FIT4004 - System Validation & Verification, Quality and Standards

Assignment 3 Test Analysis Report

Student Name: Wanyu Yin
Student ID: 24141232

Introduction

The purpose for writing this report is to demonstrate what our team learned from the automated unit testing (including mock), regression testing and continuous integration as well as adopting the Test Driven Development (TDD) techniques.

What is more, the report will also discuss how our team benefited from the coverage tool.

Bugs Found With The Unit Tests

The bugs our team discovered for the unit tests are mostly functional and regression defects. There were quite a lot of bugs in the function *fill_template()*, *translate_scalar()* and *translate_loop()*.

The reason why there were many bugs for *translate_scalar()* and *translate_loop()* is quite straightforward. It is because our team were not very familiar with what scalar and loop were at the beginning of the coding stage, we needed to test and fix the defects of the functions many times. However, we gained more understanding during the process.

For *fill_template()*, the function is quite complicating comparing with the other functions in our program, and this function contains several other functions, which made it very easy to go wrong when the other functions had defects. But since we were following the TDD, we made sure all the other functions' behaviors were right, and by doing this, we ensured that the defects occurred in this function was only resulted from the function itself.

Coverage Tool

Our team found the coverage tool of great use to assess the adequacy for our testing. Because the coverage tool can actually analyze and produce the result for which part of the code has been tested. Our team was originally aiming at reaching 90% or above for the code coverage. And after running the tool *coverage.py*, our testing was 100% code coverage according to the results.

Before reaching 100% code coverage, the coverage tool was really helpful in identifying which part of the code that had not been executed by the tests. In the report generated by *coverage.py*, it marked the un-tested part in red to inform us to either add tests for this part of the code or check if we had duplicate test names (the coverage tool will ignore the tests which have the same name).

Who Is Better To Write The Tests

I would highly recommend the one who writes the code to write the tests. If a person is experienced in programming, and there are a huge amount of time for a person to read

through and understand the requirements before writing the tests, it is ok to let someone else other than the programmer himself to write the tests. But this ideal situation will not always exist in the real world.

Another main and the most important reason is that, because TDD requires quite heavy code implementation while writing tests, it is much more efficient for the developer to write the tests. The developer will have more concrete understanding of the code, and if he is also in charge of testing, he does not need to wait for the testers to run and test the code before moving on to the rest of the implementation.

Mock

It took our team approximately 50% of the overall time to write the mock code. Comparing with the number of the rest of the tests, there are fewer mock tests. But since we were both new to mock and the inner logic for mock was quite confusing to us at the beginning, we did spent some time playing around the mock in order to understand how it works.

The best part for mock is its magic of separating. Our team mocked out the actual server connection while using mock for testing. This significantly improved the speed of the testing by faking the situation of connecting the SMTP server without actually sending data to the server. Another best part of mock is that we could define what we wanted it to return, and therefore, this made it much easier for exception handling.

Improvements

The things need to be improved from my point of view are:

- Plan ahead that which part of the functionality needs to adopt the mock technique since we already have the experience of mock.
- Follow the rules for TDD more strictly. (Sometimes, I had to pause myself from writing the functions and go back to do the tests first. But now I am getting trained better and better. Thanks to my teammate Arvin!)
- Refactor the tests. (Not required in this assignment but I think it is a must for real industry.)

Conclusion

In conclusion, conducting test driven development and automated unit testing are very important for the software development process. I would definitely use the two techniques for the university studying as well as well as for my future industrial experience!