineering Testing Problems**

The following testing problems are related to the specialty engineering testing of quality characteristics and attributes:

- **SPC-1** Inadequate Capacity Testing
- **SPC-2** Inadequate Concurrency Testing
- **SPC-3** Inadequate Performance Testing
- **SPC-4** Inadequate Reliability Testing
- **SPC-5** Inadequate Robustness Testing
- **SPC-6** Inadequate Safety Testing
- **SPC-7** Inadequate Security Testing
- **SPC-8** Inadequate Usability Testing

---

28 Note that analogous testing problems could also exist for other quality characteristics.
### 2.6.1 SPC-1 Inadequate Capacity Testing

**Description:** An inadequate level of capacity testing is being performed.

<table>
<thead>
<tr>
<th>Symptoms</th>
<th>Observed</th>
</tr>
</thead>
<tbody>
<tr>
<td>All capacity requirements are not identified and specified. There is little or no testing to determine if performance degrades gracefully as capacity limits are approached, reached, and exceeded. There is little or no verification of adequate capacity-related computational resources (e.g., memory utilization or processor utilization).</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Potential Consequences</th>
<th>Observed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing is less likely to detect some defects causing violations of capacity requirements. The system may not meet its capacity requirements.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommendations</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensure that all capacity requirements are properly specified. Specify how capacity requirements will be verified (and tested) in a project test planning document. Ensure that all capacity requirements are adequately tested to determine performance as capacity limits are approached, reached, and exceeded. Use tools that simulate large numbers of simultaneous users.</td>
<td></td>
</tr>
</tbody>
</table>

### 2.6.2 SPC-2 Inadequate Concurrency Testing

**Description:** An inadequate level of concurrency testing is being performed.

<table>
<thead>
<tr>
<th>Symptoms</th>
<th>Observed</th>
</tr>
</thead>
<tbody>
<tr>
<td>The testing of concurrent behavior is not addressed in any test planning or process description documents. There is little or no testing being performed explicitly to identify the defects that cause the common types of concurrency faults and failures: deadlock, livelock, starvation, priority inversion, race conditions, inconsistent views of shared memory, and unintentional infinite loops. Any concurrency testing that is being performed is based on a random rather than systematic approach to test case identification (e.g., based on the interleaving of threads). Any concurrency testing is being performed manually. Such concurrency faults and failures are only being identified when they happen to occur while unrelated testing is being performed. Concurrency faults and failures occur infrequently, intermittently, and are difficult to reproduce. Testing is performed using a low fidelity environment with regard to concurrency: threads rather than processes, single rather than multiple processors, the use of deterministic rather than probabilistic drivers and stubs, and the use of hardware simulation rather than actual hardware.</td>
<td></td>
</tr>
</tbody>
</table>

© 2012 by Carnegie Mellon University
### Potential Consequences
Any concurrency testing is both ineffectual and labor intensive. Many defects that can cause concurrency faults and failures are not being found and fixed until final system testing, operational testing, or system operation when they are much more difficult to reproduce, localize, and understand.

### Recommendations
Provide testers with training in concurrency defects, faults, and failures. Use concurrency testing techniques that enable the systematic selection of a reasonable number of test cases (e.g., ways of interleaving the threads) from the impractically large number of potential test cases. For testing of threads sharing a single processor, use a concurrency testing tool that provides control over thread creation and scheduling. When such tools are unavailable or inadequate, develop scripts that (1) automate the testing of deadlock and race conditions, (2) enable the reproducibility of test inputs, and (3) record test results for analysis. To the extent possible, do not rely on (1) merely throwing large numbers of simultaneous inputs/requests\(^{29}\) or (2) performing manual testing.

---

#### 2.6.3 SPC-3 Inadequate Performance Testing

### Description
An inadequate level of performance testing is being performed.

### Symptoms
Performance requirements are not specified for all of its component quality attributes: event schedulability, jitter, latency, response time, and throughput. There is little or no performance testing or testing to determine if performance degrades gracefully. There is little or no verification of adequate performance-related computational resources (e.g., I/O bandwidth, bus bandwidth, or processor utilization). Performance testing is performed using a low fidelity environment.

