



For additional information please contact the
Software Program Managers Network

(703) 521-5231 • Fax (703) 521-2603

E-Mail: spmn@aol.com

<http://www.spmn.com>

**LITTLE BOOK
OF
TESTING**

VOLUME I

**OVERVIEW AND
BEST PRACTICES**



APRIL 1998

THE AIRLIE SOFTWARE COUNCIL



This publication was prepared for the

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

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



Norm Brown
Director, Software Program Managers Network

Copyright © 1998 by Computers & Concepts Associates

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 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



As the significance of software in defense systems has increased, so has the demand for effective software testing and the prudent management of software-related risks. The costs of software testing and the time required for it have also grown; consequently, the quality of systems in most organizations is often below standard. Knowledge of the testing practices outlined in this guide can help managers face the challenges of software testing and risk management.

Collected here is wisdom on software testing and evaluation (T&E) gathered from successful software managers and consultants in government and industry around the world. This first volume of the Software Program Managers Network testing series offers strategies for effective testing of large-scale software programs.

With practical, real-world advice on how to manage testing and evaluation programs, this guidebook demonstrates that good testing practices are a key element of all modern software programs, leading toward the development and implementation of significant quality improvements.



Norm Brown
Executive Director

WHAT IS SOFTWARE TESTING?

Testing plays many roles in the development of software.

1. Testing is exercising or simulating a system or program operation.
2. Testing is establishing confidence that a program does what it is supposed to, and doesn't do what it isn't supposed to.
3. Testing is analyzing a program with the intent of finding problems and errors.
4. Testing is measuring system functionality and quality.
5. Testing is evaluating the attributes and capabilities of programs and project work products, and assessing whether they achieve required or acceptable results.
6. Testing includes inspections and structured peer reviews of requirements and design, as well as execution test of code.



Q: So what is testing really?

A: All of the above!

WHY IS IT IMPORTANT?

Five important reasons why testing should be ranked as a top concern, especially at the start of a project:

1. A poor testing program can cause mission failure, can significantly impact operational performance and reliability, and can double or triple field support and maintenance costs.
2. A good testing program is a major project cost. Complex programs can spend more than half their total program effort on T&E activities. To make testing effective, you have to take the time up front to plan and organize it properly.
3. A good testing program will help significantly as you define your early requirements and design work. That help is critical to getting the project started right, and it can have a major influence on overall project success.
4. A good testing program forces you to face and deal with problems as the work is done, and when the cost of rework and fixes is much lower.
5. A good testing program can't totally make up for a poor software project, but it does help prevent many ills and will at least let you know you are in trouble early. Testing is insurance, and many project managers have learned the hard way that most need big policies!

PURPOSES AND BENEFITS OF GOOD TESTING

TESTING OBJECTIVES

- **To help clearly describe system behavior.**
- **To find defects in requirements, design, documentation, and code as early as possible.**

Historically, testing has focused on exercising software to obtain confidence in its readiness for use and to demonstrate that it is working satisfactorily. This leads to an emphasis on the detection and removal of defects. Modern testing continues to emphasize demonstration and detection, but also recognizes that many—maybe even most—major defects are rooted in requirements and design misunderstandings, omissions, and inconsistencies. Early, structured peer reviews are now commonly used to help prevent problems from ever being coded. Demonstration, detection, and prevention have all become important objectives and benefits of good testing.



DEMONSTRATION

1. Gain confidence that systems can be used with acceptable risk.
2. Try out features and functions under unusual conditions and situations.
3. Assure that a work product is completed and ready for use or integration.

DETECTION

1. Discover defects, errors, and system deficiencies.
2. Define system capabilities and limitations.
3. Provide information on the quality of components, systems, and work products.

PREVENTION

1. Clarify system specifications and performance.
2. Provide information that prevents or reduces the likelihood of errors being made.
3. Detect errors earlier in the process.
4. Identify risks and problems and ways to avoid them in the future.

