**Project 2 – Summary and Reflection Report**

**Summary**

Unit testing is required when writing code as it allows for the detection of errors or defects in prior to a final releasable build. This, in turn, allows the developer to quickly find and address defects.

**Describe your unit testing approach for each of the three features. To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**

I aligned the testing with software requirements by testing for features specified. For example, in ContactServiceTest, I tested for the requirements for the addition of unique string ID, of a length no longer than 10 characters. Here, it was also important to ensure that contacts could be deleted and updated based on software requirements as well.



In AppointmentTest, I specifically tested for addition of appointments to dates that were not in the past, and descriptions that were within the character limits, as required by the software.



**Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?**

The overall quality of my JUnit tests are fair. However, there are multiple tests that I need to improve on due to lower coverage percentages. While some tests are above 80%, the AppointmentSeriveTest is at 57.4%. This indicates that testing did not cover the whole code. I will need to return to this test and add additional testing parameters in order to detect deficiencies.

**Describe your experience writing the JUnit tests. How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate. How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

In order to ensure my code was technically sound, it was important to stay consistent between the class and the JUnit test cases. This required me to pay specific attention to the names of my variables, the requirements set forth by the client, and specifically test for said requirements. I had a lot of difficulty with consistency between these because I found I was changing variable names and cases quite often. As a result, I had to be more mindful about naming conventions, and used notes to remind myself of the testing requirements in order to get higher code coverage.

**Reflection**

I used the dynamic testing method in order to check the behaviors of the software. This allowed me to test the multiple variables which, by their nature, are able to change independently multiple times. The method that I did not use was static testing, which would have allowed me to test the code without execution.

I tried to limit bias by reading and rereading the testing requirements, making notes in the test classes, and implementing them. However, I imagine there are multiple test cases I have missed, simply based on the fact that I felt I already knew the code functioned as it was required to and accidentally neglected to create test cases. Due to this, I can imaging it would be prudent to work on a team where test cases are writing by developers who have not worked on a specific portion of the project.

It is extremely important not to be lax in the testing process because it is the chance to identify deficiencies before the launch of a piece of software. If a client is delivered a completed product that does not function as required, the developer has then failed to meet their needs. This may result in major errors. For example, in the case of this project, if there hadn’t been a test case involving deleting appointments, there may be multiple appointments booked for the same slot. I hope to avoid these oversights by working as a part of a team that may help me find test cases that I have neglected to consider.