### Potential Consequences
Testing is less likely to detect some performance defects. Specify how performance requirements will be verified (and tested) in a project test planning document. The system may not meet its performance requirements.

### Recommendations
Ensure that all performance requirements are properly identified and specified. Ensure that all performance requirements are adequately tested.

---

\(^{29}\) Such tests may redundantly test the same interleaving of threads while leaving many interleavings untested. Unexpected determinism may even result in the exact same interleaving being performed over and over again.
2.6.4  **SPC-4 Inadequate Reliability Testing**

**Description:** An inadequate level of reliability testing is being performed.  

<table>
<thead>
<tr>
<th>Symptoms:</th>
<th>There is little or no long duration reliability testing (a.k.a., stability testing) under operational profiles.</th>
<th>Observed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential Consequences:</td>
<td>Testing is less likely to detect some defects causing violations of reliability requirements (and data to enable the estimation of system reliability will not be collected). The system may not meet its reliability requirements.</td>
<td>Observed</td>
</tr>
<tr>
<td>Recommendations:</td>
<td>Ensure that all reliability requirements are properly identified and specified. Specify how reliability requirements will be verified (or tested) in a project test planning document. To the degree that testing as opposed to analysis is practical as a verification method, ensure that all reliability requirements undergo sufficient long duration reliability testing under operational profiles to estimate the system’s reliability.</td>
<td>Implemented</td>
</tr>
</tbody>
</table>

2.6.5  **SPC-5 Inadequate Robustness Testing**

**Description:** An inadequate level of robustness testing is being performed.

| Symptoms: | Robustness testing is not based on robustness analysis such as abnormal (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). There is little or no robustness testing:  
  
  - **Error Tolerance Testing**, the goal of which is to show that system does not detect or react properly to input errors (a subtype of which is Fuzz Testing)  
  - **Fault Tolerance Testing**, the goal of which is to show that system does not detect or react properly to system faults (bad internal states)  
  - **Failure Tolerance Testing**, the goal of which is to show that system does not detect or react properly to system failures (to meet requirements)  
  - **Environmental Tolerance Testing**, the goal of which is to show that system does not detect or react properly to dangerous environmental conditions | Observed |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential Consequences:</td>
<td>Testing is less likely to detect some defects causing violations of robustness requirements. Some error, fault, failure, and...</td>
<td>Observed</td>
</tr>
</tbody>
</table>

---

30 Note that reliability (load and stability) testing are nominal tests in the sense that they are executed within the performance envelop of the System Under Test (SUT). Capacity (stress) testing, where you test for graceful degradation, is outside the scope of performance testing.
environmental tolerance defects will not be found. The system may exhibit inadequate robustness.

**Recommendations**: Ensure that all robustness requirements are properly identified and specified. Specify how robustness requirements will be verified (and tested) in a project test planning document. Ensure that there is sufficient testing of all robustness requirements to verify adequate error, fault, failure, and environmental tolerance. Ensure that this testing is based on proper robustness analysis such as abnormal (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).

**2.6.6 SPC-6 Inadequate Safety Testing**

**Description**: An inadequate level of safety testing is being performed.

**Symptoms**: There is little or no:
- testing based on safety analysis (e.g., abuse/mishap cases, ETA, or FTA)
- testing of safeguards (e.g., interlocks)
- testing of fail-safe behavior
- safety-specific testing:
  - *Vulnerability Testing*, the goal of which is to expose a system vulnerability (i.e., defect or weakness)\(^{31}\)
  - *Hazard Testing*, the goal of which is to make the system cause a hazard to come into existence
  - *Mishap Testing*, the goal of which is to make the system cause an accident or near miss

**Potential Consequences**: Testing is less likely to detect some defects causing violations of safety requirements. Some defects with safety ramifications will not be found. The system may exhibit inadequate safety.