PRINCIPLES OF GOOD TESTING

- **COMPLETE TESTING ISN'T POSSIBLE**
No matter how much you test, it is impossible to achieve total confidence.
The only exhaustive test is one that leaves the tester exhausted!
- **TEST WORK IS CREATIVE AND DIFFICULT**
You must understand and probe what the system is supposed to do.
You must understand and stress the limitations and constraints.
You must understand the domain and application in depth.
- **TESTING IS RISK-BASED**
We can't identify all risks of failure.
Risk assessments indicate how much to test and what to focus on.
- **ANALYSIS, PLANNING, AND DESIGN ARE IMPORTANT**
Test objectives must be identified and understood.
Tests must be planned and designed systematically.
Without a road map, you will get lost.
- **MOTIVATION IS IMPORTANT**
You cannot be effective if you don't care about the job.
You must want to find problems and enjoy trying to break the system.
- **TIME AND RESOURCES ARE IMPORTANT**
You can't be effective if you don't have the time or resources to do the job.
- **TIMING OF TEST PREPARATION MATTERS A LOT**
Early test preparation leads to an understanding of project requirements and design.
Early test preparation uncovers and prevents problems.
Early tests improve the effectiveness of subsequent reviews and inspections.
- **MEASURING AND TRACKING COVERAGE IS ESSENTIAL**
You need to know what requirements, design, and code have and have not been covered.
Complex software is too difficult to cover without systematic measurement.



TOP TEN BEST TESTING PRACTICES

1. MAKE EVALUATION EVERYONE'S JOB—A TEAM RESPONSIBILITY
Insist on making T&E an integral element of all professional work. Cultivate the attitude to take it seriously, and cooperate as a project team to produce well-tested, high-quality work products.

2. ESTABLISH AN EARLY, INTEGRATED MASTER TESTING AND EVALUATION PLAN

Provide and maintain a Master Testing and Evaluation Plan (MTP) covering all anticipated T&E activities and deliverables. Integrate the test plans with the overall project and program plans, and ensure that responsibilities and resources are assigned as early as possible.

3. MAKE PREVENTIVE TESTING PART OF ALL SPECIFICATION WORK
Establish systematic evaluation and preliminary test design as a sizable part of all system engineering and specification work. Design test scenarios early and use them in reviews and inspections to help assure you are building the right features, and test for specification work product completion.

4. USE TESTS AS PROGRESS AND MILESTONE GATES
Require the use of tests to verify that all project deliverables and components are complete, and to demonstrate and track true project progress. Avoid delay in addressing problems by making test completion a gate to be achieved before work product sign-off.

5. INVENTORY TEST OBJECTIVES AND DESIGN FOR TESTABILITY
Develop and maintain a risk-prioritized list of test requirements and objectives: requirements-based, design-based, and code- or implementation-based. Use the inventory to guide test design and development, and to trace or cross-reference what each test covers.

6. TEST EARLY, TEST OFTEN

Use tests as early and as often as possible to provide developer feedback and to get problems found and fixed as they occur. Emphasize testware engineering as a concurrent life cycle process tightly coupled with software design and development.

7. DESIGN AND DEVELOP TESTWARE AS DELIVERABLE COMPONENTS
Design and develop major testware components and procedures with the same discipline and care reserved for software components. This includes planning, analysis, design, review, configuration management, and change control and management reporting.

8. PROVIDE INTEGRATED TEST TOOLING AND INFRASTRUCTURE SUPPORT
Support and enable project team T&E with a database or repository and an integrated test management system for describing, documenting, and tracking tests. Also supply support tools, such as simulators, coverage analyzers, capture playback tools, test drivers, and utilities.

9. MEASURE TEST COSTS, COVERAGE, RESULTS, AND EFFECTIVENESS
Collect information to monitor T&E costs, results, and benefits. Measure test coverage of requirements, design, and code at all levels and for all major builds and test environments. Measure what tests are run on what software versions and what each test level or major activity finds or misses. Use the information to evaluate T&E effectiveness.

