Little Book of Testing

Volume II

Implementation Techniques

June 1998

THE AIRLIE SOFTWARE COUNCIL

This publication was prepared for the

Software Program Managers Network
4600 North Fairfax Drive, Suite 302
Arlington, VA 22203

[illegible]

Norm Brown
Director, Software Program Managers Network

This work was created by Computers & Concepts Associates in the performance of Space and Naval Warfare Systems Command (SPAWAR) Contract Number N00039-94-C-0153 for the operation of the Software Program Managers Network (SPMN).

This guidebook is one of a series of guidebooks published by the Software Program Managers Network (SPMN). Our purpose is to identify best management and technical practices for software development and maintenance from the commercial software sector, and to convey these practices to busy program managers and practitioners. Our goal is to improve the bottom-line drivers of software development and maintenance—cost, productivity, schedule, quality, predictability, and user satisfaction.

The Airlie Software Council was convened by a Department of the Navy contractor in 1994 as a focus group of software industry gurus supporting the SPMN and its challenge of improving software across the many large-scale, software-intensive systems within the Army, Navy, Marine Corps, and Air Force. Council members have identified principal best practices that are essential to managing large-scale software development and maintenance projects. The Council, which meets quarterly in Airlie, Virginia, is comprised of some 20 of the nation's leading software experts. These little guidebooks are written, reviewed, generally approved, and, if needed, updated by Council members. Your suggestions regarding this guidebook, or others that you think should exist, would be much appreciated.

THE PURPOSE OF THIS LITTLE BOOK

The Software Program Managers Network (SPMN) provides resources and best practices for software managers. The SPMN offers practical, cost-effective solutions that have proved successful on projects in government and industry around the world.

This second volume in the testing series answers the question:"How can I test software so that the end product performs better, is reliable, and allows me, as a manager, to control cost and schedule?" Implementing the basic testing concepts and ten testing rules contained in this booklet will significantly improve your testing process and result in better products.

It is not easy to change the way systems and software are tested. Change requires discipline, patience, and a firm commitment to follow a predetermined plan. Management must understand that so-called silver bullets during testing often increase risk, take longer, and result in a higher-cost and lower-quality product. Often what appears to be the longest testing process is actually the shortest, since products reach predetermined levels of design stability before being demonstrated.

Managers can implement the concepts and rules in this guidebook to help them follow a low-risk, effective test process. An evaluation of how you test software against these guidelines will help you identify project risks and areas needing improvement. By using the principles described in this guide, you can structure a test program that is consistent with basic cost, schedule, quality, and user satisfaction requirements.

Norm Brown

Norm Brown
Executive Director

INTRODUCTION

Testing by software execution is the end of the software development process. When projects are discovered to be behind schedule, a frequent worst-practice way out is to take shortcuts with test. This is a worst practice because effective test by software execution is essential to delivering quality software products that satisfy users' requirements, needs, and expectations. When testing is done poorly, defects that should have been found in test are found during operation, with the result that maintenance costs are excessive and the user-customer is dissatisfied. When a security or safety defect is first found in operation, the consequences can be much more severe than excessive cost and user dissatisfaction.

Although metrics show that the defect removal efficiency of formal, structured peer reviews is typically much higher than that of execution test, there are certain types of errors that software-execution test is most effective in finding. Full-path-coverage unit test with a range of path-initiating variable values will find errors that slip through code inspections. Tests of the integrated system against operational scenarios and with high stress loads find errors that are missed with formal, structured peer reviews that focus on small components of the system. This book describes testing steps, activities, and controls that we have observed on projects.

[illegible]

Michael W. Evans
President
Integrated Computer Engineering, Inc.

Frank McGrath

Frank J. McGrath, Ph.D.
Technical Director
Software Program Managers Network

BASIC TESTING CONCEPTS

This section presents concepts that are basic to understanding the testing process. From the start it is critical that managers, developers, and users understand the difference between debugging and testing.

- *Debugging* is the process of isolating and identifying the cause of a software problem, and then modifying the software to correct the problem. This guidebook does not address debugging principles.
- *Testing* is the process of finding defects in relation to a set of predetermined criteria or specifications. **The purpose of testing is to prove a system, software, or software configuration *doesn't* work, not that it *does*.**

There are two forms of testing: *white box testing* and *black box testing*. An ideal test environment alternates white box and black box test activities, first stabilizing the design, then demonstrating that it performs the required functionality in a reliable manner consistent with performance, user, and operational constraints.

