Project Two

Omar Al Yousuf

CS 320 Software Test Automation & QA

Southern New Hampshire University

August 10, 2021

**Summary and Reflection Report**

JUnit testing ensures a program is performing its tasks and requirements without any errors or defects by writing and running repeatable automated tests. I approached the three features with the same technique to align them with the software requirements. I have made a list of all the software requirements requested by documentation for each assignment when creating the codes for contact service, task service, and appointment service. I ensured that each requirement was addressed in all three programs by paying close attention to all details. An example in the “Appointment” class code from the Appointment Service assignment, required the appointment date to not be entered empty or in the past.

Text

Description automatically generated

According to the Junit coverage in Eclipse, my JUnits tests were very effective and almost completely covered all the methods in each class. For example, the “ContactTest” covered 100% of the “Contact” class code and “ContactServiceTest” covered also 100% of the “ContactService” class code. This shows that these test cases checked all the requirements requested by running many tests through each of these lines of codes and I was able to confirm before running the program.

A screenshot of a computer

Description automatically generated with medium confidence

My experience with JUnit was a little challenging at first until I practiced more and more of it and started to understand how it works and what I needed to do in order to cover %100 of my classes if possible. I ensured my code was technically sound by organizing the code with spacing and aligning, as well as added error handling in case the user inputs any unaccepted data that might crash the program or result in a complication. An example below illustrates some of those techniques from the “Appointment” class:

Text

Description automatically generated

To ensure my code was efficient, I made sure that my code does what it meant to do and what was requested by the client by utilizing JUnit test cases for each of the classes created for all the three different services of the mobile application. One example is in the “TaskServiceTest” testing and ensuring a new description added is not too long or empty.

Text

Description automatically generated

I have employed specification-based and structural-based software testing techniques such as a black and white box. According to our course textbook, the black box technique tests the system without knowing its internal workings (Hambling et al., 2015). The white box technique, on the other hand, makes the code visible to testers. Knowing the internal workings is ideal in this technique which is what differentiates it from the black box one.

One other software testing technique that I did not implement for this mobile application is the experience-based technique. This technique depends heavily on the experience of both testers and users. This testing technique includes error guessing and exploratory testing (Hambling et al., 2015).

The black box technique is generally used for outsourced code testing and it's ideal for higher-level system testing. The white box technique is utilized when the testers have a complete understanding of the software, and it is usually used for lower-level testing such as unit testing. The experience-based technique is used when “there is no adequate specification from which to derive specification-based test cases or no time to run the full structured set of tests” (Hambling et al., 2015, p. 126).

I have adopted an experimental mindset throughout this project, by applying what I have learned through this course resources and videos and tried to implement and test with caution. I believe a complex code could raise some unnecessary issues and defects. It is important to make the code organized and simple to allow testers and developers to understand what the code is doing and makes it easier to find any errors and fix them. This is when good comments and annotation within the code come to benefit the testers when it comes to creating test cases for each function of the program. My codes for the programs in project one illustrated good annotation and simple coding and it was easy enough for me to know where an error was and proceed to fix it.

To avoid bias when coding a program, I think it is essential to have different developers or testers to code and fix other than having one individual code and test their own. I believe if I was to code and test my code, I would think my code is complete and clear from any defects. However, having another set of eyes to look over the code and test it might see or discover errors and defects that I could not see unintendedly. In this course project, I ensured that my code met all the requirements requested by the client, ensured the code was properly written and tested the different functions accordingly. For instance, requirements such as those for name, ID, and description.

It is important to be disciplined as a software engineer and avoid any corners cut to ensure that we have a high-quality program that meets the client’s requirements. As a software engineer it is essential to act professionally and in a good manner to best serve the clients and employer. In order to avoid technical debt, it is essential to apply Agile methodology thought the development of the project to ensure testing is done from the beginning and throughout the development cycle and keep constant communication between developers and clients.

**Reference**

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition) - 4.5 Specification-Based (Black-Box) Techniques. BCS The Chartered Institute for IT. Retrieved from <https://app.knovel.com/hotlink/pdf/id:kt00UC2J24/software-testing-> an-istqb/specification-based-black