10. COACH AND MANAGE THE TEAM

Provide ongoing T&E leadership and management support so that all on the team know what is expected and take testing seriously. Understand your testing activities and process, and make measurement-driven policy and process changes as needed.

BEST TESTING PRACTICE #1

MAKE EVALUATION EVERYONE'S JOB—A TEAM RESPONSIBILITY

Insist on making T&E an integral element of all professional work. Cultivate the attitude and responsibility to take it seriously, and cooperate as a project team to produce well-tested, high-quality work products.

Most of us consider testing and evaluation as an integral part of any work we do. We want to test our own work carefully and thoroughly, and we want our team to deliver the highest-quality results possible. We inherently understand that good testing is everyone's job. We know it is part of the process, and in most instances we even know what needs to be done to achieve it.

However, the reality is that under the pressures of time and the heat of battle, T&E tasks are often the first to be delayed, trimmed, or dropped altogether. Why is this, and what should we be doing to get everyone on the team to meet their testing responsibilities better?

Much of the problem is psychological:

- Testing when we know there are many more problems to find is discouraging.

- Testing when we feel fairly sure everything already works is boring.
- Weak testing lets us pretend we have achieved more progress than we have.
- To developers, testing exposes more problems and means more work or rework.
- To managers, testing looks like overhead that can be cut when in trouble.
- Users and customers willingly trade and accept low quality for more features.

The solution isn't a big mystery. Management has to cultivate the right attitudes and insist that T&E responsibilities be taken seriously by everyone on the team, not just testers. This takes leadership and sometimes means making tough decisions, but effective modern testing is a team activity that requires direction and discipline.

Best Practice #1 is both the easiest to achieve and the hardest to maintain. It is built on the organizational culture and the determination as a project team to collectively produce high-quality work products all the time.

BEST TESTING PRACTICE #2

ESTABLISH AN EARLY INTEGRATED MASTER TESTING AND EVALUATION PLAN

Provide and maintain a Master Testing and Evaluation Management Plan (MTP) covering all anticipated T&E activities and deliverables. Integrate the test plans with the overall project and program plans, and ensure that resources and responsibilities are understood and assigned as early in the project as possible.

Most projects fail to address T&E issues early enough. The MTP helps overcome this problem and makes the testing effort and strategy visible to and understood by everyone on the project. At a minimum, the MTP should encompass the total T&E effort and should assign responsibilities for all major tasks and deliverables at all levels of evaluation. The intent is to provide the big picture and to help coordinate the overall T&E effort.

The MTP is intended for all members of the team, including users, customers, and managers. Activities for everyone involved in evaluation are described, including those assigned to developers, such as unit review and testing. Program managers and those outside the project will find the MTP helpful in relating the testing process to the overall project and

its associated risks. The MTP supplements the Project Plan and addresses management and technical issues associated with evaluation. As the project proceeds, the MTP is revised and updated to reflect current expectations, deliverables, and assignments.

Creating the MTP does not require much effort, and it need not be an especially lengthy or weighty document. Much of the content can be developed effectively in a RAD-like, brainstorming session early in the project.

BEST TESTING PRACTICE #3

MAKE PREVENTIVE TESTING PART OF ALL SPECIFICATION WORK

Establish systematic evaluation and preliminary test design as a sizable part of all system engineering and specification work. Design test scenarios early, and use them in reviews and inspections to help ensure you are building the right features and to better understand acceptable and required system behaviors.

The central idea here is to exploit the power of early test definition and the proven benefits of systematic evaluation (such as analysis, reviews, and inspections) to ensure that the right system and requirements are specified, and that design problems are discovered as early as possible. In commercial industry this is sometimes referred to as preventive testing. It is built on the observation that one of the most effective ways of specifying something is to describe in detail your criteria for accepting (or testing) a product under development.