White box testing is conducted on code components which may be software units, computer software components (CSCs), or computer software configuration items (CSCIs). These tests exercise the internal structure of a code component, and include:

- Execution of each statement in a code component at least once
- Execution of each conditional branch in the code component
- Execution of paths with boundary and out-of-bounds input values
- Verification of the integrity of internal interfaces
- Verification of architecture integrity across a range of conditions
- Verification of database design and structure

White box tests verify that the software design is valid and that it was built according to the specified design. White box testing traces to configuration management (CM)-controlled design and internal interface specifications. These specifications have been identified as an integral part of the configuration control process.

Black box testing is conducted on integrated, functional components whose design integrity has been verified through completion of traceable white box tests. As with white box testing, these components include software units, CSCs, or CSCIs.

Black box testing traces to requirements focusing on system externals. It validates that the software meets requirements without regard to the paths of execution

BASIC TESTING CONCEPTS (cont.)

taken to meet each requirement. It is the type of test conducted on software that is an integration of code units.

The *black box testing process* includes:

- Validation of functional integrity in relation to external stimuli
- Validation of all external interfaces (including human) across a range of nominal and anomalous conditions
- Validation of the ability of the system, software, or hardware to recover from or minimize the effect of unexpected or anomalous external or environmental conditions
- Validation of the system's ability to address out-of-bound input, error recovery, communication, and stress conditions

Black box tests validate that an integrated software configuration satisfies the requirements contained in a CM-controlled requirement or external interface specification.

Ideally, each black box test should be preceded by a white box test that stabilizes the design. You never want to try to isolate a software or system problem by executing a test case designed to demonstrate system or software externals.

BASIC TESTING CONCEPTS (cont.)

Figure 1 illustrates the *testing process*. Testing is a continuum of verification and validation activities conducted continuously from the point where a product is defined until it is finally validated in an installed system configuration.

Testing need not be performed on a computer. Projects that regularly apply structured inspections not only test products against criteria at the point where they are produced, but also reduce the *rework* rate from 44 percent to 8 percent. Use of inspections as a form of testing can reduce the cost of correction up to 100 times. A complete test program consists of the following nine discrete *testing levels*, each feeding the next:

Test Levels	Test Activity	Test Type	Documentation Basis for Testing	Test Responsibility	Test Focus
Level 0	Structured Inspections	Non-Computer-Based Test	Various	Inspection Team	Various
Level 1	Computer Software Unit Testing	White Box Testing	SDF	Developer	Software Unit Design
Level 2	CSCI Integration Testing	White Box Testing	SWDD	Independent Test	CSCI Design/Architecture
Level 3	CSCI Qualification Testing	Black Box Testing	SRS	Independent Test	CSCI Requirements
Level 4	CSCI/HWCI Integration Testing	White Box Testing	SSDD	Independent Test	System Design/Architecture
Level 5	System Testing	Black Box Testing	SSS	Independent Test	System Requirements
Level 6	DT&E Testing	Black Box Testing	User Manuals	Acquirer Test Group	User Manual Compliance
Level 7	OT&E Testing	Black Box Testing	ORD or User Requirements Document	Operational Test	Operational Requirements
Level 8	Site Testing	Black Box Testing	Transition Plan (Site Configuration)	Site Installation Team	Site Requirements

Key to Terms:

DT&E–Development Test and Evaluation

HWCI–Hardware Configuration Item

ORD-Operational Requirements Document

OT&E–Operational Test and Evaluation

SDF–Software Design File

SRS–Software Requirements Specifications

SSDD–System Subsystem Design Document

SSS–System Segment Specification

SWDD–Software Design Document

Figure 1. Test Levels

BASIC TESTING CONCEPTS (cont.)

Level 0—These tests consist of a set of structured inspections tied to each product placed under configuration management. The purpose of Level 0 tests is to remove defects at the point where they occur, and before they affect any other product.

Level 1—These white box tests qualify the code against standards and unit design specification. Level 1 tests trace to the *Software Design File (SDF)* and are usually executed using test harnesses or drivers. This is the only test level that focuses on code.

