

Surviving Testing Risks – a practitioner’s guide

By Donna O’Neill, IV&V Australia

Risk management forms a central activity of any project and test management strategy. Test managers must live with test planning risks and test technical risks on a day to day basis. The key to surviving these risks is to:

- Gain a clear understanding of the factors that contribute to testing risks (both planning and technical)
- Ensure that you have an adequate level of visibility into the progress and effectiveness of the testing effort
- Establish a mechanism for controlling the contributory risk factors and their adverse consequences.

This paper identifies a number of test planning and technical risks that are typically encountered on software development projects. Suggestions for surviving the risks are presented in terms of the contributing risk factors, the consequences of the risks, and suggested mitigation strategies.

What is risk management?

Risk management forms a central activity of any project and test management strategy. It involves monitoring any number of project areas to identify whether known risk factors have occurred, and if so, that appropriate mitigation strategies are in place to minimise their impact.

Risks to be monitored include requirements, technical issues, the schedule, personnel skills, and social and political issues that may affect progress.

Monitoring for risk can be achieved through the use of reviews, status reports, metrics collection, and a number of different types of testing.

Metrics collection and schedule monitoring form a central part of this activity, by providing the facts required to see trends and highlight missed dependencies, and to determine whether they may impact the project’s progress or product quality.

Why is it important to manage risk?

Risk management enables us to foresee and prevent problems before they occur, or at least reduce the impact of problems when they inevitably occur.

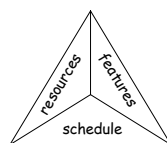
If managers understand the typical risk areas on a development project, they can put a mitigation strategy in place to minimise the risk of problems occurring, and respond in a timely manner.

The importance of understanding the factors that contribute to risk applies equally to project, development and test management, even if the individual factors and their consequences differ.

What are test planning risks?

Effective test management involves a complex strategy of coordinating three main planning factors:

- Testing resources
- Test schedule
- Product features.



The changing nature of these factors, combined with their impact on each other, produces a set of risks that need to be monitored on a

regular basis. Test managers need to be aware that an expansion or contraction of one of the factors will result in the need to adjust the other two factors.

The Test Resource Factors

Testers, testing expertise, and the test environment are all resources that are all too often in short supply.

Risk: The tester drought. Many projects simply do not have enough testers to adequately meet their testing need. Given that most projects have a strong imperative to adhere to the schedule, they end up doing less testing (and thus incurring a higher incidence of faults released to the field).

Where this reduction in testing is inevitable:

- Let the testers get started testing as early as possible – don’t wait until the end.
- Prioritise requirements to focus the testing effort on the most critical areas and key functionality.
- Share the testing load across the project through the use of effective developer-led techniques such as design reviews, code inspections, and unit tests.

Risk: The expertise drought. Even where projects have enough people to assign to testing tasks, they often are not experienced professional testers who have lived through the full development lifecycle and/or are familiar with the part of the industry.

Inexperienced testers, or testers who are not familiar with the technicalities of the system under test, work more slowly and have a tendency to design relatively superficial tests that demonstrate what the system is doing rather than testing what it should do. They are also reluctant to raise defect reports, because they lack confidence in their technical judgement. Teamwork is the key to handling this problem:

- Develop explicit guidelines on how to design and conduct tests.
- Have experienced testers plan and design the tests, and have the less experienced testers flesh out the procedural detail and run the tests.
- Use metrics to provide visibility into how the test running is going (test run rates) and whether defects are being logged (defect raising rates).

- Allow testers to raise issues without fear of being summarily dismissed, and encourage them to report facts, not interpretation. Use a filtering process to sort out software problems (that belong in a defect database) from tester problems (that require management support).

Risk: The environment drought. Developers and testers use their environments differently and with different objectives. Developers constantly change and refine the system, whereas testers need to unambiguously evaluate the state of the system at an identified point in time.

When the environment is shared between the two groups, the schedule must consider the interactions and risk of delays that this may cause. A shared environment also means that the validity and accuracy of the test results can not be assured.

In addition, the lack of a realistic end-user environment creates the risk that system will not operate correctly when it goes “on-line”, regardless of how much previous testing has been done.

It is essential that projects allocate enough hardware, support software, and data to enable the testing to be done in an isolated, controlled environment. This includes an environment that accommodates:

- Testing on all required platform types
- Simulating (or using) the number of expected users and amount of data.

Projects must ensure that they test what they are going to release and they release what they tested.

Risk: The test tool distraction. Confusion over when and why to use test automation tools can seriously affect the success of a testing activity.

Ad-hoc automation without a well-understood underlying process can cost the project a significant amount of time and money, and does not result in a better process. When test automation tools are used inappropriately, overhead costs are rarely recouped.

Test automation tools should only be applied to existing, well-understood manual processes, and to situations where the costs vs benefits are clear. Make an early decision as to whether test tools are appropriate to the project. If they are to be used, integrate them into your process from the beginning.

The Test Schedule Factors

A common characteristic of project/software development schedules is that they are often established without first considering testing issues and the risks associated with testing activities.

Risk: The big-bang approach. Unfortunately, it is not uncommon in the industry for 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 employ 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.

Risk: The schedule squeeze. Project planners do not always 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.

When combined with a shortage of experienced testers and a big-bang approach to testing, the results include hurried, inadequate test coverage, the release of low quality software, and tester burn-out.

Mitigating these effects requires the same techniques used for many of the other resource and schedule risk factors. That is:

- An incremental end to end test methodology
- Focussed testing and test priorities
- Effective developer test and review activities.

