TESTING IN A HURRY – GETTING YOUR PRIORITIES STRAIGHT

What do you do when you are assigned to do the testing on a project, and you realise that you don't have enough time to get the job done the way you think it should be done?

It could be that the project did not plan enough time for testing right from the very start. It could be that the schedule has slipped and so you have less time for test execution. Either way, the deadline is looming (and unmovable), and you need to do as much testing as you can in the time remaining.

To start with, you need to take a cold, hard look at the tasks that you think should be done (if you had plenty of time) and how much time you think each one will realistically take. For each task (eg, a test to run, a document to write), consider the impact of either eliminating the task completely or cutting corners and doing it less thoroughly.

The key to making sensible decisions at a time like this is to understand both the priorities of the project and the risk of doing less testing. The answer to this will vary from project to project, and it is doubtful that you will ever make the absolutely correct decision (which will be proven with hindsight). But you have to start somewhere. You have to get your priorities straight. This tHints considers two time-saving techniques.

STREAMLINE THE DOCUMENTATION PROCESS

When time is running short, one of our first tendencies is to forget about planning and documenting tests, in favour of just "getting the job done". While this may feel faster, it is an inefficient way to work - you do not know how much of the system you have tested, and it is difficult to later repeat tests for regression purposes. Without some measure of repeatability, you cannot easily make judgements about the improving (or not) stability of the system. If you cannot make this judgement, what have you really achieved as a tester?

A better approach is to keep all test documentation tasks, but to streamline them. In particular, you can greatly reduce your documentation burden if you write detailed Test Cases with well-defined objectives and clear pass/fail criteria, but then free-play the test steps. By not laboriously writing keystroke-specific test steps, you will save the up-front documentation time plus you could save potential rework in rewriting test steps to keep up with user interface changes.

If you want to capture the detailed test actions (for re-testing/re-use purposes), you can log them during testing (and update the document with the detail after the schedule panic has died down).

Be sure that you do not skip the initial Test Planning task. Skipping this task will not save you much time, but will reduce your ability to anticipate "big picture" problems that could occur later (ie, with test environments, dependencies between teams, etc).

CONCENTRATE TEST EXECUTION ON KEY FUNCTIONALITY

Despite our best attempts at planning, it is common for time to run out before we have run all of our planned tests or tested/retested all of the functionality. It is important to be realistic about this and work out a strategy for minimising the impact of a reduction in testing.

As one of your earliest test planning activities, identify the relative priorities of the functional areas and user scenarios. If this information does not already exist, talk with the developers, talk with the users, and talk with the managers, to gain a clear picture of what is most important about the system.

Some criteria for assessing functional priorities include:

- Mission criticality - without this function, the system might as well not exist.
- Scope of use - will this function be used often or only once in a while?
- Impact of failure - if this function fails, what will happen (how great will be the loss)?

- Technical and interface complexity - defects tend to cluster in complex functions.
- User needs - is this function critical to the operation of the system from the user's perspective?
- Component history - how well do we trust this 3rd party software that we are using to achieve important goals?

Use the criteria to classify your tests for each functional area and user scenario as either a High, Medium or Low priority test. The High priority areas (at least) are your key functionality tests, which should be run earliest, most often, and by the most experienced tester. These are the "must run" tests that you should never drop off when time is running short.

As an example of a typical risk-based testing scenario:

- Start out by planning to completely run all tests (regardless of priority) at least twice: first to find defects, and second on the "to be released" version of software.
- As the time available for testing becomes reduced, progressively drop off tests for the second run based on their relative priority.
- When the time gets reduced further still, start to drop off tests for the first run based on relative priority.
- As a hedge against total disaster, always try to find the time for a final run on the "to be released" version of the software with tests for at least the key functionality, the main user scenarios, and the tests for functionality that has been unstable (ie, has had a high defect rate).

Every time you decide to reduce the amount of testing, ensure that you communicate this, and the resulting risk to the quality of the system, to all stakeholders. Remember that risk-based "Testing in a Hurry" is fine up to a point, and then the risk becomes too great and the result is a shambles.