Level 2—These white box tests integrate qualified CSCs into an executable CSCI configuration. Level 2 tests trace to the *Software Design Document (SWDD)*. The focus of these tests is the inter-CSC interfaces.

Level 3—These black box tests execute integrated CSCIs to assure that requirements of the *Software Requirements Specification (SRS)* have been implemented and that the CSCI executes in an acceptable manner. The results of Level 3 tests are reviewed and approved by the acquirer of the product.

Level 4—These white box tests trace to the *System Subsystem Design Document (SSDD)*. Level 4 tests integrate qualified CSCIs into an executable system configuration by interfacing independent CSCIs and then integrating the executable software configuration with the target hardware.

Level 5—These black box tests qualify an executable system configuration to assure that the requirements of the system have been met and that the basic concept of the system has been satisfied. Level 5 tests trace to the *System Segment Specification (SSS)*. This test level usually results in acceptance or at least approval of the system for customer-based testing.

Level 6—These black box tests, planned and conducted by the acquirer, trace to the user and operator manuals and external interface specifications. Level 6 tests integrate the qualified system into the operational environment.

Level 7—These independent black box tests trace to operational requirements and specifications. Level 7 tests are conducted by an agent of the user to assure that critical operational, safety, security, and other environmental requirements have been satisfied and the system is ready for deployment.

Level 8—These black box tests are conducted by the installation team to assure the system works correctly when installed and performs correctly when connected to live site interfaces. Level 8 tests trace to installation manuals and use diagnostic hardware and software.

TESTING RISKS AND SOLUTIONS

The process of testing must not be compromised by the occurrence of preventable problems. In most cases, the effects of problems can be minimized by the early execution of a preplanned mitigation strategy. *Risk management* requires a management belief that risk is a problem that has not yet occurred. Through risk management, the full effect of a problem can be minimized or avoided. This level of risk management requires that common testing risks be identified early, a mitigation strategy identified, and resources reserved to implement the strategy if the need arises. Figure 2 examines impact and planning factors for risk areas commonly encountered during testing.

RISK AREA	IMPACT	PLANNING
1. Schedule difficulty due to extended development and code time	Insufficient time to test software resulting in schedule slip	Phase software testing into build testing
2. An inability to develop executable test cases from procedures	Test scripts not adequate representation of software execution characteristics	Phase development personnel into test planning activities
3. Insufficient interface definition and inability to duplicate exact test conditions	Software systems with known problems delivered to customer	Hard code test scripts readable through simulator
4. Ambiguous user specifications or system operator instructions	Procedures do not match system support requirements	Complete high-level test plan early and fill in details as system or software specifications become available
5. Schedule delay between system releases when new software configurations stabilize	Significant schedule impacts due to poor initial performance of software releases	Schedule informal software burn-in period with each release
6. Delays associated with correction of problems	Repeated slips due to problem correction delays	Plan for software configuration management and overlap system releases
7. Baselines lost during development and corrections lost during test	System releases not controlled or of unknown content	Plan, implement, and integrate strong, effective configuration management
8. Inadequate hardware availability or reliability	Significant hardware-caused delays on the software schedule	Find alternate hardware source of simulation facility at beginning of project
9. a. Insufficient manpower available	Excessive overtime required	Provide for manpower pool, sharing resources between test, development, and support
b. Loss of key individual	Project delay, poor productivity, inadequate support	Find and train key man backups early in project
10. Diversion of key resource	Scheduling impacts and project delays	Back up all planned resources

Figure 2. Predictable Resource and Test & Integration Problems

TESTING QUESTIONS THAT MANAGERS SHOULD BE ABLE TO ANSWER

TESTING QUESTIONS THAT MANAGERS SHOULD BE ABLE TO ANSWER:

1. Have I defined, and do I understand, my overall strategy for software testing?
2. Are the test planning requirements clearly defined, consistent, assigned for development, and supportable by the staff responsible for their implementation?
3. Are the test methods and techniques defined, are they consistent with the strategy, and can they be implemented in the software environment?
4. Is each test activity traceable to a controlled set of requirements and/or design data?
5. Are the configuration management control, and quality disciplines adequate to support the test strategies, methods, and techniques, and are they really in place?
6. Do I know when to quit?

