
Software Quality Assurance: SOFT3302

Tutorial – Week 3

Objectives

This tutorial is meant to be a hands-on introduction to (structured) testing. The tutorial compares structured and unstructured testing and requires construction of test cases for an item of software.

Pework

The SUT for this assignment will be a solution to SOFT3104 Assignment 2 (available at <http://www.it.usyd.edu.au/~cs3/soft3302/resources/>). Work in small groups of 2 or 3. Use your own solution to this assignment or group with another who submitted a solution. (For the purposes of this tutorial it will not be important to be the author of the SUT.)

If you prefer you may choose a simple application instead such as gaim and analyse that.

Labwork

1. Spend about half-an-hour performing ad-hoc testing of the SUT. This will involve simple experimentation with the SUT and attempts at various actions. Record any issues discovered for discussion later in the tutorial. Keep a shared list in the group of all issues uncovered by this testing.

For all but the author in the group, the SUT is a small software package of unknown provenance. It may or may not be incomplete and may have little documentation. It is not clear how many bugs may lurk in the software. This situation can be the case even for the author responsible for writing the software. Such a fact is lamentable but sadly quite common.

Make sure students are documenting their discoveries in the shared list.

2. Discuss any perceived advantages and disadvantages of unstructured testing.

Keep this discussion short. Discussion may take place in small groups or with the tutorial class as a whole. It will be fruitful to visit this discussion again at the end of the tutorial when unstructured testing may be contrasted with structured testing.

3. Produce a functional hierarchy of the SUT.

There will be many ways of grouping functions hierarchically. A quick look should determine whether functions have been sensibly grouped. Things to avoid are too little grouping, e.g. as an extreme: one top level and everything under that, or too much, e.g. functions split into too many separate groups which should really be together.

It's not a complete analogy but a comparison may be made with breaking a programme into modules. One doesn't want the modules to be too large, encompassing different functionality, or too small, where each of multiple modules appears to be handling part of a larger set of tasks. (There may not necessarily be a direct correspondence between the modular breakdown of a programme and the functional breakdown of its interface.)

4. Choose one function from the hierarchy and develop a set of use cases for that function and a model of the data flow.

Stress the importance of uniform documentation in this step and the following steps.

5. Describe the input and outputs of the function.
6. Define tests for the function. Think carefully to ensure you are not just duplicating the same test or tests under different guises.

Templates similar to those used in the lectures should be used here. It will rapidly become apparent that (a) there are a lot of these tests; and (b) the number of these tests will mean that careful thought needs to be put into them to make sure they neither duplicate tests nor omit tests.

Students may be surprised at the number of tests needed for even a simple operation.

- a. Define criteria-based tests.
 - b. Define output-based tests.
 - c. Define input-based tests. Consider both valid and invalid inputs.
 - d. Define internal condition-based tests. The conditions may be satisfied or not satisfied.
 - e. Define state-based tests.
7. Depending on the protection afforded to the various data members of the class, how easy/hard was it to determine the outcome of each test? Do you think that testing needs to be taken into account during design? What does this say about the temporal ordering of testing and coding?

Clearly testing needs to come before coding so that the ability to test can be built into the unit under test. (In C++, for example, this may mean declaring testing classes friends of the classes under test.)

8. Revisit the discussion on the advantages and disadvantages of testing techniques, this time contrasting unstructured *versus* structured testing.

Hopefully it is clear that structured testing should come out the winner in the comparison.