**Recommendations**: Ensure that all safety-related requirements are properly identified and specified. Specify how safety requirements will be verified (and tested) in a project test planning document. Ensure that there is sufficient blackbox testing of all safety requirements and sufficient graybox/whitebox testing of safeguards (e.g., interlocks) and fail-safe behavior. Ensure that this testing is based on adequate safety analysis (e.g., abuse/mishap cases) as well as the safety architecture and design.

---

\(^{31}\) Note that the term vulnerability (meaning a weakness in the system/software) applies to both safety and security. Vulnerabilities can be exploited by an abuser [either unintentional (safety) or intentional (security)] and contribute to the occurrence of an abuse [either mishap (safety) or misuse (security)].
2.6.7 **SPC-7 Inadequate Security Testing**

**Description:** An inadequate level of security testing is being performed.

**Symptoms:** There is little or no:
- 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)
- testing of fail-secure behavior
- security-specific testing:
  - *Penetration Testing*, the goal of which is to penetrate the system’s defenses
  - *Fuzz Testing*, the goal of which is to cause the system to fail due to random input
  - *Vulnerability Testing*, the goal of which is to expose a system vulnerability (i.e., defect or weakness)

**Potential Consequences:** Testing is less likely to detect some defects causing violations of security requirements. Some vulnerabilities and other defects having security ramifications will not be found. The system may exhibit inadequate security.

**Recommendations:** Ensure that all security-related requirements are properly identified and specified. Specify how security requirements will be verified (and tested) in a project test planning document. Ensure that all system actors are documented (e.g., profiled). Ensure that there is sufficient security testing (e.g., penetration testing) of all security requirements, security features, security controls, and fail-secure behavior. Ensure that this testing is based on adequate security analysis (e.g., attack trees, abuse/misuse cases). Note: use static vulnerability analysis tools to identify commonly occurring security vulnerabilities.

---

2.6.8 **SPC-8 Inadequate Usability Testing**

**Description:** An inadequate level of usability testing is being performed.

**Symptoms:** There is little or no explicit usability testing of the system’s or software’s human interfaces.

**Potential Consequences:** Testing is less likely to detect some defects causing

---

32 **Warning:** although a bad idea, security requirements are sometimes specified in a security document rather than in the requirements specification/repository. Similarly, security testing is sometimes documented in security rather than testing documents.
violations of usability requirements. Some defects with usability ramifications will not be found. The system may exhibit inadequate usability.

**Recommendations:** Ensure that all usability requirements are properly identified and specified. Specify how usability requirements will be verified (and tested) in a project test planning document. Ensure that there is sufficient usability testing of the human interfaces. Include usability testing for all relevant usability attributes such as accessibility, attractiveness (also known as engagability, preference, and stickiness), credibility (also known as trustworthiness), differentiation, ease of entry, ease of location, ease of remembering, effectiveness, effort minimization, error minimization, learnability, navigability, retrievability, suitability (also known as appropriateness), understandability, and user satisfaction.

### 2.7 System Testing Problems

The very nature of system testing often ensures that these problems cannot be eliminated. At best, the recommended solutions can only mitigate them.

The following testing problems are related to system testing:

- **SYS-1** Testing Robustness is Difficult
- **SYS-2** Testing Code Coverage is Difficult
- **SYS-3** Lack of Test Hooks

#### 2.7.1 SYS-1 Testing Robustness Requirements is Difficult

**Description:** The testing of robustness requirements (specifying error, fault, and failure tolerance)\(^{33}\) is difficult.

**Symptoms:** 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.

**Potential Consequences:** The system or software is less testable because it is less controllable (e.g., causing local faults). Less robustness testing will be done and the delivered system will contain an unnecessarily large number of defects that lessen error, fault, and failure tolerance.

**Recommendations:** Ensure that robustness requirements are specified and implemented.

\(^{33}\)An error is bad input (from a human, another system, or hardware). A fault is an encapsulated (information hiding) incorrect state or incorrect stored data. A failure is an externally visible incorrect response (e.g., output data or control) that typically is a violation of some requirement. An error may or may not result in a fault depending on whether it is stored and there is error tolerance. A fault may or may not cause a failure depending on whether it is executed and there is fault tolerance.
associated architecture/design decisions are documented. Ensure adequate test tool support or that sufficient robustness including error, fault, and failure logging is incorporated into the system to enable adequate testing for tolerance (e.g., by causing encapsulated errors and faults, and observing the resulting robustness). Where appropriate, incorporate test hooks, built-in test (BIT), fault logging (possibly triggered by exception handling, a prognostics and health management (PHM) function or subsystem, or some other way to overcome information hiding in order to verify test case preconditions and post-conditions.

