Amber Foster

February 20, 2022

7-2 Project Two

**SUMMARY AND REFLECTIONS**

**SUMMARY**

The first project was broken into three different sections that focused on one part of each section from the overall application. The three different sections were called Contact Service, Task Service, and Appointment Service. Each of the three sections had several requirements that needed to be verified using the JUnit tests. As for the unit testing approach that aligned to the software requirements needed to test each required method that were implemented in the base code. Each of the three classes create an object from the string variables to test and ensure that each variable meet their requirements. Contact class, requirements that the contact id cannot be longer than 10 characters. The ContactTest JUnit implemented the method “void testAdd()” to test the input to see if the contact id is greater than 10 characters. Task class, requirements that the task id cannot be longer than 10 characters. The TaskTest JUnit implemented the method “void testTaskId()” to test the input to see if the task id is greater than 10 characters. Appointment class, requirements that the appointment id cannot be longer than 10 characters.

The AppointmentTest JUnit implemented the method “void testSetUp ()” to test the input to see if the appointment id is greater than 10 characters. The overall quality of the JUnit tests is improving effectively when finding the errors. The JUnit tests were effective based on the coverage percentage and the overall project was at 89.4% coverage percentage. The contact test was a little higher than the task test and appointment test. All three service tests functions had a higher coverage percentage from the rest and had a positive percentage that covered the test functions. Ensuring my code was technically sound, by using data structures and objects to help check the requirements. For the contact, I added “public String toString()” to check if the two contacts are the same or not. For the contact service, I added “contacts.add(contact)” to see if the contact was added successfully or not. For the task, I added “public void setTaskId(String tID)” to check if the task id is less than 10 characters and not a null. For the appointment, I added “public void updateAppointmentId(String aID)” to check if the appointment id is less than 10 characters and not a null. Ensuring my code was efficiency, by used multiple measures to ensure that the code was efficient and testing each class for their requirements. By using the provided resources in the modules help fix different errors in the code. As for contact service test, I tested correct contact input that tests for valid updated inputs. “assertEquals(true, cs.updateContact("T003", "JoeyFirst", "DoeLast", "123", "TX ST"));”. In contact service test, I tested incorrect contact input that tests for invalid update inputs. “assertEquals(false, cs.updateContact("T004", "JoeyFirst", "DoeLast", "123", "TX ST"));”.

**REFLECTION**

The software testing techniques that I used in this project would be under the white and black box testing categories. White box testing is great for unit testing and black box testing is great for testing specific functions in the application. Fundamental testing methods in software testing are in both white and black box testing. The other software testing techniques that I did not use in this project is the experience-based techniques. The specific techniques include error guessing and exploratory testing which was not use in this project. The techniques described above include the black box techniques and white box techniques. The black box techniques are used for outsourced testing and to find the gaps that are mostly in functionality, usability, and features. The white box techniques are used to verify and test the flow of inputs and outputs throughout the application. The mindset that I adopted working on this project was logical, solving, and growth. As the software tester, I employed caution through testing and several trial and error from implementing test runs. It is important to appreciate the complexity and interrelationships of the code from how much it effects the performance and quality of the product and prevention of possible viruses. By implementing additional tests, this will help cover a lot more code, and the coverage percentage will raise higher. Thus, ensuring the technically sound, and overall product of the code preforming at high quality. The way to limit bias in my review of the code is by testing everything numerous times, review fewer lines of code before testing, and take my time when reviewing and testing my code. For the software developer side, I tested the function and inputs helped to limit bias. Testing functions help check the length of the characters. Testing the valid and invalid inputs will help limit the bias in the code. Evaluate the importance of being disciplined in the commitment to quality as a software engineering professional by not cutting corners and avoiding the performance and quality of the product when completed. I plan to avoid technical debt as a practitioner in the field is by implementing the agile development techniques and testing the code often which will help ease technical debt when using agile approach.

**REFERENCES**

García, Boni. (2017). *Mastering software testing with JUnit 5*. Packt Publishing.

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015).

*Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition). The Test Development Process (K3). BCS The Chartered Institute for IT.*