FIT4004 - A3: Test Analysis

Lawrence MacDonald (26020750)

Bugs found

Our unit tests did find a few bugs. Very rarely did they detect algorithmic issues, such as incorrect values being generated, but they did find quite a few small errors in syntax or handling that weren't detected by just running the program as it was still in development. An example is a situation where an invalid boundary input should have been throwing an error, but wasn't because of a misunderstanding regarding a method in the datetime library.

Test strategy adequacy

Because our test strategy included, after more formal methods, exploratory testing, we managed to create unit tests for any situations that we thought could be relevant to test. That said, they made up a significantly small portion of the overall test suite. The black box methods provided almost all of the unit tests we ended up with, which also happened to achieve branch and statement coverage. That said, while our black box tests generated very comprehensive tests, because of the amount of information which was testable (eg. in the collection of tweets from the Twitter user's timeline, we could test the Tweet object's generation, and each of the individual values within the Tweet were correct after our processing) for a particular set of inputs (our black box methods were very input-centric), if the tests failed, they didn't give much insight regarding how exactly that happened. All we could garner from the result was that one of the things that the test tested was wrong. We then needed to debug the function manually ourselves from scratch, which wasn't too difficult because of the system's small scale, but this solution would not scale well into a complex system.

Who should write the tests

While our process resulted in the member of our team responsible for a particular unit writing that unit's tests, we did find it very useful during the black box test generation to be able to ask the other team member what they thought would make good boundary input values, simply from what they know about the unit's purpose. This prevented the test writer from adding their bias from knowledge of the code into the generation of boundary input values. That said, having the author of the function writing tests made it significantly easier to write them for more complex units, and especially ones that required mocking or patching. This was because to write effectives tests for these units efficiently, the author needs to understand what the unit will actually be doing with the inputs, what should be returned from it, and what sort of exceptions can be raised along the way. Hence, I believe that it is important for the author of the unit to write the unit’s tests, but it is important to get outside opinion for black box testing.

Mocking code

The first times we needed to mock in various ways for testing, we spent a lot of time on it. In one particularly complex case of mocking, setting the mock up and getting it to function properly took over an hour of internet research and experimentation with the mock library. Once we understood more about the process, however, mocking sped up considerably, and was taking mere minutes. In respect to the overall testing effort, mocking was not a major time consumer. The process had much higher understanding overhead than anything else encountered in the assignment, however.

Lessons for the future

I think the most important thing I would remember for the future is to plan a test strategy that will result in more granular tests, output-wise. I think the input-centric tests were extremely valuable, as they allowed us to predict lifecycle of data going in, through, and out of a unit, but combining them with some more output-centric tests probably would have resulted in more useful feedback from these tests upon failure.