TEN COMMANDMENTS OF TESTING

THE TEN COMMANDMENTS OF TESTING

1. Black box tests that validate requirements must trace to approved specifications and be executed by an independent organization.
2. Test levels must be integrated into a consistent structure.
3. Don't skip levels to save time or resources: test less at each level.
4. All test products configurations, tools, data, and so forth, need CM during tests.
5. Always test to procedures.
6. Change must be managed.
7. Testing must be scheduled, monitored, and controlled.
8. All test cases have to trace to something that's under CM.
9. You can't run out of resources.
10. Always specify and test to criteria.

RULE 1: Tests Must Be Planned, Scheduled, and Controlled if the Process Is to Be Effective.

- Test planning is an essential management function that defines the strategies and methods to be used to integrate, test, and validate the software product.
- Test planning defines how the process is managed, monitored, and controlled.
- Planning of the software test environment is the bottom of a complex planning tree that structures and controls the flow of testing. The planning tree shows the products move from developer, through the software organization, to the system organizations, and finally to the acquirer and the operational test organizations.
- Test planning is "preparing for the future," while Test Plan implementation is "making sure we have what we want when we get there."
- The test planning process must be consistent from one level to the next. The Test Plan must be consistent with the CM Plan, which must be consistent with the standards, and so on.
- The control discipline must be a complete representation of the Test Plan as adapted to the specific needs of the project to which it is applied.

The Test Planning Tree

- *Program Plan* defines the program requirements, obligations, strategies, constraints, and delivery requirements and commitment. It identifies test requirements to be completed prior to delivery.
- *Test and Evaluation Management Plan* usually traces to user specifications. It sets the entire testing strategy and practices.
- *System Engineering Management Plan (SEMP)* identifies engineering plans and methods, engineering standards, management processes, and systems for integration and test requirements.
- *System Test Plan* defines the system's testing plan, strategies, controls, and testing processes. Test case requirements are discussed, and criteria for success or failure are identified.
- *Software Development Plan* describes what the software project must accomplish, how these software process and product standards must be met, how the process must be managed and controlled, and what the criteria are for overall integration, test, and completion.

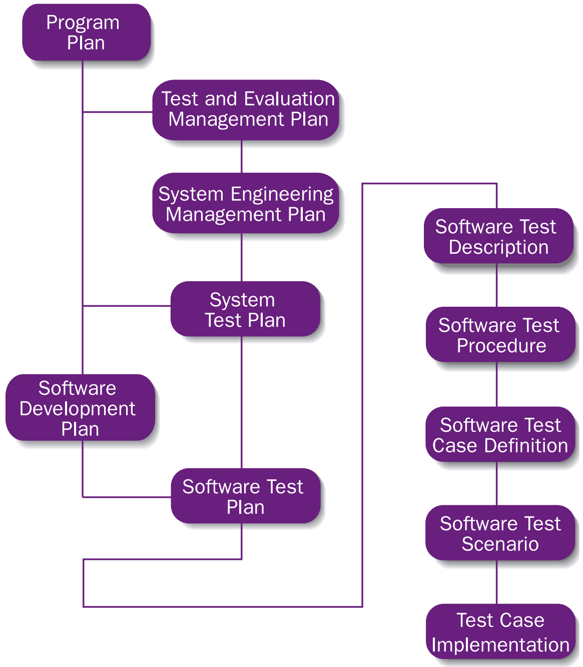
Program Plan
Test and Evaluation Management Plan
System Engineering Management Plan
System Test Plan
Software Development Plan
Software Test Plan
Software Test Description
Software Test Procedure
Software Test Case Definition
Software Test Scenario
Test Case Implementation

Figure 3. The Test Planning Tree

- *Software Test Plan* defines software integration and test plans, strategies, software test controls and processes, test case requirements, and test case completion criteria.
- *Software Test Description* details the requirements for individual test cases. This description serves as the test case design requirement.
- *Software Test Procedure* takes each test case and provides a step-by-step definition of test execution requirements traceable to a requirement specification.
- *Software Test Case Definition* details the test case design and specifies criteria for successful and unsuccessful execution of a test step.
- *Software Test Scenario* identifies specific test data, data sources, interface address or location, critical test timing, expected response, and test step to take in response to test condition.
- *Test Case Implementation* is the actual test case ready for execution.

These plans may be rolled up into fewer individual documents, or expanded based on project size, number of organizations participating, size of risk and contract, and customer and technical requirements of the test program.

