SURVIVING TESTING RISKS - A PRACTITIONER'S GUIDE

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.

There are a number of planning and technical risks that development projects should look out for. By understanding and acknowledging these risk areas, project and test managers can prevent problems before they occur, or at least reduce the impact of problems when they inevitably occur. Here are five risk areas to consider:

1. The expertise drought

Many projects simply do not have enough testers to adequately meet their testing need. Even where projects have enough people to assign to testing tasks, they often are not experienced 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.

Teamwork is the key to handling this problem:

- Share the testing load across the project through the use of effective developer-led techniques such as design reviews, code inspections, and unit tests
- Develop explicit guidelines on how to design and conduct tests.
- Use mentoring within the test team.
- Use metrics to monitor test progress.

2. 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.
- The validity and accuracy of the test results can not be guaranteed.
- The lack of a realistic end-user test 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 (and eventually) a realistic environment. Projects must ensure that they test what they are going to release and they release what they tested.

3. 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/prioritise development tasks.

4. 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.

5. 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.
- Assign more experienced staff to the higher priority requirements.