The Product Features Factors

One of the most fundamental planning risks that testers can face is the lack of clear test objectives. For independent testers, their primary objective is (or should be) to verify that the requirements have been met and to validate that the system is fit for purpose.

Risk: 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. The resulting tests tend to be inefficient and poorly

directed (rather than systematic) and tend to demonstrate what the system does rather than what it should do. Time is wasted in arguments between teams and customers over differing views of the product implementation and test results.

It is essential that product requirements are clearly defined and understood by all parties:

- Enforce the preparation of a requirements specification (ie, a functional specification).
- 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.
- Use requirements tracing techniques to help focus testing and systematically ensure that all requirements are covered by tests.

Risk: The moving target. For projects using an evolutionary development methodology, requirements change can be a managed and planned activity. However, on projects that are not using this paradigm, it is also common for customers to want to add or change features during development.

While some “requirements creep” is to be expected, managers need to understand that changes in scope mean that tests may have to be reworked, new tests and test data may have to be prepared, and more tests have to be run.

A thorough technical review (and sign-off) of the initial specification with the customer will reduce the likelihood of ad-hoc changes. However, when changes do occur, be sure to trade off the inclusion of new features/requirements with an increase in the development and test schedule.

What are test technical risks?

A number of technical factors have direct impact on the quality of the testing effort. An ineffective and/or inefficient testing process will increase the risk of the schedule slipping and the product being released with an unacceptable (or unknown) level of quality. These technical risk factors include:

- A reliance on testing done by other teams or companies
- Dependencies between the test team and other groups including development and the customer.
- The characteristics of the system/requirements under development

The Testing Coverage Factors

An effective testing methodology involves the use of different types of testing, done at different points in the development, by different teams. How much or what type of testing to do at each level is dependent on the testing that occurred before and will follow.

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

It is not possible to conduct complex and effective functional and system testing on an unstable system, and so the product requirements get tested less thoroughly. Furthermore, the majority of design and code-level defects will get through to the field regardless of any extra effort applied to functional testing (because of the differing focus of the tests).

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 design and code reviews and unit tests, using checklists to help focus the activities and optimise their effectiveness. This should ensure a robust base for independent requirements testing.

Risk: Assumed quality. For systems that are upgrades to existing systems or are developed using bought-in/ reused 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 components.

Component quality should be assessed, through analysis of prior test results, software inspections, hands-on testing, and feedback from other users. The system test strategy should target specific risk areas such as interfaces and integrated system behaviour.

The Dependencies Factors

One of the biggest contributing factors to testing mayhem is the reliance of the test group on other parts of the organisation. Missed dependencies can provide an early warning of potential problems.

Risk: Dependence on Developers. Testers need to know, in advance, the order in which functionality is being developed so that they can plan their test development schedule. On fixed-length projects, it is far more effective for testers to design their tests in advance, to maximise the amount of time they can spend running tests and providing feedback.

Testers rely on developers to hand over testable builds, rather than diverse bits of functionality that cannot be tested until the system is fully integrated. A failure to receive individually testable builds forces testers into a *big-bang test approach*.

Testers also depend on developers to unit test their software, and correct defects in a timely manner.

The project should undertake a build-planning task, to determine the build make-up and implementation order. The project schedule should identify all dependencies, to quickly signal when testing risks and compromises are occurring:

- Ensure that build planning feeds into test planning.

- Plan to start running tests immediately after the end of unit testing for that functionality.
- Allocate time to run all of the tests one last time on the “to be released” version of software.

Risk: Dependence on the environment. 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.

Plan to complete the test environment set-up task prior to the start of running tests.

Risk: Dependence on the customer. Projects rely on the customer (ie, the users) to tell them what they want the system to do, and to accept the system at the end of the development.

The user requirements form the basis for the development of the test procedures. However, if the requirements are not clearly understood and agreed by all parties, the scope and schedule of the testing effort cannot be effectively planned and managed, and the acceptability of the test results (based on the test procedures) is at risk.

The project also needs to understand which features are the most important to the customer. If they get these features right, the customer will be more eager to accept the final system (and may even be more willing to overlook a few minor problems).

Ways to highlight these dependencies include:

- In the project schedule, plan to complete a user/customer requirements review (and sign-off) prior to the start of the test planning task.
- Identify a manageable set of requirements that are most important to the customer. Define these requirements clearly and plan to demonstrate them to the customer as early as possible.
- Gain customer agreement that the test procedures adequately exercise the system under test. Schedule the review as early as possible, but no later than the start of the final test run.

The Requirements Characteristics Factors

Risk: 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:

- Prioritise requirements by (at least) technical and interface complexity, mission 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.

Risk: 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. 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.

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. The acceptance criteria will determine the expected results in the test procedures.

Conclusions

Effective software testing can make an enormous contribution to the efficient development of quality products that meet the customer’s needs.

In the real world, however, because this potential is not well understood, there is never enough time allocated to testing activities. To get the most out of their limited testing budgets, it is important that projects remove the obstacles and smooth the way ahead of their vital testing tasks.

Projects need to understand and acknowledge that there are typical planning and technical risks that always exist. They need to:

- Anticipate the risks
- Eliminate as many as possible before the project starts
- Put monitoring mechanisms in place to catch the occurrence of the risks (using schedules and metrics)
- React as soon as the risk is triggered.

Effective testing risk management techniques can help to ensure that a quality product is delivered to a satisfied customer with a minimum of pain.