**CS 320 Summary and Reflections Report**

**Summary**

The software testing required to adequately meet the requirements of the client was extensive. The three different classes that were used in the development of the software employed different types of data structures and variables to be used, so the testing that was used were created to ensure that the data was passing correctly through the functions that were defined in the classes and worked correctly. The Junit tests completed successfully on most test cases but failed on certain checks related to confused understanding of the values that were expected to be received and checked for something else. The contact service testcases required that the name not cross a limit of 30 characters and not be null, the phone number not exceed 10 digits, not be null and only contain numbers, and the ID not be more than 10 digits and be uniquely identified. This ensured that the contact service system worked correctly and stored information accordingly. The Appointment service required the use of the Date field, which could not be set in the past, and descriptions and appointment name could not cross their character limits of 50 and 10 respectively, and the appointment ID needed to have a unique ID for each appointment created. The Task Service class testing required to check for a task name limit of 20 characters, a description character limit of 50 and a task ID String limit of 10 characters, all of which could not be null, and the ID cannot be changed. I ensured that the code was technically sound as I was able to get 100% coverage for tests, when using assertTrue(equals)). I also ensured that the code was running efficiently, since the tests did not take long to finish, and improved the test cases where I could by using a different variation of the assert method, for example, by checking for a assertFalse when a certain test case was supposed to fail, instead of waiting for a testcase to set the values for it to fail.

**Reflection**

For this application software, I did not use Software Inspection as one of the testing techniques. Software inspection is used in earlier stages of SDLC, which ensures that the code is being moulded the way the product is intended to be. This technique focuses on identifying the objective of the software and then is checked for completion. In the case of this software, the requirements were laid out beforehand, then the code was distributed to the developers to be developed and tested, then submitted to the supervisor for inspection and either was put forward to being deployed or reworked to match the requirements correctly.

The other software technique I used was Unit Testing. Unit testing involves testing the code base at an early stage in the SDLC workflow and helps to test the code under an actual real-time load which helps in determining where the code goes wrong and works well. Unit testing also helps to test individual parts of the code individually, ensuring that they all are working together perfectly. Unit testing prevents rising testing burdens in later stage of development and ensures that the code is being written according to standards.

A combination of both these testing methods can be seen as useful, since unit testing involves bringing the code live and seeing how it behaves in real time, which helps in faster debugging and optimization of the code, with which paired with Software Inspection can suggest to management that the idea or code itself might have to be reworked if it is to be done on time or efficiently, which can be brought to the attention of developers earlier so they accommodate changes accordingly and will not have to make expensive changes to the already established code base.

The mindset that I had adopted during the work on this project were to have an open mind and test for how many scenarios the code could go through and happened to adapt my code to changes that I noticed while working with the testcases to see if I could improve them. I also had a bias to not check if my code was correct in the first place, since I believed it to be perfect on the first go. I also tend to ignore any failing testcases and will work around them to get the testcase to pass instead of changing the code, which is not up to standard. Being disciplined also comes with a good understanding of what makes a certain thing of good quality, and when that quality is ingrained into the minds and hands of a developer, it becomes a little difficult to let go of quality. Cutting corners can lead to a certain thing breaking, which cannot be fixed by a simple code change, but rather reworking the entire base. I plan to avoid doing so by working in small parts, testing before I move forward and then ensuring that the puzzle gets slowly completed.