<table>
<thead>
<tr>
<th>2.7.2 SYS-2 Lack of Test Hooks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> System testing is difficult because temporary test hooks have been removed.</td>
</tr>
<tr>
<td><strong>Symptoms:</strong> Internal test hooks and testing software has been removed prior to system testing (e.g., for security or performance reasons).</td>
</tr>
<tr>
<td><strong>Potential Consequences:</strong> It will be difficult to test locally implemented requirements. Such requirements will not be verified at the system level because of decreased testability due to low controllability and observability.</td>
</tr>
<tr>
<td><strong>Recommendations:</strong> Ensure that unit and integration testing have adequately tested locally implemented and encapsulated requirements that are difficult to verify during system testing. Use a test/logging system mode (if one exists).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2.7.3 SYS-3 Testing Code Coverage is Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> Ensuring that tests provide adequate code coverage is difficult.</td>
</tr>
<tr>
<td><strong>Symptoms:</strong> It is difficult for tests of the integrated system to demonstrate code coverage.</td>
</tr>
<tr>
<td><strong>Consequences:</strong> Adequate code coverage as mandated for mission-, safety-, and security-critical software will not be verified. The system will not receive its safety and security accreditation and certification until code coverage is verified.</td>
</tr>
<tr>
<td><strong>Recommendations:</strong> Ensure that unit and integration testing (including regression testing) have demonstrated sufficient code coverage so that code</td>
</tr>
</tbody>
</table>

---

34 Code coverage is typically very important for software with safety or security ramifications. When software is categorized by safety or security significance, the mandatory rigor of testing (including the completeness of coverage) increases as the safety and security risk increases (e.g., from function coverage through statement coverage, decision or branch coverage, and condition coverage to path coverage).
coverage need not be demonstrated at the system level. Use software test tools or probes to measure and report code coverage.

### 2.8 System of Systems (SoS) Testing Problems

Note that system of systems means the integration of separately developed, funded, and scheduled systems having independent governance. This is not referring to a system developed by a prime contractor or integrated by a system integrator consisting of subsystems developed by subcontractors or vendors.

The following testing problems are related to system of systems testing:

- **SoS-1 Inadequate SoS Planning**
- **SoS-2 Poor or Missing SoS Requirements**
- **SoS-3 Unclear SoS Testing Responsibilities**
- **SoS-4 Inadequate Funding for SoS Testing**
- **SoS-5 SoS Testing not Properly Scheduled**
- **SoS-6 Inadequate Test Support from Individual Systems**
- **SoS-7 Inadequate Defect Tracking Across Projects**
- **SoS-8 Finger-Pointing**

#### 2.8.1 SoS-1 Inadequate SoS Planning

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>An inadequate amount of system of systems planning is being performed.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms:</strong></td>
<td>Little or no planning has occurred for testing above the individual system level. The SoS activities have not been determined, planned for, and documented.</td>
</tr>
<tr>
<td><strong>Potential Consequences:</strong></td>
<td>There are no clear test responsibilities, objectives, methods and techniques, and completion/acceptance criteria at the system of systems level. It is unclear who is to do what. Adequate resources (funding, staffing, and schedule) are unlikely to be made available for SoS testing. SoS testing is unlikely to be adequate. There are likely to be numerous system to system interface defects causing the failure of end-to-end mission threads.</td>
</tr>
<tr>
<td><strong>Recommendations:</strong></td>
<td>Create a SoS-level test plan in order to ensure that adequate SoS test planning has occurred above the individual system level. Evaluate the SoS test plan. Ensure that there are clear test completion/acceptance criteria at the SoS level.</td>
</tr>
</tbody>
</table>

