Clearing the Testing Minefield

By Donna O'Neill, IV&V Australia

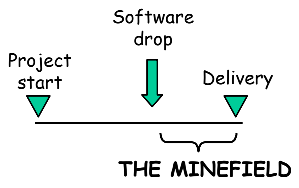
Contact details *IV&V Australia Pty Ltd*
Suite 3A, 10-12 Clarke Street, Crows Nest, NSW, 2065
Phone +61 2 9957 6577; Fax +61 2 9957 1522
Email donnao@ivvaust.com.au; URL http://www.ivvaust.com.au

Abstract *As testers, we constantly have to dodge all sorts of obstacles that get in our way. We never have enough time to do adequate testing; we rarely have clear requirements on which to base our tests; we battle for our own test environment, and we beg for robust software that works well enough for us to do meaningful testing. The list goes on. Our challenge is to ease our way through this minefield, so that we can maximise the time we spend adding value to the project, and reduce the time we spend responding to non-productive diversions.*

This paper will provide some strategies for meeting this challenge and achieving real value-added testing.

What is the testing minefield?

Software testers often spend much of their time working in a critical phase at the end of the development lifecycle. This phase lies between the release of software from development and the final system delivery to the customer.



It is during this phase that testing is supposed to be providing its major contribution to the project, yet it is also the time when the project is under the most schedule pressure.

When not managed properly, this phase is like a minefield, with hidden dangers everywhere. However, when appropriately managed, the minefield can be cleared and traversed in safety.

What dangers lie in the minefield?

The testing minefield is littered with obstacles that conspire to make our job a very difficult one.

To start with, project planners rarely allocate sufficient time to conduct adequate testing. This situation worsens when the inevitable development delays occur, which then eat into the fixed amount of time allocated for testing.

Testers face the unenviable task of trying to get their jobs done within an ever-reducing window of opportunity, compromised by factors such as:

- The big bang approach to testing
- The folklore syndrome of requirements definition
- Fuzzy non-functional requirements
- Assumed quality of components
- The software surprise from developers
- The test environment set-up rush
- The shaky foundation of unstable software
- The path of least resistance.

These factors contribute to hurried, inadequate test coverage, the release of low quality software, and tester burn-out.

How can we clear the way (and deal with the minor explosions)?

The key to efficient and effective testing is to use a risk-based testing strategy, where you identify in advance as many of these "gotchas" as possible. Be open and honest with yourself and your management about what you can and cannot achieve in the allocated time and plan your test strategy accordingly.

By understanding the typical testing hazards, we can foresee and manage them before we reach the minefield, or at least reduce their impact when they inevitably occur.

The goal is to do as many up-stream tasks in advance as possible, to maximise the time you spend actually testing. This way, you will maximise the effectiveness of your testing by applying the testing at the right time in the lifecycle, with the right people doing the right testing (including developers).

Many of the testing "mines" are essentially communication issues between test and development and management. Increasing the visibility of the testing process, and integrating testing activities earlier into the development lifecycle, will promote a greater understanding of the role of testing on projects. This, in turn, will decrease the diversions caused by ill-timed and non-productive activities and thus maximise the likelihood that projects will get the greatest benefit from the testing activities.

Avoid the big-bang approach

Unfortunately, it is not uncommon in the industry for all testing to be left until the end of the project, just prior to release.

These projects deny themselves the chance to build quality into their product through the visibility and control that early and ongoing feedback can provide.

Late testing can lead to overtime work for testers (leading to burn-out and attrition), schedule chaos when major defects are found at the last minute, or the release of poor quality software.

The key to avoiding last-minute schedule chaos is to remove as many tasks as possible from the minefield phase by employing an incremental end to end testing approach:

- Involve testers in the project early, during the requirements review process, and start the test plan immediately afterwards.
- Use an incremental development strategy with short builds. Release working threads of functionality to testers early and often.
- Use tester feedback to focus development tasks. Effective metrics for testers to collect include:
  - The number, nature, and severity of defects found per functional area/test
  - The cumulative number of defects raised vs closed over time (and shown on a graph)
  - The number of "fixed" defects that are actually rejected during retesting.

Manage the folklore syndrome

When a development effort is based around poorly defined or incomplete requirements, the project becomes heavily reliant on the folklore and domain knowledge that exists in the organisation. Unfortunately, the keepers of this knowledge are often reluctant to document it.

Testers, in an attempt to define their objectives, often try to capture this information themselves, without the budget for this unplanned work or the skills to do it well.

Without clear requirements, tests tend to be inefficient and poorly directed (rather than systematic) and tend to demonstrate what the system does rather than verify what it should do. Time is wasted in arguments between teams and customers over differing views of the product implementation and test results.

To minimise the impact of the folklore syndrome, ensure that the requirements are well understood and agreed, and that all test objectives are clearly defined, *before* you enter the minefield phase.

To improve your approach to requirements:

- Encourage the development team to undertake a systematic requirements gathering activity.
- Ensure that the specification has been reviewed by all stakeholders (customers, developers, testers) for completeness, clarity, testability, etc. A requirements characteristics checklist can help to focus this review activity.

To define clear test objectives:

- Identify the "test basis" for each level of testing, so that all tests have clearly defined objectives (eg, use cases, business rules, functional requirements).
- Use requirements tracing techniques to help focus testing and ensure that all tests systematically meet their objectives.

