SYSTEMATIC FREEPLAY TESTING

What do you do if you are a software tester, responsible for testing the functionality of a system, and there are no defined requirements? Or if there *are* requirements, they are vague and incomplete? The answer is Systematic Freeplay Testing.

Freeplay testing is a technique by which testers explore the system's actual functionality, while at the same time determining whether it shows any anomalous behaviour. Typically, the focus is on whether the system "does what it does correctly", rather than it "does what it is supposed to do".

The nature of these tests is that they are typically worked out "on the fly" rather than being run from pre-defined and detailed test procedures. For this reason, freeplay testing is commonly used on fast-turnaround projects where there is not sufficient time to develop detailed test specifications in advance of receiving the software.

On the plus side, freeplay testing can be done very quickly and provides at least some measure of confidence in the system's general behaviour. On the negative side, if not done correctly, it will be unsystematic, the tests will not be repeatable, and the tests will probably not find missing or incorrect functionality.

However, by applying a systematic approach to freeplay testing, it is possible to gain most of its benefits, while avoiding most of its shortcomings.

BEING SYSTEMATIC

The first step to systematic freeplay testing is to identify your test objectives. That is, you should identify what tests you will run.

Even without well-defined requirements, there is a large amount of information (ie, implied requirements) available to guide the resourceful tester. To uncover this information, you need to *explore*:

- *What is the system supposed to do?* Read user manuals, marketing brochures, and design/architecture documentation. Talk with developers and users. What you find out here is the closest thing you will have to real requirements, against which you can compare the actual system operation.

- *What does the system actually do?* Use the system to identify basic system functionality and behaviour. Identify the menu hierarchy, input fields and dialog boxes, and use on-line help. By understanding what the developers tried to do, you are in a better position to identify where they might have gone wrong.

- *What do I (or other testers in my company) know from testing similar systems before?* Recall what those systems were required to do and how they were tested. Also identify whether there were any industry standards that they were required to meet. If you are lucky you may be able to reuse tests from these other systems.

Having formed your test objectives, the next step is to identify (and run) a set of test cases. Again, even though you are doing freeplay testing, you should try to be systematic. Try to prioritise the test cases to ensure that you have time to run the most important tests (ie, don't waste time with extraneous "gold-plating"). Test cases to consider include:

- Functional tests:
  - Tests of all menu paths and input fields
  - Does the functionality come in complete subsets? For example, if you can create a record, can you also modify and delete one? If not, why not?
  - Does the system do all the things other similar systems did? If not, why not?
  - What functionality problems have you seen in similar systems?
  - Error guessing - what might the developers might have done wrong?

- System tests:
  - Tests that follow realistic/actual user scenarios and situations
  - Tests of the end-to-end operability of the complete system
  - Tests of the interfaces between different parts of the system (particularly if your company did not write all of the software)
  - Does the software install and de-install correctly?

- Robustness tests:
  - Does the system work correctly at boundary values?
  - What happens at high stress and load levels?
  - How does the system handle exception conditions?
  - Error guessing – what might the developers might have done wrong?

- Compatibility tests:
  - Is the software supposed to run under a number of operating systems, browsers, etc?
  - What compatibility issues did you have with similar systems?

- Conformance tests:
  - Test against industry standards (formal and informal).

CAPTURING THE KNOWLEDGE

The next step to efficient freeplay testing is to capture the test objectives, test cases and observed system behaviour in a detailed test log while you are running the tests.

Rather than just report on observed discrepancies (per usual), take a bit more time and document your test objective/cases just before you run each test. The level of detail for the test objective/cases should be sufficient to enable you to reproduce the intent of the test (ie, not necessarily detailed steps) at a later date.

This documentation can either be hand-written (eg, write a Test Log, mark up a screen print) or on-line (by typing straight into a template), depending on which method is faster and more convenient. The objective is to quickly capture the test information without overly impacting the schedule. Even a fairly "quick and dirty" attempt at documentation during freeplay testing will bring benefits later on.

By capturing this information, you are progressively preparing a test specification and building a body of knowledge on how to test systems of this type. After testing is completed (and when you have more time), you can turn your quick documentation into a checklist of test cases (maybe as a collection of "one-liners") that you can use when retesting the system or when testing a similar system at a later date. By refining the checklist each time it is used, you are gradually improving the effectiveness of your freeplay testing and thus the quality of your released product.