© 2012 by Carnegie Mellon University
## 2.8.2 SoS-2 Poor or Missing SoS Requirements

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>Many system of systems requirements are either missing or of poor quality.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms</strong></td>
<td>Little or no requirements exist above the system level. Those SoS requirements that do exist do not exhibit all of the characteristics of good requirements.</td>
</tr>
<tr>
<td><strong>Potential Consequences</strong></td>
<td>Requirements-based SoS testing will be difficult to perform because there are no officially-approved SoS requirements to verify. It will be hard to develop test cases and to determine the corresponding expected test outputs. It is likely that system to system interface defects will cause the failure of end-to-end mission threads.</td>
</tr>
<tr>
<td><strong>Recommendations</strong></td>
<td>Ensure that there are sufficient officially approved SoS requirements to drive requirements-based SoS testing.</td>
</tr>
</tbody>
</table>

**Observed**

## 2.8.3 SoS-3 Unclear SoS Testing Responsibilities

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>The responsibilities for performing end-to-end system of systems testing are unclear.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms</strong></td>
<td>No project is explicitly tasked with testing end-to-end SoS behavior.</td>
</tr>
<tr>
<td><strong>Potential Consequences</strong></td>
<td>No project will have planned to provide the resources (e.g., staffing, budget, schedule) needed to perform SoS testing. Adequate SoS testing is unlikely to be performed, and the SoS will be unlikely to meet its schedule for deployment of new/updated capabilities.</td>
</tr>
<tr>
<td><strong>Recommendations</strong></td>
<td>Ensure that responsibilities for testing the end-to-end SoS behavior are clearly assigned to some organization and project.</td>
</tr>
</tbody>
</table>

**Implemented**

## 2.8.4 SoS-4 Inadequate Funding for SoS Testing

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>The funding for system of systems (SoS) testing is not adequate for the performance of sufficient testing.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms</strong></td>
<td>Little or no funding has been provided to perform end-to-end SoS testing. None of the system-level projects have been funded to perform end-to-end SoS testing.</td>
</tr>
<tr>
<td><strong>Potential Consequences</strong></td>
<td>Little or no end-to-end SoS testing will be performed. It is likely that residual system to system interface defects will cause the failure of end-to-end mission threads.</td>
</tr>
</tbody>
</table>

**Observed**
### Recommendations

Ensure that adequate funding for testing the end-to-end SoS behavior is clearly supplied to the responsible organization and project.

#### 2.8.5 SoS-5 SoS Testing not Properly Scheduled

<table>
<thead>
<tr>
<th>Description: System of system testing is not properly scheduled.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms:</strong> SoS testing is not in the individual system’s integrated master schedules, and there is no SoS-level master schedule. SoS testing must be fit into the uncoordinated schedules of the individual systems comprising the SoS.</td>
</tr>
<tr>
<td><strong>Potential Consequences:</strong> SoS testing that is not scheduled will be unlikely to be performed. If performed, it is likely that the testing will be rushed, incomplete, and inadequate with more mistakes than typical. The operational SoS is likely to contain more SoS integration defects and end-to-end mission thread defects than is appropriate.</td>
</tr>
<tr>
<td><strong>Recommendations:</strong> Ensure that SoS testing is on the SoS master schedule. Ensure that SoS testing is also on the individual system’s integrated master schedules so that support for SoS testing can be planned. Ensure that SoS testing is coordinated with the schedules of the individual systems.</td>
</tr>
</tbody>
</table>

#### 2.8.6 SoS-6 Inadequate Test Support from Individual Systems

<table>
<thead>
<tr>
<th>Description: Test support from individual system development/maintenance projects is inadequate to perform system of system testing.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symptoms:</strong> All available system-level test resources (e.g., staffing, funding, and test environments) are already committed to system testing.</td>
</tr>
<tr>
<td><strong>Potential Consequences:</strong> It will be difficult or impossible to obtain the necessary test resources from individual projects to support SoS testing.</td>
</tr>
<tr>
<td><strong>Recommendations:</strong> Ensure that the individual projects provide adequate test resources (e.g., people and test beds) to support SoS testing. Ensure that these resources are not already committed elsewhere.</td>
</tr>
</tbody>
</table>

#### 2.8.7 SoS-7 Inadequate Defect Tracking Across Projects

<table>
<thead>
<tr>
<th>Description: Defect tracking across individual system development or maintenance projects is inadequate to support system of systems testing.</th>
</tr>
</thead>
</table>
| **Symptoms:** There is little or no coordination of defect tracking and associated regression testing across multiple projects. Different projects collect different
types and amounts of information concerning defects identified during testing.