RULE 1 (cont.)

The test structure as documented in these plans must be consistent with the overall strategy used—grand design, incremental development, or evolutionary acquisition.

- *Test planning* is the joint responsibility of the organization receiving the capability—either the higher life cycle or the user organization that will ultimately apply the product—and the engineering organization that will build or modify the product.
- The test planning process starts with a definition by the ultimate user of the system of the level of risk that is acceptable when the product is delivered. The integration of the testing structure disciplines helps assure the acceptability of the product when it reaches the operational *testing levels.*

Key guidelines for planning a test environment are:

- Improve probability that the delivered products will meet the documented and perceived requirements, operational needs, and expectations of the user of the system.
- Increase schedule and budget predictability by minimizing rework and redundant effort.
- Maintain traceability and operational integrity of engineering products as they are produced and changed, and maintain the relationship between the formal documentation and the underlying engineering information.
- Assure that software is not debugged during operational testing because of test shortcuts or inadequacies at lower test levels.

RULE 2

RULE 2: Execution Test Begins with White Box Test of Code Units and Progresses through a Series of Black Box Tests of a Progressively Integrated System in Accordance with a Predefined Integration Test Build Plan.

- Software test flow must be documented in plans and procedures.
- Traceability between test cases at each level must be maintained, allowing tests at one level to isolate problems identified at another.
- The integrity of the planned white-box, black-box test sequence must not be violated to meet budget or schedule.
- Software cannot be adequately tested if there is no formal, controlled traceability of requirements from the system level down to individual code units and test cases.
- Test levels (see Figure 1) must be defined so that they support partitioning that is consistent with the project's incremental or evolutionary acquisition strategy.
- In the event of a failure at any test level, it must be possible to back up a level to isolate the cause.
- All test levels must trace to CM-controlled engineering information, as well as to baselined documentation that has been evaluated through a structured inspection.
- A test level is not complete until all test cases are run and a predetermined quality target is reached.

RULE 3: All Components of a Test Must Be under CM Control after They Have Been Verified by Structured Peer Reviews.

During test, CM must provide:

- A way to identify the information approved for use in the project, the owners of the information, the way it was approved for CM control, and the most recent latest approved release
- A means to assure that, before a change is made to any item under configuration control, the change has been approved by the CM change approval process, which should include a requirement that all impacts are known before a change is approved
- An accessible and current record of the status of each controlled piece of information that is planned for use by test
- A build of each release, from code units to integration test
- A record of the exact content, status, and version of each item accepted by CM, and the version history of the item
- A means to support testing of operational software versions by exactly reproducing software systems for tests with a known configuration and a controlled set of test results

CM Implementation Rules During Testing

- CM should be an independent, centralized activity, not buried within an engineering or assurance organization.
- CM can never be a bottleneck in the test schedule, because the process will not be followed.
- CM can never be eliminated for any reason, including the need to meet budget or schedule.
- CM change control must receive reports of all suspected and confirmed problems during testing, and all problems must only be fixed in accordance with a change control process managed by CM.
- CM must assure that changes to configuration-controlled items that do not follow the CM process cannot be made to systems under test, including patches to binary code.
- CM must not accept changes made to software until all related documentation and other configuration-controlled items affected by the change have also been updated.
- CM must have control of and responsibility for all test builds from Level 2 and above.

- CM must ensure that all releases of integrated software from CM during test are accompanied by a current, complete, and accurate Software Version Description (SVD).
- CM must verify that all new releases of test tools, including compilers, are regression-tested prior to release for use during testing.
- CM must process every suspected or observed problem, no matter how small.
- CM must ensure that all tests trace to controlled information that has been approved through a binary quality gate.
- CM staff during integration and test must have the technical skills necessary to generate each integration build delivered from CM for integration test.

RULE 4

***RULE 4: Testing Is Designed to Prove a System* Doesn't *Work, Not to Prove It* Does.**

- A large, integrated software system cannot be exhaustively tested to execute each path over the full range of path-initiating conditions. Therefore, testing must consist of an integrated set of test levels, as described in Figure 1. These testing levels maximize the coverage of testing against the cost across all design and requirements.
- Criteria must be used to evaluate all test activities.
- All tests executed that trace to a requirement must be executed in three ways:
  - Using a nominal data load and valid information
  - Using excessive data input rates in a controlled environment
  - Using a preplanned combination of nominal and anomalous data
- The ideal test environment is capable of driving a system to destruction in a controlled fashion. For example, the data aids and data combinations must be varied until the system no longer executes in an acceptable manner. The point at

which system support becomes unacceptable must be identified and documented.