Key Elements of Preventive Testing

- Tests are used as requirements and usage models.
- Testware design is closely linked with software design.

- Software and testware specifications are reviewed and inspected together.
- Testers and developers work together.
- Most defects are detected before the software is built and tests are run.

With preventive testing, requirements and tests are closely linked. The hardest job with many tests is understanding what results are required or acceptable. By detailing specific test cases and conditions ahead of time, preventive testers can interpret and clarify required system behavior, and many defects can be detected before a single test is run.

Another challenge testers face is the temptation to build, test, and rework designs and features that aren't needed or shouldn't be used. Early T&E emphasizes the *right* work and stresses simplicity as well as defect detection. Perhaps the worst waste of project resources is to test a complicated system thoroughly with the wrong features and design.

Preventive testing boils down to one lesson: It's not about bug counts or density; it's about being smarter!

BEST TESTING PRACTICE #4

USE TESTS AS PROGRESS AND MILESTONE GATES

Require the use of tests to verify that all project deliverables and components are complete, and to demonstrate and track true project progress. Avoid delay in addressing problems by making test completion a gate to be achieved before work product sign-off.

Many projects seem to be in good shape until they enter integration and system test, when suddenly even simple tests don't run. You can avoid surprise failures like this by using tests as gates to control and sign off project work products. When you insist on running tests to demonstrate that a feature or object is complete, you ensure that the design and code are truly complete. Without a program test gate, developers under pressure to show interim progress may be tempted to turn over designs or code on schedule but with key internal design decisions unmade and significant (even fatal) open bugs and issues. Testing becomes unmanageable when it reveals that previous development phases have postponed key decisions and design issues. Using tests as gates ensures that major design issues are closed when they need to be.

Use of test gates also significantly improves project tracking and schedule prediction accuracy. It is much easier to estimate during development coding how long it will take to complete a small number of features than it is to estimate during system test how long it will take to close bugs that range across all or most of the system functionality. Completing features incrementally also means that if you get behind, you can consider shipping without the unfinished features.

The philosophy is simple: If a design, unit, object, or build doesn't pass its tests with suitable coverage, it isn't done and can't be reported as coded or completed.

BEST TESTING PRACTICE #5

INVENTORY TEST OBJECTIVES AND DESIGN FOR TESTABILITY

Develop and maintain a risk-prioritized list of test requirements and objectives: requirements-based, design-based, and code- or implementation-based. Use the inventory to help ensure work products are designed for testability, to guide test design and development, and to trace or cross-reference what each test covers.

A key modern testing practice is to create inventory lists of objectives to be covered before tests are designed and built. The lists detail what needs to be tested, and are used to guide and drive test design and development work. Prioritization by risk (usually into low, medium, and high) helps concentrate testing on the highest-risk areas.

Guidelines for Obtaining Good Test Inventories

- Start early and cover any requirements or functional design specifications.
- Use brainstorming and informal meetings to create worry lists.
- Evaluate the inventories during reviews.
- Group the inventories into three major increments: requirements, design, and code.
- Break each increment into a small number of logical groups.

Since inventory lists can get quite long, it helps to group them into related categories or classes. Missing objectives are easier to identify from these groups.

As each test is developed, related inventory objectives and groups are recorded as part of the test description. With some simple test management tooling support, you can report on the test coverage of requirements and objectives, especially of objectives not yet covered by any tests.

BEST TESTING PRACTICE #6

TEST EARLY, TEST OFTEN

Use tests as early and as often as possible to provide developer feedback and get problems found and fixed as they occur. Emphasize testware engineering as a concurrent life cycle process tightly coupled with software design and development.