**Potential Consequences**: It will be unnecessarily difficult to synchronize system- and SoS-level activities. Defect localization and allocation of defects to individual or sets of systems will be difficult to perform.

**Recommendations**: Develop a consensus concerning how to address defect reporting and tracking across the systems making up the SoS. Document this consensus in all relevant plans (SoS and individual systems). Verify that defect tracking and associated regression testing across the individual projects of the systems making up the SoS are adequately coordinated.

### 2.8.8 SoS-8 Finger-Pointing

**Description**: Different system development/maintenance projects assign the responsibility for defects and fixing them to other projects.

**Symptoms**: There is a significant amount of finger pointing across project boundaries regarding whether something is a defect (or feature) or where defects lie (i.e., in which systems and in which project’s testing).

**Potential Consequences**: Time and effort will be wasted in the allocation of defects to individual or sets of systems. Defects will take longer to be fixed, and these fixes will take longer to be verified.

**Recommendations**: Ensure representatives of the individual systems are on the SoS change control board (CCB). Work to develop a SoS mindset among the members of the SoS CCB.

### 2.9 Regression Testing Problems

The following problems are specific to the performance of regression testing including testing during maintenance:

- **REG-1** Insufficient Regression Test Automation
- **REG-2** Regression Tests Not Rerun
- **REG-3** Inadequate Scope of Regression Testing
- **REG-4** Only Low-Level Regression Tests
- **REG-5** Disagreement over Maintenance Test Resources

#### 2.9.1 REG-1 Insufficient Regression Test Automation

**Description**: Some or all of the regression tests are not automated.
### Common Testing Problems: Pitfalls to Prevent and Mitigate
Checklists of Symptoms, Consequences, and Recommendations

#### Symptoms: All or some of the regression tests are not automated. At least some of the regression tests are manual.

#### Potential Consequences:
Regression testing is not sufficient, especially when an agile (iterative, incremental, and parallel) development cycle, which causes numerous, short-duration increments that must be retested. Manual regression testing will take so much time and effort that it is not done. If performed, the testing will likely be rushed, incomplete, and inadequate with excessive mistakes. A higher than normal number of defects will not be found and therefore remain in the system.

#### Recommendations:
Automate as much of the regression/maintenance testing as is practical. Ensure that adequate resources (staffing, budget, and schedule) are planned and available for automating and maintaining the tests. Ensure that manual test results are integrated into the overall test results database so that test reporting and monitoring are seamless.

---

### 2.9.2 REG-2 Regression Tests Not Rerun

**Description:** Some or all of the regression tests are not rerun after changes are made.

**Symptoms:** Regression testing is not being done because:

- 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.

**Potential Consequences:** Defects introduced while changing existing previously tested subsystems/software will remain in the operational system because they will not be found during regression testing.

**Recommendations:** Ensure that sufficient regression testing is being performed by providing sufficient time and staffing to perform it as well as ensuring adequate automation. Resist efforts to skip regression testing because of the “minor scope of most changes” because defects often unexpectedly propagate faults and failures beyond their local scope.

---

### 2.9.3 REG-3 Inadequate Scope of Regression Testing

**Description:** The scope of regression testing is not sufficiently broad.

**Symptoms:** Only the changed subsystem or software is retested because of the mistaken belief that the change will only have local effects and thus can’t

---

© 2012 by Carnegie Mellon University
affect the rest of the system.

**Potential Consequences**: Defects introduced while changing existing previously tested subsystems/software will remain in the operational system because they will not be found during regression testing.