- Tests must be run at all levels to test not only nominal conditions but also all potential test failure conditions documented in controlled specifications.
  - **This is critical even though the test environment may be difficult to establish.**
- No observed testing problem, however small, must ever be ignored or closed without resolution.
- All test case definitions must include success and failure criteria.
- Tests must be made against scenarios (timed sequences of events) designed to cover a broad range of real-world operational possibilities as well as against static system requirements.
- System test must be made on the operational hardware/operating system (OS) configuration.

RULE 5

RULE 5: All Software Must Be Tested Using Frequent Builds of Controlled Software, with Each Build Documented in SVDs and Change Histories Documented in Header Comments for Each Code Unit.

- Build selection and definition must be based on availability of key test resources, software unit build and test schedules, the need to integrate related architectural and data components into a single test configuration, availability of required testing environments and tools, and the requirement to verify related areas of functionality and design.
- Builds must be ordered to minimize the need to develop throw-away, test-driver software.
- Builds must not be driven by a need to demonstrate a capability by a certain date (a form of political build scheduling).
- Build definition must consider technical, schedule, and resource factors prior to the scheduling of a test session.
- Test cases used during frequent builds must be planned (structured in a Test Plan, documented in a procedure), traceable to controlled requirements or design, and executed using

controlled configurations, just as with builds that are not frequent.

- Test support functions, such as CM integration build and change approvals, must be staffed to a sufficient level and must follow processes so that they do not delay frequent integration builds and tests.
- The build planning must be dynamic and capable of being quickly redefined to address testing and product realities. For example, a key piece of software may not be delivered as scheduled; a planned external interface may not be available as planned; a hardware failure may occur in a network server.
- Frequent builds find integration problems early, but they require substantial automation in CM, integration, and regression test.

RULE 6

RULE 6: Test Tools Must Only Be Built or Acquired If They Provide a Capability Not Available Elsewhere, or If They Provide Control, Reproducibility, or Efficiency That Cannot Be Achieved through Operational or Live Test Sources.

- Test tools must be used prior to the point at which the system is considered stable and executing in a manner consistent with specifications.
  – This requirement reflects the need to control test conditions and to provide a controlled external environment in which predictability and reproducibility can be achieved.
- Some test functions are best provided by automated tools and can substantially lower the cost of regression test during maintenance.
  – For example, automating key build inputs reduces operator requirements and the cost of providing them during test sessions.

Classes of Test Tools That Every Program Should Consider

- *Simulation When Test Is Done Outside the Operational Environment*—Drives system from outside using controlled information and environments. For example, it can be used to exchange messages with the system under test, or to drive a client/server application as though there were many simultaneous users on many different client workstations
- *Emulation When Hardware Interfaces Are Not Available*—Software to perform exactly as missing components in system configurations, such as an external interface
- *Instrumentation*—Software tools that automatically insert code in software, and then monitors specific characteristics of the software execution such as test coverage
- *Test Case Generator*—Identifies test cases to test requirements and to achieve white box test coverage
- *Test Data Generator*—Defines test information automatically, based on the test case or scenario definition
- *Test Record and Playback*—Records keystrokes and mouse commands from the human tester, and saves corresponding software output for later playback and comparison with saved output in regression test
- *Test Analysis*—Evaluates test results, reduces data, and generates test reports

Rules for Test Tools

- Test tools must only be acquired and used if they directly trace to a need in the Test Plan and/or procedures, or if they will significantly reduce the cost of regression test during the maintenance phase.
- All tools used and the data they act upon must be under CM control.
- Test tools must not be developed if there is a quality COTS test tool that meets needs.
- Plan test tool development, acquisition, and deployment for completion well in advance of any need for tool use item.
- Test tools that certify critical components must be certified using predefined criteria before being used.
- Always try to simulate all external data inputs, including operator inputs, to reduce test cost and increase test reproducibility.

RULE 6 (cont.)