An important philosophy of modern testing is to get as much feedback as early and as often as possible. This implies that tests have been prepared early and are shared with the development teams for use in review walkthroughs, unit testing, prototype evaluation, early simulations, and so forth. The aim of early testing is to uncover any surprises and bad news as early as possible, and to help developers produce higher-quality and better-tested units.

This approach assumes that the problems and bugs found in the early testing will be addressed and fixed. Many project managers postpone fixes until developers complete the feature design and code. This is analogous to remodeling every room in your house at once while leaving all the little problems you run into to be fixed “later.” It is generally much better to complete one room or feature at a time and make sure it is really done properly. Run as many tests during development as possible, and fix as many of the problems and bugs as soon as you can.

Reasons to Fix Problems Now, Not Later

- Changes made weeks or months later are highly error-prone.
- Design decisions and subtle code limitations and conditions are easily forgotten.
- Time that would be required to reanalyze your designs and code is saved.
- Early feedback on problems helps prevent similar errors from being made.
- Developer testing is more thorough when all code is tested together.
- Defect- and problem-tracking overhead is significantly reduced.
- Rework of the design and coding is feasible, if necessary.

BEST TESTING PRACTICE #7

DESIGN AND DEVELOP TESTWARE AS DELIVERABLE COMPONENTS

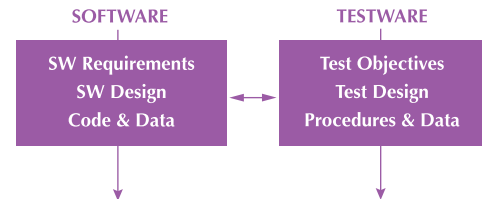
Design and develop major testware components and procedures with the same discipline and care reserved for software components. This includes planning, analysis, design, review, configuration management, and change control and management reporting.

It is important to recognize that testware (such as test suites, simulators and drivers, special evaluation programs, and utilities) is also software and needs to be managed and engineered just like other software components in your application. The testware is really a specialized application system whose main purpose is to test and evaluate some other application or system. If the application is critical, the testware that tests it is critical and must be built with good engineering discipline, including proper testing and evaluation.

Integrated Testware Characteristics

- Recognized as true project deliverable(s)
- Engineered and developed in parallel with software deliverables
- Changed and updated as the software changes
- Used for retesting the software as it evolves

The life cycle process for engineering testware is essentially the same as for software. All the concepts and principles of good software engineering apply equally to good testware engineering. The basic phases of analysis, design, development, and implementation in staged increments apply just as they would for any other software component.



The key to effective testware engineering is to get the timing right. If you build your testware late, after most of the software components have been designed and coded, you don't get the prevention and positive feedback or test-gating benefits. If you try to build it too early, before the software designs and requirements have stabilized, you can't develop your test requirements and objectives effectively, and you face a lot of rework as the software changes.

BEST TESTING PRACTICE #8

PROVIDE INTEGRATED TEST TOOLING AND INFRASTRUCTURE SUPPORT

Support and enable project team testing and evaluation with a T&E database or repository and an integrated test management system for describing, documenting, and tracking tests; also supply support tools, such as simulators, coverage analyzers, capture playback tools, test drivers, and utilities.

Large-scale testing, like large-scale software, needs significant automated support to be performed and managed effectively. This is particularly true when we recognize evaluation as everyone's job (Best Testing Practice #1) and coordinate workflow across the entire project team.

Types of Tools and Applications

- T&E asset tracking and management
- Test plans, objectives, cases, procedures, and scripts
- Analysis, review, and test results and reports
- Work product analyzers
- Test drivers, simulators, and playback systems
- Test coverage analyzers

- Defect and problem tracking
- Process-effectiveness analysis and metrics

A key automation support element is a central project repository for all T&E deliverables and work products. This is referred to as a T&E management system, and includes facilities for saving, describing, documenting, and tracking tests, and recording and tracking review and test observations and results. The best tools make this information easily accessible to the project team, and provide substantial workflow support to simplify and track the software development process. These tools feature templates for all document types, built-in analyzers and checkers for conformance to standards and the T&E process, and templates for periodic tracking reports and metrics.