**Recommendations**: Resist efforts to limit the scope of regression testing because of the “change can’t affect the rest of the system”; defects have a way of causing propagating faults and failures. Automate as many of the regression tests as is practical so that it will be possible to rerun them.

<table>
<thead>
<tr>
<th>2.9.4 REG-4 Only Low-Level Regression Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>: Only low-level regression tests are rerun.</td>
</tr>
<tr>
<td><strong>Symptoms</strong>: Only unit tests and some integration tests are rerun. System and/or the SoS tests are not rerun.</td>
</tr>
<tr>
<td><strong>Potential Consequences</strong>: Integration defects introduced while changing existing previously tested subsystems/software will remain in the operational system because they will not be found during regression testing.</td>
</tr>
<tr>
<td><strong>Recommendations</strong>: Ensure that all relevant levels of regression testing (e.g., unit, integration, system, specialty, and SoS) are rerun when changes are made. Automate as many of these regression tests so that it will be practical to rerun them.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2.9.5 REG-5 Disagreement over Maintenance Test Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong>: The development and maintenance projects disagree over who is responsible for providing the test resources (e.g., staffing, budget, test work products) during maintenance.</td>
</tr>
<tr>
<td><strong>Symptoms</strong>: There is disagreement as to whether the resources for maintenance testing should be provided by the development or maintenance projects.</td>
</tr>
<tr>
<td><strong>Potential Consequences</strong>: Insufficient resources will be made available to adequately support maintenance testing. Testing will be delayed while the source of these resources is negotiated.</td>
</tr>
<tr>
<td><strong>Recommendations</strong>: Ensure that the funding for maintenance testing is clearly assigned to either the development or sustainment project. Include funding responsibilities in the transition plan (if there is one).</td>
</tr>
</tbody>
</table>
3 Conclusion

3.1 Testing Problems

There are many testing problems that 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 problems, most projects will suffer several of them. Similarly, while exhibiting these testing problems does not guarantee failure, these problems are definitely risks that need to be managed.

The 65 common problems involving how testing is performed have been grouped into the following categories:

1. **General Testing Problems** – 23 problems of a general nature not restricted to a specific type or scope of testing.
2. **Test Planning Problems** – 5 problems that occur due to inadequate test planning
3. **Requirements-related Problems** – 7 testing problems due to poor requirements
4. **Unit Testing Problems** – 3 problems specific to unit testing
5. **Integration Testing Problems** – 5 problems specific to integration testing
6. **Specialty Engineering Testing Problems** – 8 problems specific to the specialty-engineering testing of quality requirements
7. **System Testing Problems** – 3 problems specific to the testing of complete systems
8. **System of System Testing Problems** – 8 problems specific to the testing of systems of systems
9. **Regression Testing Problems** – 5 problems specific to the performance of regression testing including testing during maintenance

3.2 Common Consequences

While different testing problems have different proximate negative consequences, they all tend to contribute to the following overall ultimate results:

- The testing effort is less effective and efficient.
- Some defects are discovered later than they should be, when they are more difficult to localize and fix.
- The testers must work unsustainably long hours causing them to become exhausted and therefore make excessive numbers of mistakes.
- The software-reliant system or software application is delivered late and over budget because of extra unplanned time and effort spent finding and fixing defects late during development.
- In spite of this extra budget and schedule, the software-reliant system or software application is still delivered and placed into operation with more residual defects than either expected or necessary.
3.3 Common Solutions

In addition to the individual problem-specific recommendations provided in the preceding checklists, the following general solutions are applicable to most of the common testing problems:

- **Prevention Solutions** – The following solutions can prevent the problems from occurring in the first place:
  - *Formally require the solutions* – Customer representatives formally require the solutions to the testing problems in the appropriate documentation such as the Request for Proposals and Contract.
  - *Mandate the solutions* – Managers, chief engineers (development team leaders), or chief testers (test team leaders) explicitly mandate the solutions to the testing problems in the appropriate documentation such as the System Engineering Management Plan (SEMP), System/Software Development Plan (SDP), Test Plan(s), and/or Test Strategy.
  - *Provide training* – Chief testers or trainers provide appropriate amounts and levels of test training to relevant personnel (such as to acquisition staff, management, testers, and quality assurance) that covers the potential testing problems and how to prevent, detect, and react to them.
  - *Management support* – Managers explicitly state (and provide) their support for testing and the need to avoid the commonly occurring test problems.