Questions to Ask When Defining a Testing Environment and Selecting Test Methods or Tools

1. Have the proposed testing techniques and tools been proven in applications of similar size and complexity, with similar environments and characteristics?
2. Does the test environment include the hardware/OS configuration on which the software will be deployed, and will the test tools execute on this operational hardware/OS?
3. Do the proposed test methods and tools give the customer the visibility into software quality that is required by the contract?
4. Is there two-way traceability from system requirements through design, to code units and test cases?
5. Have the organizational standard test processes, methods, and use of test tools been tailored to the characteristics of the project?

RULE 7

RULE 7: When Software Is Developed by Multiple Organizations, One Organization Must Be Responsible for All Integration Test of the Software Following Successful White Box Testing by the Developing Organization.

- *Development tests*—those that test units and CSCs prior to integration—must be the responsibility of the developers.
- When development tests are complete, *inspections* must be conducted to ensure the process was followed; the requirements of the Test Plan were successfully completed; the software is documented adequately; the source code matches the documentation; and the software component is adequate to support subsequent test activities.
- All software builds that support test activities subsequent to development testing must be built under control of CM.
- Testing subsequent to development testing must be executed by an independent testing organization. After development testing ends, any changes to controlled configurations must be evaluated and approved prior to incorporation in the test configuration.

Rules for Organizing Test Activities

- A testing organization must be an integral part of the program and software project plans from the beginning, with staffing and resources allocated to initiate early work.
- While some testing will be done internally by the engineering organizations, the white box and black box test activities documented in Test Plans and procedures must be executed by an independent test organization.
- Test organizations must be motivated to find problems rather than to prove the products under test work.
- Test organizations must have adequate guaranteed budget and schedule to accomplish all test activities without any unplanned shortcuts.
- A tester must never make on-the-fly changes to any test configuration. A tester's role is to find and report, not to fix. During testing, the role of the tester is to execute the test case. All fixes should be made by an organization responsible for maintaining the software, after processing in accordance with the change control process of the program.
- Test organizations may delegate tasks such as test planning, test development, tool development, or tool acquisition to engineering, but responsibility, control, and accountability must reside with the test organization.
- The relationship between testing and other program and software organizations must be clear, agreed to, documented, and enabled through CM.

RULE 8

RULE 8: Software Testing Is Not Finished until the System Has Been Stressed above Its Rated Capacity and Critical Factors—Closure of Safety Hazards, Security Controls, Operational Factors—Have Been Demonstrated.

- Software functional testing and system testing must have a step that demonstrates anticipated realistic, not artificial, system environments. In this step, the system must be loaded to the maximum possible level as constrained by external data services. Also, potential failure conditions must be exercised in stressed environments.
- All safety hazard control features and security control features must be exercised in nominal and stressed environments to ensure that the controls they provide remain when the system is overloaded.
- The test data rates must be increased until the system is no longer operationally valid.
- Test scenarios that are used in stress testing must define nominal system conditions. The specifics of the scenarios must then be varied, increasing in an orderly manner operator positions, database loads, external inputs, external outputs, and any other factors that would affect loading and, indirectly, response time.
- During execution of these tests, the available bandwidth must be flooded with low-priority traffic to ascertain if this traffic precludes high-priority inputs.
- Specific tests must exist to assure the capability of a fully loaded or overloaded system to gracefully degrade without the failure of some hardware component. Specific shutdown options of the system must be explicitly exercised in a stressed environment to assure their viability.
- On certain systems that have potential external input sources, the ability of an unauthorized user gaining access in periods of system stress must be tested. These penetration tests must be consistent with the security policy of the program.
- Satisfaction of predefined quality goals must be the basis for completion of all test activities, including individual test step execution, cases described through a test procedure, test level completion, and completion and satisfaction of a Test Plan. These quality goals must include stress and critical factor components.
- All software that impacts safety, or software that is part of the Trusted Computing Base segment of a secure application, must be tested to assure that

RULE 8
(cont.)

security and hazard controls perform as required. All testing of critical components must be executed in nominal and stressed environments to assure that controls remain in place in periods of system stress.