Other important tool support includes test beds and simulator tools to support various test environments; static analyzers and comparators to provide before- and after-change analysis and assessments of work product risk and complexity; test driver and capture playback tools to run and rerun automated tests; and dynamic analyzers to measure and report test results and coverage.

BEST TESTING PRACTICE #9

MEASURE TEST COSTS, COVERAGE, RESULTS, AND EFFECTIVENESS

Collect information to monitor T&E costs, results, and benefits. Measure test coverage of requirements, design, and code at all levels and for all major builds and test environments. Measure what tests are run on what software versions and what each test level or major activity finds or misses. Use the information on an ongoing basis to assess and evaluate T&E effectiveness.

You can't manage what you can't see or understand, and to understand T&E you have to collect and track data about the test process and its effectiveness.

What Should Be Measured?

- Effort and costs spent
- Test coverage against requirements and objectives inventories
- Test coverage against code (requires dynamic analyzer tools)
- Status of the test effort—what has been done, what hasn't
- Defects and problems found
- Defects and problems missed (collected during operational use)

- Quality of the product—required vs. expected end user perceptions
- Quality of the testing effort—prevented and found vs. impact of what is missed

Think about your last completed program or project. Did you track what tests were run and what they covered? Did you report what was spent at each test level (reviews, unit, integration, system, and acceptance testing)? Did you analyze and assess the effectiveness of that effort and expenditure? Do you know what each test level missed or failed to discover? Can you relate these measures to other projects and past test efforts?

If you can't answer yes to these questions, you don't have the information you need to manage effectively. At a minimum, you should have visibility of the effort and costs spent on T&E activities, the deliverables and work products produced, and the overall results and effectiveness of the test coverage.



BEST TESTING PRACTICE #10

COACH AND MANAGE THE TEAM

Provide ongoing T&E leadership and management support so that everyone on the team knows what is expected and takes testing seriously. Understand your testing activities and process, and make measurement-driven policy and process changes as needed to steadily improve over time.

Testing in most organizations is more mismanaged than managed. Many software managers act more like fans and cheerleaders than quarterbacks. Any really professional team needs good coaching and software. T&E is no exception. Modern testing requires managers to get off the sidelines and lead the team, to shape priorities and set direction.

Managers should get more personally involved in T&E policy and direction. T&E issues seldom involve purely technical or simple yes/no decisions. There are almost always tradeoffs to consider and risks to judge and balance. Management's job is to provide timely, appropriate guidance on everything from scoping the T&E effort to deciding when to stop or what to do when quality targets and schedules are missed.

The principles for managing testing and evaluation are fundamentally the same as those for managing anything else. There are no easy solutions. You've got to assign responsibilities and accountabilities, establish a solid, integrated process with strong automation support, involve good people who know what they are doing, and consistently support improvement over time. The basics of good planning, risk management, measurement, tools, organization, and—above all—professionalism apply to T&E just as they do to any other project area.



AUTOMATED TESTING

Many automated tools are now available to support software testing. Integrated tools are just as important to testing and evaluation as they are to software development and management, and use of these tools should enable the following major benefits:

1. TEST REPEATABILITY AND CONSISTENCY

Automated tests should provide the same inputs and test conditions no matter how many times they are run. Human testers can make errors and lose effectiveness, especially when they are fairly convinced the test won't find anything new.

2. EXPANDED AND PRACTICAL TEST REUSE

Automated tests provide expanded leverage due to the negligible cost of executing the same test multiple times in different environments and configurations, or of running slightly modified tests using different input records or input variables, which may cover conditions and paths that are functionally quite different.