Tie down fuzzy non-functional requirements

Non-functional requirements such as quality attributes (eg, reliability, availability, user friendliness, etc) and performance objectives can be very difficult to specify. This means that they are also very difficult to test and objectively verify. At the same time, these characteristics are usually very important to the customer.

Testers, in their role of defacto customer liaison on most projects, are the ones who bear the brunt of the disagreements that this situation may cause. These disagreements often occur at the end of the project lifecycle (ie, in the testing minefield) and can cause costly delays to product release and/or acceptance.

The obvious answer is to encourage the authors of the specification to characterise these attributes in quantifiable, testable terms. If they will not, or can not, cooperate, then testers can appeal directly to the customer, to gain agreement on acceptance criteria for these requirements. Technical support staff may also be a good source of this information, which can then be used to determine the expected results of the test procedures.

Do not assume quality components

For systems that are upgrades to existing systems or are developed using bought-in/re-used components, the testing strategy needs to consider the quality history of the components. Once integrated into the system, the project takes on the problems of these external components.

Unfortunately, it is very difficult to really know for certain the true status of re-used or bought-in components. However, the impact of the problem can be reduced:

- Component quality should be assessed prior to the minefield phase, through analysis of prior test results, analysis of assessment data that was gathered during the component acquisition process, software inspections, hands-on testing, and feedback from other users.
- During system testing, target specific risk areas such as interfaces and integrated system behaviour.

Spoil the software surprise

One of the biggest contributing factors to testing mayhem is the reliance of the test group on other parts of the organisation.

Testers need to know, in advance of receiving software, the order in which functionality is being developed so that they can plan their test development schedule. Failure to receive some advance notice forces testers to delay test execution while they hastily prepare their tests, gather test equipment, and wait for appropriately skilled testers to be available.

Communication is the key to ensuring that testers have advance warning of the timing and contents of software drops. It is far more effective for testers to design their tests in advance of reaching the minefield phase, to maximise the amount of time they can spend running tests and providing feedback.

The software, when it is released, should contain testable (vertical) slices of functionality so that testing can commence immediately. If testers only receive diverse bits of functionality that cannot be tested until the system is fully integrated, they are forced into a *big-bang test approach* or into wasting time on the construction of throw-away test harnesses.

Avoid the setup rush

One of the most time consuming tasks facing testers is the establishment of the test environment with the required hardware, support software, tools and test data. On many projects, however, testers only begin this task upon receipt of the software for testing.

This means that valuable testing time is eroded away by this set-up task. To maximise the testing time, the test environment should be established by the time test running is scheduled to start (at the very latest). This will enable the testers to provide feedback to the developers as quickly as possible.

Do not build on a shaky foundation

When developers do not conduct adequate unit testing, the system that is released to the testers is almost always unstable.

These systems have a tendency to crash unexpectedly and display other such erratic behaviour. This is commonly due to poor exception handling or low level logic errors.

Erratic behaviour makes it very difficult to conduct effective functional and system testing, and so the product requirements get tested less thoroughly. Furthermore, because of the differing focus of the tests, the majority of design and code-level defects will get through to the field regardless of any extra effort applied to functional testing.

Developers must understand the critical and unique role that unit testing plays in the testing lifecycle.

Prior to hand-over to the testers, the developers should conduct unit tests (ideally preceded by design and code reviews), using checklists to help focus the activities and optimise their effectiveness. This should ensure a robust base for independent requirements testing.

Do not settle for the path of least resistance

Developers have a tendency to implement the “easier” functionality first, to show early progress. Testers have the same tendency, for the same reason.

This leaves the “harder” requirements until later, when there is less time available to effectively deal with them. Unfortunately, “hard” requirements are harder because they are more complex and critical to the operation of the system.

These requirements will take longer to implement and they will be more prone to defects. They will also be more difficult and time-consuming to test.

Project and test managers need to give themselves more time to manage high priority/higher risk issues. As early as possible in the development lifecycle:

- Prioritise requirements by (at least) technical and interface complexity, mission or safety criticality, and scope of use.
- Plan the development and testing schedule to implement and test the higher priority requirements as early as possible.
- Plan to review the tests (and corresponding area of the design and code) of these requirements more thoroughly.
- Assign more experienced staff to the higher priority requirements.

Conclusions

To find a path through the testing minefield, it is important to understand what tasks you can and should start doing right from the beginning of the project. Apply your effort throughout the development lifecycle, so that by the time you receive software for test:

- You have already reviewed and understood the requirements
- You know what functionality you will be getting to test, and when
- You have written tests based on clear objectives (eg, requirements)
- You have set up the test environment with the required hardware, software and test data
- The software is robust enough to make meaningful test progress
- You understand the functional priorities, so that you know where to concentrate your efforts if and when time is running short.

Until project teams understand how testing fits in with the development lifecycle, and what the critical dependencies between teams are, then the testing minefield will remain a risky place to be.

To effectively clear the testing minefield, project teams must understand that testing is an integral part of the project rather than an adjunct to it. That is, the success of one team enhances the success of the others, towards the common goal of an effective and efficient development process.

Remember:

- Wherever possible, address risk areas early, before the time-critical “minefield”
- Anticipate the remaining “mines” so that their impact can be limited.