**Project Two**

Summary and Reflection report

Unit testing is very important when building a program because it helps reduce the risk of running into an error when the program is up and running in production environments. To make sure that I tested everything, for each milestone, I made sure to test each classes multiple times to avoid any false positive results. I did a lot of examples of testing with classes to ensure the code doesn’t have any error. Although each milestones had its own testing, to finish I created a group test for everything in one folder. All my unit tests applied aligned with the software requirements. For example, the contact service milestone, required that the contact ID object had no longer than 10 characters, it cannot be null or duplicate. The contact FirstName and LastName couldn’t be null or duplicate, they also wanted a phone field which is a string and must be exactly 10 digits and should not be null and the contact class should also have an address field which should not be longer than 30 characters ad should not be null. All these requirements were supposed to be in the code, and for my unit test cases to pass, I had to align these requirements in the classes to test. The overall quality of my unit test was good as I tried to test each of the fields and the operation to check the validity of the test cases. My experience with Junit was quite challenging because I had a hard time using Eclipse to test my codes, however, once I understood it was nice to see how the code perform. I will be using Junit testing moving forward to help me debug my code and find any mistakes that the users might find before it is deployed into production. Graphical user interface, text, application

Description automatically generated

The code above, was using a BeforeEach function to test if the TaskID was too long, to make this code efficient, I tested for a very long task ID and a regular size task ID and the unit test passed.

Reflection

In this class, I used the Dynamic software technique to test my classes which is to test the dynamic behavior of the software code. This method helped me check the behavior of the dynamic variables which are not constant and have weak areas. The software technique that we didn’t use is the static software technique. The static and dynamic technique are very similar and different too. One of the major difference is, with the dynamic technique, It is hard to identify small defects such as inconsistencies in the software model, and development standard breaches.

Mindset, looking at this project, I was able to capture all the test cases based on the software requirements that was provided. It was quite important to appreciate the complexity and interrelationships of the code I was testing because

each of the class objects had some requirements which must be fulfilled to certify the code as

working correctly. All this requires the person to understand what each part of the code is doing.

I believe when it about testing your code, there is a lot of bias because the developer always feels confident about their code which can caused them to miss some requirements. It is important to make sure that the tester is a different person than the developer. In my job, we have a specific team for code review and quality control to avoid missing a mistake just because someone was confident about their code.

Discipline is very important when it comes to testing a code, we can’t go into short cut otherwise we will miss errors that the client will find in production. A company can lose millions of dollars just with a mistake that could have been avoided if testing was done correctly. For example, with task ID, we need to make sure that the user can update the ID if needed.