3. PRACTICAL BASELINE SUITES

Automated tests make it feasible to run a fairly comprehensive suite of tests as an acceptance or baseline suite to help ensure that small changes have not broken or adversely impacted previously working features and functionality. As tests are built, they are saved, maintained, and accumulated. Automation makes it practical to run the tests again for regression-testing even small modifications.

NOT THE SILVER (EVEN TIN OR IRON) BULLET

These benefits are not easy to realize. The tools themselves are fairly easy to use, but ease of use doesn't mean success. A hammer and saw are easy to use, but good cabinetwork takes great skill and much more than just good tools. Similarly, a new test automation tool, by itself, will do little for your test program. Most new owners of testing tools tend to overpromise and underdeliver, at least initially, so be cautious about raising expectations too high, too soon.



REQUIREMENTS FOR TEST AUTOMATION SUCCESS

Successful test automation generally goes hand in hand with two critical ingredients:

1. A WELL-UNDERSTOOD AND STABLE APPLICATION BEHAVIOR

You can't automate something that is not well defined. Applications that are still undergoing significant requirements and design changes make poor candidates for test automation. Even fairly stable applications may be problematic if the testers responsible for automation don't understand an application's behavior or the relevant domain-specific issues and needs. To automate successfully, you must understand and be able to predict application behavior, and then design the test in such a way that your test plan stays with the tool.

2. A DEDICATED AND SKILLED TEAM WITH SUFFICIENT TIME AND RESOURCES ALLOCATED

Remember, testware is software. Test automation should be managed like a separate support project with its own budget and resources. You will generally fail if you give your whole project team a new automation tool and ask everyone to do a little automation in their spare time. (Imagine trying to automate your accounting or payroll process by

giving everyone in the finance department a spreadsheet and asking them to work on it when they have some free time.)

HOW NOT TO SUCCEED WITH TEST AUTOMATION

- Buy a tool and give it to everyone on the team.
- Presume automation will save project time or money.
- Expect a lot of bugs to be found.
- Test functions and features you don't understand.
- Test functions and features that are changing significantly.
- Automate tests you don't understand how to run manually.
- Avoid a project.
- Avoid any dedicated resources.
- Eliminate training and support.
- Ignore synchronization.
- Ignore test documentation.
- Forget about environments and configurations.
- Ignore test management.

TESTING PARADOXES

Following are some common paradoxes faced by those involved in testing. Contemplate these for your benefit—and at your own risk!

1. The best way to test and verify requirements is to figure out how to test and verify requirements.
2. The best testing process is one that forces developers to build the software right and test it as they go, which in turn makes it largely unnecessary to have a best testing process.
3. The best way to build confidence is to try to destroy it.
4. Testing is looking for what you don't want to find. A successful test is one that fails, and a failure is one that succeeds.
5. Managers who fail to spend appropriately on testing end up with failed projects because they can't meet their deadlines if they do spend appropriately on testing.
6. Unsystematic testing can and should be very systematic.

7. Heavy use of automated tests eliminates the need for heavy use of automated tests.
8. A big improvement in testing can ruin a project by giving it more problems than it has time to deal with and making the software appear much worse than it is.
9. Quality is free if you pay for it.

WHEN IS IT OK NOT TO TEST?

- When the responsibility for failure can be shifted to someone else
- When the impact or significance of any problem is insignificant
- When the software does not have to work
- When no one is or will be using the system
- When the project has already failed

TESTING IS *NOT* A PHASE!

A central principle of modern testing is that good T&E proceeds in parallel with software development. Testing should not be viewed as just execution or just the phase that happens after coding. For maximum benefit and leverage, test development and preparation need to be completed before, rather than after, coding.

Testware engineering is more significant and integral to the analysis and design stages than it is to the coding and implementation stages. This is analogous to testing in school: The final may be important for demonstrating knowledge and graduating, but it is the testing and evaluation during the semester that really helps students gauge their progress.

WHAT SHOULD YOU SPEND ON TESTING?

