Software Quality Assurance: SOFT3302

Tutorial — Week 6

Objectives

This tutorial gives experience designing unit tests using a unit-testing framework. (You are free to do this tutorial in Python, C++ or Java.) By the end of this tutorial you should be able to write a suite of test cases and have an understanding of this processes as well as experience in running the test and displaying the results using a test runner.

Prework

Review the use of unit testing frameworks from the lecture.

Consider the short specification of a class below. You will be writing this class and unit tests for this class in the lab this week.

Write a 'queue of integers' class. The constructor should take one argument: the maximum size of the queue. The default constructor should create a queue of 10 elements. The queue should have the usual push/pop semantics. An attempt to pop from an empty queue should raise an exception. An attempt to push onto a full queue should also raise an exception. The queue should have a size method that returns the number of elements in the queue and an empty method that returns true/false depending on whether or not the queue is empty/non-empty.

Labwork

Write an implementation of the above queue class using test-drive development. *Remember, this means write the tests first and then code to pass the tests.* Run the tests you develop using a text test runner and a graphical test runner if available.

At least the following test cases should be covered from a functional (black-box) perspective:

- **Creation of a new queue with the default size (the size should be checked)**
- **Creation of a new queue with a specific size**
- **A new queue should be empty**
- **A queue with something in it should not be empty**
- **A queue with everything removed should again be empty**
- ***Similar tests as to 'empty' but for 'size'***
- **Appropriate boundary tests on the queue size**
- **Attempts to overflow a queue should raise an exception**
- **Attempts to underflow a queue should raise an exception**

The following functionality also needs to be covered. Even though it is not explicitly stated in the description, it is an implicit property of the *queue* data type.

- **Items inserted into the queue should be returned in FIFO order**

This covers the risk that the queue is actually not a queue but something else such as a stack or just returns values in a random order such as a set.

As well as these cases white-box testing should give rise to more cases including the following:

- **If the queue is implemented on top of an array-like structure—not an uncommon implementation for fixed-length queues—boundary tests when the queue wraps around the array and when the queue straddles the end/start of the array**
- **Similarly, the implementation needs to distinguish between a full and an empty queue (as the head/tail indices may have the same values under these circumstances)**
- **Statement, branch and appropriate path coverage tests**

If the queue class is implemented as a wrapper/extension of another class, such as a deque, which presumably is 'tested' and is low-risk/high confidence, is there a need to test the new class?

Yes! Remember, any line of code, any branch should be considered to have undefined behaviour until tested.

If subclassing is involved then substitutability tests need to be implemented. If such functionality is not appropriate, *i.e.* it makes no sense for a *queue* to be considered an *array* or a *deque* then exposing the subclassing is not appropriate. Either private inheritance should be used in a language such as C++ where this is possible or delegation should be used instead of inheritance.