Summary and Reflections report

**To what extent was your approach aligned to the software requirements? Support your claims with specific evidence**

My approach to the project was prioritizing the functional requirements of the assignment. For example, in the contact service project, one of the requirements required that we implement a function for adding a contact. This gave me a clear guide to creating a unit test, so I created a test that made sure that a contact was added properly by comparing the output to the expected results.

**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?**

With each new project that I worked on, I endeavored to create more quality tests. Using the test coverage casts a wide net over all your code but it is only the beginning. Test coverage is not the main metric where exit criteria should be based upon, but it gives us a good understanding of the extent to which our code is being exercised. You want to exercise every line of code including branches of conditionals. Chances are you might find some dead ends or redundancies in your code.

**How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate**

Code is already confusing enough so to better understand you must strive for simple code that promotes best practices. When learning a new program, one of the first places I go to is the program’s documentation page. For me to write sound code I must first understand what the limitations of a tool are to employ the proper logic. These pages show the best practices used in the syntax and the features and methods that it offers. My previous experience with code has taught me how to employ best practices. The program structure was created with simplicity in mind, thus utilizing the standard program template using OOP as the basis. Using best naming conventions helped me make sure that the program was easy to follow. Using simple and straightforward variable names can make following your code easier. For example, in my test files I made sure to properly name test variables to differentiate them from the regular program variables. Using “appointment” as an object name in the main program and using “testAppointment” in the test program.

**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate**

In my opinion, efficiency means writing the same code with the least number of lines. The setup methods in my appointment project allowed me to write fewer lines but were just as effective as previous projects. Setup methods allow you to reuse code before executing every test instead of writing the same block of code in each test to initialize instances and variables. In my setup methods, I created the instance for appointment and executed the test and it reinitialized again to start a fresh test for the next method. You typically want to start a fresh slate before every test to isolate it in case there is an issue allowing you to better pinpoint the error.

**What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.**

The main testing technique that I utilized was the use of unit tests. These mini tests are helpful as they help you test only one feature at a time which can help with debugging. Once you know that the individual method is working properly then you can begin to integrate it into your main system. With the features of the Junit framework, creating a unit test was simple and effective.

**What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.**

Some testing techniques that were not used were integration testing and exhaustion testing. Starting with integration testing, this testing technique is typically done after unit testing and before system testing. Integration testing is used to ensure that individual components can talk to each other properly. This technique is variably different from unit testing as it requires more planning and requires a broader view of the system. Exhaustive testing is typically used to test many use cases in the system. These test cases cover a wide range of input combinations from the user that can be used either by accident or maliciously to initiate an attack on the system.

**For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**

Unit testing can be applied in all manners of projects whether they are big or small. The bigger the program is the more unit testing will be important to the overall health of the system in the long run. System integration is best suited for bigger projects as more component interaction will need to be properly tested. Exhaustive testing can be used to meet a security requirement. This can be more demanding as the tester is not looking for what is allowed as input but what isn’t allowed which is a longer list to test for. In combination, these tests can be used together to make a secure system.