FUNDING FOR THE TESTING PROCESS DEPENDS ON:

- The RISK
- The application complexity and change
- The talents and skills on the team
- The tooling and infrastructure support
- The quality of the software effort
- The availability of existing tests
- The process and methods
- The degree of retesting required

Plan for a minimum of 25 percent of total program costs.

High-risk components (such as software modules, units, and configuration items) as well as system features tied to mission-critical or safety concerns need more thorough testing (more insurance) and may well drive up the proportionate cost of testing and evaluation to 50 percent or more of the program budget.

KEYS TO TESTING AND EVALUATION IMPROVEMENT

ADDITIONAL INFORMATION

MANAGE T&E AS AGGRESSIVELY AS YOU MANAGE DEVELOPMENT

- Take testing seriously.
- Foster a quality culture that wants to find and prevent problems.
- Set clear directions and expectations.
- Delegate responsibility and accountability to good people.
- Manage the process and the project.
- Emphasize early planning and preparation.
- Make testware a real deliverable.
- Insist on tests to demonstrate progress.
- Invest in team tooling and support infrastructure.
- Measure effectiveness.
- Reward team contributions.

CAUTION

Reengineering the T&E process is known to be a tough—even wicked—problem because managers, developers, and testers are often highly resistant to change. *Can'titude* will flourish, and attacking it may be harmful to your well-being.

Bear in mind this advice to mountaineers: It is very dangerous to leap chasms in two or more bounds.

There are many excellent publications, training courses, and conferences that address software testing and modern, best testing practices. Useful resources include:

BOOKS

Boris Beizer, *Software Testing Techniques & Black Box Testing*, Van Nostrand Reinhold

Tom Gilb and Dot Graham, *Software Inspection*, Addison Wesley

Bill Hetzel, *Complete Guide to Software Testing*, John Wiley & Sons

Cem Kaner et al., *Testing Computer Software*, Course Technology

Ed Kit, *Software Testing in the Real World*, Addison Wesley

Brian Marick, *Craft of Software Testing*, Prentice Hall

William E. Perry and Randall W. Ricer, *Surviving the Top Ten Challenges of Software Testing*, Dorset House

WEB SITES

Client Server Software Testing Technologies:
www.csst-technologies.com

Software Quality Engineering: www.sqe.com

Software Testing Institute: www.ondaweb.com/sti

Software Testing Laboratories: www.stlabs.com

ACKNOWLEDGMENTS

Much of the material in this booklet has been drawn from my consulting and teaching experience at Software Quality Engineering and from several of my own publications, especially *The Complete Guide to Software Testing*.

The importance of effective test and evaluation practices for managers cannot be stressed enough. It is my hope that this booklet adequately addresses the need to support this all-too-often-neglected process.

Special acknowledgment and appreciation are due to Ed Kit, Dot Graham, Dave Gelperin, Boris Beizer, Linda Hayes, Tom DeMarco, and Steve Maguire for their own publications and insights that are directly reflected in the assembled material.



Bill Hetzel



NOTES

THE AIRLIE SOFTWARE COUNCIL

Victor Basili	University of Maryland
Grady Booch	Rational
Norm Brown	Software Program Managers Network
Peter Chen	Chen & Associates, Inc.
Christine Davis	Texas Instruments
Tom DeMarco	The Atlantic Systems Guild
Mike Dyer	Lockheed Martin
Mike Evans	Integrated Computer Engineering, Inc.
Bill Hetzel	Qware
Capers Jones	SPR Inc.
Tim Lister	The Atlantic Systems Guild
John Manzo	3Com
Lou Mazzucchelli	Gerard Klauer Mattison & Co., LLC
Tom McCabe	McCabe & Associates
Frank McGrath	Software Focus, Inc.
Roger Pressman	R.S. Pressman & Associates, Inc.
Larry Putnam	Quantitative Software Management
Howard Rubin	Hunter College, CUNY
Ed Yourdon	American Programmer