What Happens When You Don't Have a Test Plan?

By: Donna O'Neill, IV&V Australia

Software companies can no longer afford to develop and test their systems on an ad-hoc basis. Without a defined testing strategy, companies will get left behind in the increasingly sophisticated world of software development.

A well-defined test methodology will help to ensure that companies develop their products in the most efficient and cost-effective manner.

When a project does not identify its overall approach to testing, there is no end to the number of problems that can arise. This paper:

- *Identifies the four most commonly seen problems*
- *Highlights the effects that these problems can have on a project*
- *Discusses some ways to fix these problems (or to avoid having them in the first place).*

What is strategic test planning?

Most projects develop software test plans. However, these plans usually only identify the testing that will be conducted by the Test Team on a particular part of the system or for an incremental release. They do not produce a plan that takes a "big picture" view of the project's approach to testing.

Software Test Plans do not usually identify:

- The way in which the testing conducted by the Development Team fits in with the testing conducted by the Test Team
- The strategy for progressing the software and the tests from one build/release to the next (including regression testing)
- The approach to the test environment that identifies the progression of testing on an artificial development environment through to testing on the deliverable system.

Strategic test planning provides a mechanism for ensuring that:

- The testing effort is applied throughout the development cycle to maximise test coverage
- The test program facilitates and demonstrates progress towards completion
- There are clear objectives for documents and tests
- Teams do not spend time writing documents when they could be testing software.

Where do you document test strategies?

Project Managers and Test Managers engage in strategic test planning to help their teams conduct test activities in the most efficient and effective way possible. To that end, documenting the test strategy should not add to the documentation burden.

The information should be clear and to the point, using process diagrams, tables and checklists to convey information.

The strategies can be presented in several ways:

- As part of another project document (eg, a *Software Development Plan*)
- A separate formal project document (eg, a *Master Test Plan*, *Test & Evaluation Master Plan*, or *Test & Evaluation Program Plan)*
- An informal project document (eg, *How We Do Testing*, *Tester's Survival Guide*)

Separate formal documents are favoured by most new projects for Defence applications. Informal project documents are favoured by commercial projects that are not required to deliver a strategic plan to an outside customer. There is no reason why all three approaches cannot be reasonably similar, and equally useful.

For the purpose of this paper, the term *Master Test Plan* will be used.

What happens when you don't have a test plan?

When a project does not have an overview document to tie the testing process together, there tends to be no process. This results in issues falling through the cracks or being duplicated unnecessarily.

The four most commonly seen problems are as follows:

- No clearly defined roles and responsibilities
- No clear test objectives
- Ill-defined test documents
- No strong feedback loop.

Problem 1: No clearly defined roles and responsibilities

The Effects

When a project does not have a Master Test Plan, where the roles and responsibilities between and within teams are identified, important tasks often get left undone. This is primarily because there is no "ownership" of the tasks.

For a test team, this is most obvious during Reviews of documents and activities:

- No one allocates enough time to do them
- No one makes sure all aspects of the task or document are reviewed
- No one makes sure the purpose of the review is served before moving on to the next task (ie, that the entry and exit criteria for the review has been met)
- No one makes sure that review action items are created and/or addressed.

This can be seen on all parts of a project, however given that the testing activities tend to follow development activities, the impact of poor reviewing is greater during testing.

When roles and responsibilities are not defined, tasks often end up being done in an inefficient manner, by people who are not best qualified to do the job.

This often occurs with unit/module testing, where developers think that because there is a separate "test" team, they do not have to do any testing themselves. Obviously, they are wrong. Furthermore, they are costing the project time and money.

Both white-box testing of the code and black-box testing of requirements must take place to ensure adequate test coverage. The developers are best qualified to conduct white-box testing, since they are closer to the code. This allows the testers to remain as independent of the design as possible, so they can evaluate the system objectively, from a user's perspective.

In addition, misunderstandings of who is supposed to do what tasks leads to an Us vs Them attitude, and alienation between individuals and project teams.

The Solution

The best way to eliminate confusion about who is supposed to do what, is to clearly define roles and responsibilities as early as possible. This can be done by developing checklists to scope the tasks and by diagramming processes. Even just simple tables with the responsibilities for doing the tasks, reviewing the results, and accepting that the task is complete can help considerably.

It will not be possible to foresee every task that will come up throughout the development lifecycle. However there are tasks that always occur and always require interaction between teams, for which roles and responsibilities can be defined in a Master Test Plan. These include:

- Requirements analysis
- Unit/module testing
- Integration
- Software handover from development to test
- Functional/system testing
- Problem reporting
- Final acceptance or system delivery.

Problem 2: No clear test objectives

The Effects

When a project does not have a Master Test Plan, project teams do not usually have a clear understanding of the different types of testing that should be done on a system, and the reasons for doing the different types of tests.

Without clear objectives for running tests, there is an increased likelihood that critical and essential system characteristics will not be adequately tested. This ties in with roles and responsibilities for doing tasks. Each test level serves a particular purpose. If this purpose is not well understood and tied together in a systematic way, tests tend to be ad-hoc, with unnecessary overlaps as well as gaps in coverage.

Projects end up with developers doing the same type of testing as testers, and worse, no one doing some essential type of tests. Developers and testers spend inordinate amounts of time finding problems that clearly-directed testing would have found the first time through.