- **Detection Solutions** – The following solutions enable existing problems to be identified and diagnosed:
  - *Evaluate documentation* – Review, inspect, or walk through the test-related documentation (e.g., Test Plan and test sections of development plans).
  - *Oversight* – Provide acquirer, management, quality assurance, and peer oversight of the testing process as it is performed.
  - *Metrics* – Collect, analyze, and report relevant test metrics to stakeholders (e.g., acquirers, managers, technical leads or chief engineers, and chief testers).

- **Reaction Solutions** – The following solutions help to solve existing problems once they are detected:
  - *Reject test documentation* – Customer representatives, managers, and chief engineers refuse to accept test-related documentation until identified problems are solved.
  - *Fail the test* – Customer representatives, managers, and chief engineers refuse to accept the system/subsystem/software under test until identified problems (e.g., in test environments, test procedures, or test cases) are solved. Rerun the tests after prioritizing and fixing the associated defects.
— **Provide training** – Chief testers or trainers provide appropriate amounts and levels of remedial test training to relevant personnel (such as to acquisition staff, management, testers, and quality assurance) that covers the *observed* testing problems and how to prevent, detect, and react to them.

— **Update process** – Chief engineers, chief testers, and/or process engineers update the test process documentation to minimize the likelihood of reoccurrence of the observed testing problems.

— **Formally raise risk** – Raise existing test problems as formal risks and inform both project management and the customer representative.

### 3.4 Potential Future Work

The contents of this document were not the results of a formal academic study. Rather, they were derived largely from the author’s 30+ years of experience assessing and taking part in numerous projects as well as numerous discussions with testing subject matter experts. This paper has been provided for review to over 120 academics and professionals and incorporates comments and recommendations received from the individuals listed in the following acknowledgements.

As such, the current qualitative document leaves several important quantitative questions unanswered:

- **Frequency.** What is the probability distribution of these problems? Which problems occur most often? Which problems tend to cluster together?

- **Impact.** Which problems have the largest negative consequences? What are the probability distributions of harm caused by each problem?

- **Risk.** Based on the above frequencies and impacts, which of these problems cause the greatest risks? Given these risks, how should one prioritize the identification and resolution of these problems?

- **Distribution.** Do different problems tend to occur with different probabilities in different application domains such as commercial vs. governmental vs. military, web vs. IT vs. embedded systems, etc.)?

Provided sufficient funding, it is the author’s intent to turn this document into an industry survey and to perform a formal study to answer these questions.

### 4 Acknowledgements

I would like to thank the following reviewers who have provided comments and recommendations regarding this document:

1. Vince Alcalde, Independent Consultant, Australia
2. Alexandru Cosma, ISDC, Romania
3. Lee Eldridge, Independent Consultant, Australia
4. Eliazar Elisha, University of Liverpool, UK
5. Sam Harbaugh, Integrated Software Inc., USA
6. M. E. Hom, Compass360 Consulting, USA
7. Thanh Cong Huynh, LogiGear, Vietnam
8. Ronald Kohl, SEI, USA
9. Wido Kunde, Baker Hughes, Germany
10. Philippe Lebacq, Toyota Europe, Belgium
11. Ken Niddefer, SEI, USA
12. Anne Nieberding, Private Consultant, USA
13. Mahesh Palan, Calypso Technology, USA
14. Dan Pautler, Elekta, USA
15. Mark Powel, Attwater Consulting, USA
16. Sudip Saha, Navigators Software, India
17. Alejandro Salado, Kayser – Threde GmbH, Germany
18. Matt Sheranko, Knowledge Code, USA
19. Oleg Spozito, Independent Consultant, Canada
20. Barry Stanly, Private Consultant, USA
21. Lou Wheatcraft, Requirements Experts, USA
22. Thomas Zalewski, Texas State Government, USA