Christopher Sharp

CS: 320 – Software Test Automation & QA

Dr. Kyle Morton

October 11, 2023

Summary and Reflections Report

Throughout creating the software and its components, I took on the developer's role to create the code that agreed with the specified requirements. I first read the requirements multiple times to understand what each piece of the code was doing. Then, I set about creating the code that met those requirements and tested the created code using Junit. With these tests, I ran code against all the main classes' components and services. For example, I created a Junit test for the Contact class to ensure the error handling worked correctly when invalid data was entered.

A black and white text

Description automatically generated

A screenshot of a computer code

Description automatically generated

Similar tests were performed throughout the modules to ensure the error handling worked correctly. Also, to ensure that the testing was technically sound and efficient, I split up the tests to test the individual components of the modules.

A screenshot of a computer code

Description automatically generated

A second example could be pulled from the ContactService module. In this instance, the contact shall have a unique identifier and then be able to be removed with the provided identifier. Another test was created to ensure this occurred and did not produce any errors.

A screen shot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

Though one hundred percent coverage could be achieved by creating additional tests, I chose not to do that because testing for that type of coverage could cause the introduction of redundant testing for some of the functions. So, instead, I focused on ensuring I could achieve at least an eighty percent coverage rate.

A screenshot of a graph

Description automatically generated

If we break that coverage down to the individual module level, we can see that all modules scored at least eighty-five percent, with the Appointment module scoring the lowest. This was caused by not knowing how to test the Date functions in Java.

A screenshot of a data report

Description automatically generated

The software testing techniques I employed for the assignments throughout the class are twofold. Firstly, I used static testing methods by reviewing the code base after it was created to ensure there were no typos or missing characters, thus also checking over the code's logic to ensure no glaringly apparent mistakes were present. This also allowed for the checking of the requirements. Secondly, I used a dynamic testing strategy to test the functionality of the code base. This allowed me to create test cases using the Junit module. This allowed me to check the functionality of the code base to ensure that the functionality was correct and to catch any potential errors in the logic of the code. These techniques, though a small portion of many larger and indebted techniques, create a great starting point for any development that may be brought forth in the future. Further, different methods could have possibly been employed, but I did not pursue them in this case. My mindset while working on this project differed from what I have employed in many of my other classes. For example, I had to think about how the various elements of the code base would work together to create the future functionality of our application. I see where bias could play a role when testing the same code you created. For example, it can be effortless to overlook possible edge-case scenarios that could cause havoc on the application and cause issues in the functionality of the application. Finally, being disciplined when pursuing quality code creation is of paramount importance. The way I look at it, if you were to cut corners or be careless, it could cause severe issues for yourself or the company where you are employed.