- Free play testing must only be used when the design has been verified through Level 2 or 4 testing or when requirements have been validated through Level 3 or 6 testing. Free play tests of unstable design or nonvalidated requirements are not always productive.
- Black box tests of requirements (Levels 3, 5, 6, 7, or 8) must be defined, or at least validated, by someone with operational or domain experience so that the tests reflect the actual use requirements of the capability. Traceability between test cases and requirements must be maintained.
- All test procedures and cases must be defined to identify what the software does, what it doesn't do, and what it can't do.
- All critical tests, especially those that relate to critical operational safety or security requirements, must be executed at 50 percent of the rated system capacity, 100 percent of the rated capacity, and 150 percent of the rated capacity.
- Documentation of failures must include sufficient information to evaluate and correct the problem. Knowledge of the stress load the system was under can be used to identify points at which operational risk is excessive due to system loading.

RULE 9: A Robust Testing Program Reduces Program Acquisition Risks.

- System acquisition is becoming increasingly more complex. More comprehensive testing strategies that can identify defects earlier in the development process are required.

ACTIVITIES THAT CAN SIGNIFICANTLY REDUCE PROGRAM RISKS WHEN PROPERLY TIED TO THE TEST PLAN:

- Ensure that the Program Plan addresses the testing strategy for the entire acquisition life cycle. It must consider schedule, cost, and performance objectives.
- Ensure that requirements traceability and the test strategy builds confidence in the program development process during each phase.
- Develop an exhaustive Test Plan that retains requirements traceability from requirements documentation through product delivery. The Test Plan should define a testing strategy from cradle to grave while bolstering credibility to program performance capabilities. This Plan is an essential element of any software development project.
- Establish a Configuration Management Plan. The CM Plan is key to a successful testing strategy. It helps to ensure that all requirements are incorporated, tracked, tested, and verified at each development stage.
- Establish a Quality Assurance Program. The QA Program verifies that requirements and configuration management processes support testing plan objectives, and that system architecture and engineering requirements are being achieved.
- Ensure visibility and monitoring of these independent but highly integrated activities to judiciously reduce programmatic risks.

ISSUES AND CONCERNS THAT MUST BE RAISED BY PROGRAM MANAGERS, RISK OFFICERS, AND TEST PERSONNEL:

- Has a CM Program been established to track documentation and data that is shared across the multilevel organization?
- Have requirements documents been incorporated into the CM Program?
- Has a requirements traceability matrix been developed?
- Has a testing strategy been developed that defines test processes from program conception to system delivery?

RULE 9 (cont.)

- Does the testing strategy incorporate the use of commercial best practices into the Test Plan?
- Does the Test Plan define quality gates for each development step that requires either formal or informal testing?
- Does the Test Plan identify testability criteria for each requirement?
- Are processes in place to monitor and control requirements changes and incorporation of those changes into the Test Plan?
- Are all activities integrated throughout the development process to facilitate continuous feedback of testing results and consideration of those results in determining program status?
- How does the consumption of program reserves during testing factor into program schedule and cost updates? How is the consumption of program reserves during testing recognized?
- Has Quality Assurance verified that all development processes and procedures comply with required activities?

RULE 10

RULE 10: All New Releases Used in a Test Configuration Must Be Regression-Tested Prior to Use.

- Regression testing is the revalidation of previously proven capabilities to assure their continued integrity when other components have been modified.
- When a group of tests is run, records must be kept to show which software has been executed. These records provide the basis for scheduling regression tests to revalidate capabilities prior to continuing new testing.
- Every failure in a regression test must be addressed and resolved prior to the execution of the new test period.
- Regression test environments must duplicate to the maximum extent possible the environment used during the original test execution period.
- Regression tests must be a mix of white box and black box tests to verify known design areas as well as to validate proven capability.
- All regression tests must use components that are controlled by configuration management.

RULE 10 (cont.)

- Regression testing must be a normal part of all testing activities. It must be scheduled, planned, budgeted for, and generally supported by the testing organization. “Let’s just run it to see if it works” is not a valid regression-testing strategy.

SUMMARY

Testing is no longer considered a stand-alone and end-of-the-process evolution to be completed simply as an acquisition milestone. Rather, it has become a highly integral process that complements and supports other program activities while offering a means to significantly reduce programmatic risks. Early defect identification is possible through comprehensive testing and monitoring. Effective solutions and mitigation strategies emerge from proactive program management practices once risks have been identified.

Successful programs have demonstrated that integrated testing activities are key to program success with minimal associated risks.

INDEX

INDEX (cont.)