## FILLING THE REQUIREMENTS GAP – USE GENERIC TEST CASES

One of the trickiest tasks facing software testers is to identify the full extent of the tests that they should do. They are responsible for fully testing the system, however testing against the requirement specification alone is often not sufficient to design effective tests that ensure that all functionality has been correctly implemented and that the integrated system is fit for its intended purpose.

### REQUIREMENTS ARE NOT EVERYTHING

Most software development projects understand that they need to define their product requirements. However, it is extremely common for the requirements to be poorly written and incomplete. Requirements can be "missing" from a specification for a number of reasons:

- They are hard to define and quantify. These include issues such as reliability, availability, and user friendliness.

- They relate to basic system behaviour and failure. These are the "unspoken requirements" that are perceived as so rudimentary that the author does not think to include them in the specification. This would include functionality such as conformance to user interface conventions.

- Uncodified knowledge. This is the functionality that the developers know from experience must be included in the system, however they do not feel that it is an effective use of their time to document. A common example of this is where a series of products are defined by means of "deltas" from a previous core product, but where the original product was never properly specified.

The commonality between these types of "missing" requirements is that they tend to relate to typical patterns of behaviour that are common to systems of a particular type. They are seen as "generic" issues and so are given less focus when writing the specification.

To design effective tests, testers also need to be aware of a number of technical issues associated with the system under test, such as common failure modes for applications of this type, design constraints for the system, and industry norms. The tests should be designed to exercise the functionality with these issues in mind.

### UNCOVER THE FOLKLORE

As a tester, it is (theoretically) not your job to gather the product requirements. In reality, however, most testers will admit to spending a considerable amount of time doing so. We gather the requirements from a variety of sources, in the hope that we will gain sufficient insight into the functionality and system operation to produce good tests. To be effective testers, we need to understand the technical issues and common patterns of behaviour and failure associated with the implementation of systems of the type under development.

One of the best ways to uncover this information is to draw upon the domain expertise of the subject matter experts who can be found in most organisations. These "gurus" are the keepers of the folklore, who have extensive product knowledge, having lived through previous development iterations or the development of similar products. They have seen the ways in which similar products have failed during development and in the field, and they have first-hand knowledge of the "gotchas" that lie within.

### BUILD A REUSABLE TEST KNOWLEDGEBASE

The information you collect will lead you to an understanding of the general operation of the system and the ways in which it might fail. Much of it will probably also be applicable to other similar systems and application types.

  

Therefore, having made the effort to capture this information, you should make sure to document it in a reusable form. By creating a knowledgebase of test design patterns, you will gain two main benefits:

- Increase the effectiveness of testers who may have less subject matter expertise
- Speed up the test design process for subsequent similar projects.

To document this knowledge, create Test Design Checklist for each type of test that is applicable to your system. Examples of test types include User Interface Tests, Load Tests, Reliability/robustness Tests, Data Migration Tests, Interface Tests, etc.

For each Test Design Checklist, define the *Approach* to doing the test and the set of generic *Test Cases*:

- The Approach section should identify:
  - When the test type is applicable
  - The rationale for doing the test
  - Common environment issues and pitfalls associated with the test.

- The Test Cases section should be a series of short objectives that represent a grab-bag of issues to consider, places to look, data sets to use, error conditions to try, etc when testing the application.

For any given application, not all test cases in the checklists will apply. Pick and choose the ones that do, and also update the checklist regularly with new test cases as you discover them.

If the checklists are well written enough, and time is running short, it may be sufficient to simply use them as the basis for directing systematic free-play testing. They are then, effectively, a set of generic test cases. Alternatively, they can be used as an input to creating detailed (and documented) test procedures. They also can be used for directing technical reviews of the test procedures.

Once developed, the generic test cases will form a valuable part of a tester's toolkit of strategies and techniques. They can help to bridge the gap that is created by missing requirements, as well as enable testers to effectively target potential problem areas. Ideally, the issues captured in the test cases should be incorporated into the requirement specification as much as possible. However, this testing knowledgebase will mean that the testers are less dependent on the development "gurus" for technical input if this does not occur.

  