**Grand Strand Systems: Summary and Reflections Report**

**Summary:**

After reviewing the requirements for all three features (Contact Services, Task Services, and Appointment Services), I feel that my testing was very well aligned. I was able to make sure my testing was aligned by breaking down the requirements for each feature/class and making sure there was a corresponding Junit test for each requirement. An example of this would be with the test in the TaskServiceTest Junit Test Case, there was a requirement for the TaskService class to be able to update the name of the task. I was able to create a JUnit test to show the code was able to perform this requirement. Here is the section of the TaskServiceTest that shows this:

A screenshot of a computer program

Description automatically generated with low confidence

The quality of my tests is very good as evident from my coverage percentage for the tests and the code for the three features. The project has an overall coverage percentage of 93.4%, the tests have an average coverage percentage of 91.1%, and all three features has 100% coverage meaning all of the code written for these features were tested.

A picture containing text, font, number, line

Description automatically generated

When looking at how I know my code is technically sound, I made the variables within my code private so that any of the other classes in the project could not affect the variables. An example of how I did this can be seen in my Contact class:

A picture containing text, font, screenshot

Description automatically generated

The previous lines of code is also a good example of how I ensured my code was efficient as well as being technically sound. If the requirements for any of these features changes such as allowing a first or last name length up to 20 characters instead of 10, you would not need to search through all the lines of code to change this. All that would need to be done is change the variable in this section and it would cover the whole class.

**Reflection:**

When working on the project for Grand Strand Systems there were a couple techniques that I used when testing the three features. The techniques that I used are 2 of what are known as Black Box Testing Techniques. They are Equivalence Partitioning and Boundary Value Analysis. Just like other Black Box Testing Techniques, these two techniques are derived from the specifications. They test the desired outcome of the program, and not how the system got to that desired outcome. Equivalence Partitioning is based on the assumption that inputs to the program can in most cases be grouped into like inputs. For my testing, I used this in testing the Contact class when testing the length of the phone number and making sure it is 10 digits only. When testing this I only used one 10 digit number to represent any 10 digit number. Also, when testing for invalid phone numbers I only used one phone number with less than 10 digits and one phone number with more than 10 digits. The assumption is that it would throw exceptions for any number less than or more than 10 digits because these two numbers rejected correctly. This example is also good to show how I used Boundary Testing. Boundary Testing, you would use numbers right above and below the “acceptable” input. In this case since the only acceptable input would be 10, the best boundary testing would be done with 9 and 11 digit numbers to make sure it does not permit these as valid entries.

There were several other Black Box Testing Techniques that I chose not to use as the techniques had no relevance to the program. The techniques I omitted were Decision Table Testing, State Transition Testing and Use Case Testing. Decision Tables are used to test under what condition each function would be used. State Transitioning Testing is used to see how the behaviors and outcomes change when the input changes. Use Case Testing is used to show how a certain user interacts with the system.

Each of the different Black Box Testing Techniques mentioned above have a use within testing programs. Which technique(s) are used really is determined by the functionality of the program and how it was written. The three mentioned above that I did not use had no relevance to testing the program. In the program for Grand Strand Systems I did not use any type of decision-making in the code to have the need to use Decision Tables. I also did not have transitions to test or program behavior based on user type, so State Transition and Use Case Testing were not relevant to be used as a testing technique for this program.

The mindset going into this project was fairly confident as the different parts were already there from building them individually. I also felt confident when building them as I feel confident in my Java programming, and setting up JUnit tests were not that difficult. The bias I have because of my confidence in my abilities would lean towards not being as critical of the code. For this reason, it is definitely good to have peer review if possible, as most likely defects and bugs may be missed that someone else may find. It is very important to keep committed to quality when it comes to writing code and testing. If you cut corners when writing code or testing, it could lead to errors or bugs making it through to production environments. To help prevent that and a build up of technical debt, I plan to make sure best practices are followed, as well as any set policies from the company I work for.

Reference:

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). Software testing : An istqb-bcs certified tester foundation guide - 4th edition. BCS Learning & Development Limited.