tHints: The devil's in the details — design your tests before implementing them

By Donna O'Neill

Testers are often critical of developers for not working out the system detailed design before they start coding. We've all seen the tell-tale signs:

- The early drops implement all the "easy" bits, not the difficult functionality;
- You get software that implements some requirements, but not others that you thought were closely inter-related; and
- Successive software builds show major changes in functionality and user interface – even in areas that you thought were stable.

But, I have a sneaking suspicion that many testers are guilty of doing the same thing – they jump straight into writing detailed test steps before they have worked out their test design. The most obvious clue to this is missing test cases – the tests contain detailed actions and expected results, but there is no explanation of the logic of the tests (that is, what they are trying to prove or uncover).

Why are test cases important?

These missing test cases serve an important purpose in the thought process that goes into testing software efficiently and effectively. If you think of your test plan as being the equivalent of the developer's system architecture, and your test steps as being the equivalent of the developer's code, then the test cases are the equivalent of the in-between stage of detailed design. As for the developers, it is important that testers do not skip this stage because it gives us:

- A firm test structure that lets us be certain that we are covering all requirements;
- A good understanding of when to test related requirements together; and
- Confidence that we can work efficiently, without too much rework when changes are made (to requirements, to the user interface, or to our tests as we learn more about the application).

Skipping this thought process and diving straight into the detailed test steps has a number of adverse consequences:

- *You get bogged down* – You get side-tracked by the implementation details and lose sight of what you are trying to prove in the test.
- *You can't see the forest for the trees* – When trying to update your tests (or update someone else's tests), it is difficult to tell what the button-press instructions are trying to achieve and what the expected results are meant to prove, and therefore how to make the required changes.
- *You are trying to hit a moving target* – To write test steps you need a known and stable user interface, but as the user interface often changes throughout development you invariably end up having to rework the test steps to keep pace.

These consequences result in an inefficient and ineffective use of your testing time. Too much detail without enough structure conspires to keep you busy with test documentation instead of letting you get on with the job of testing the software and providing feedback to the project.

Getting the most from the test design process

A common reason that testers give for skipping the test case design stage is because “it feels slower” than writing test steps. It is a common reason given for rushing through planning tasks in general – we just want to get on with what we perceive as the real work (ie, the implementation). For developers it is coding, for testers it is test steps.

However, the opposite is actually true – a good set of test cases will help you to fast-track the testing, while helping to ensure that the testing is as effective as it can be.

The key is to *design early and implement late*. That is, as soon as you have a view of the system functionality and architecture, before you have the software, you can start thinking about:

- What you need to test;
- What your priorities are; and
- Where the defects might be.

Armed with the requirements, you can design a set of “positive” test cases to demonstrate that the software works as expected. You can also design a set of “negative” test cases to try to break the software, by setting up invalid situations.

We write test cases at a relatively high level of detail, with wording like: “This test case demonstrates that the user can retrieve and display an existing entry in the database.” Similarly, you can design a negative test case for this requirement: “This test case demonstrates that an appropriate error message is displayed when the user attempts to retrieve a non-existing database entry.”

Designing tests at this level of detail helps us to test earlier and have greater confidence in test coverage:

- *Test earlier*:
    - The wording reflects what the functionality is doing, but because it does not rely on the implementation, it can be written as soon as the requirement has been defined.
    - It is far quicker to write one liner test cases than to elaborate detailed test steps
    - You can quickly freeplay the test actions from the cases (to provide early feedback), and then when it seems that the software is stablising, you can write down your detailed test steps (avoiding unnecessary rework)
- *Test coverage*:
    - It is much easier to tell if your set of tests covers all of the requirements
    - It is easier to judge if you have tried enough negative tests to uncover invalid situations

Because you are avoiding the detail for as long as possible, you can more easily:

- Cope with fluid projects where the implementation keeps changing
- Cope with short deadlines and shrinking timeframes
- Facilitate good review of test coverage.

In conclusion

The key to efficient and effective testing is to concentrate on your test design and avoid the devilish details for as long as possible:

- As soon as you have a general idea of what the system is and how it will be implemented, you can start your test planning.
- As soon as you have a reasonably firm idea of the functional specifics, you can start writing test cases
- With early software drops, just freeplay from the test steps
- Try to hold off writing detailed test steps until you have software, and only detail the steps for a given test when that functional area has stabilised.