Also, without a clear "driver" for each type of testing, it will be very difficult to measure how far the task has progressed. If you don't have a goal, how do you know when you are finished?

The Solution

To ensure that test coverage is done most efficiently and effectively at all levels, it is essential to look at the big picture. Determine all

of the different types of tests that should be done on the system, and clearly identify:

- The objectives for testing at each level
- The entry and exit criteria for each test level
- Who is responsible for doing the tests, reviewing the tests and results, and accepting that the software has successfully passed through the test level and is ready for the next level.

Examples of test objectives may include:

- Execute all lines of code (or paths) for unit/module testing
- Demonstrate all requirements for functional testing
- Exercise all interfaces for integration testing.

The following information should be included in a Master Test Plan to define each level of testing.

- Purpose
- Test objective(s)
- Responsibility
- Methodology
  Tasks
  Types of tests to be performed
  Referenced procedures
- Documentation
- Relationship to other test activities
- Completion criteria.

Problem 3: Ill-defined test documents

The Effects

When a Master Test Plan does not identify the role of each test document, projects often do not have a clear understanding of *how* and *why* to write the different documents.

As with test levels, each test document serves a particular purpose. If this purpose is not well understood and the documents are not tied together in a systematic way, the test documents tend to be incomplete and ad-hoc (and very inefficient). This can lead to an ad-hoc and inefficient test process.

Even when projects use documentation standards as a basis for writing documents, strategic information is often omitted in favour of slavish adherence to section headings. This missing information often includes:

- Strategic overviews
- Test design rationale.

These documents pass the "weight test", but never really tell the reader how well the system is being tested.

This can be seen particularly on projects that attempt to automate their test documents. They load all of their requirements into a database, and then to use the database to allocate requirements to tests. They churn out vast quantities of details about a test, without ever providing a clear overview on what the testing effort is trying to achieve, and why it is trying to achieve it.

While testing activities lend themselves very well to automation, too often manipulating the database is used as a substitute for actually using your brain to design tests. People design tests. Databases are only a tool.

The Solution

Identify the purpose and scope of each test document, and establish design criteria for tests, test cases and test procedures. Include this information in the Master Test Plan, then make sure that this strategy is used when writing the documents.

Good test documents reflect the structure of the requirements specifications (assuming, of course, that the requirement specifications are good). Tests should correspond to functional capabilities (ie, groups of related requirements) and test steps should correspond directly to requirements.

Document templates and checklists can help to ensure that the strategy is consistently followed across the whole team. They can eliminate some of the burden associated with writing documents by providing guidance on what the documents should contain and where to obtain the information.

Checklists are also a useful tool for reviewing the documents. If a document successfully meets the criteria in the checklist, the review can be deemed to have been passed and the team may proceed to the next task without risk of having to revisit the completed task.

As long as the rationale for writing documents and tests is clearly understood and communicated in the test documents, and this is built into the automation process, test documents can be automated quite reasonably and to great advantage.

Use automation wisely. It is no substitute for good test design and clear presentation of ideas.

The following information should be included in a Master Test Plan to define each test document:

- Purpose of the document
- Scope of the document
- Testing strategy
  Test criteria
  Test design principles
  Requirements tracing
- Information provided
- Standards/DIDs used
- Relationship to other documents
  Preceding documents
  Following documents
- Reviews

Problem 4: No strong feedback loop

The Effects

Feedback between testers and developers (and management) is central to the development process, particularly for project management, who need to use this process to monitor/measure progress, schedule and software stability.

Where a Master Test Plan does not identify the process of Testing → Problem reporting → Problem resolution → Retesting, this feedback tends to not occur.

This can be seen in two areas:

- Where the problem reporting/resolution mechanism is not clear, it is less likely that all problems will be reported and resolved.
- Lack of data affects the project team's ability to measure software quality over time, to plot the increasing stability (or not) of the product.

The net result of this is that the product is shipped with problems, which are always more difficult to find and costly to fix when the software is in the field.

This lack of process can also be seen to result in an increased uncertainty about the roles and responsibilities of the team members involved in the loop.

The Solution

Develop a mechanism for raising and investigating problem reports, resolving the problems, and retesting the affected software. This could be documented/diagrammed in a Master Test Plan, and all team members should have visibility to it.

The mechanism may involve the use of a problem management tool, or a test management tool associated with a suite of test tools already in use. The key is to increase the visibility of this process so that it is used to its best advantage in managing risk.

The plan should include:

- Guidance on how and when to raise problem reports, to ensure that every observed defect is recorded
- Samples of the metrics/statistics that should be maintained by the testers
- Guidance on interpreting the statistics, to determine software quality and manage the risk areas
- Guidance on how to go about retesting the software (including unit/module testing).

Conclusion

The problems identified above are not difficult to fix. They merely require that Project Managers and Test Managers stand back and think about the big picture.

Most projects have individual activities defined fairly well. What is required, is to identify the interactions between the activities and between the teams responsible for completing the activities.

Managers should remember that one of the characteristics of a strategic plan is that it is a "living document". As the project progresses, and the tasks and interactions become better understood, the plan should be fine-tuned to reflect this new understanding. If this does not occur, the document will get added to the project's collection of "shelfware" documents that never serve their once-intended useful purpose.