Sean Toon

CS- 320 Software Test Automation & QA 23EW1

10/13/23

Project Two – Summary and Reflection

Now that all the three features of the mobile application have been finished, upon reflection I realize that my approaches for meeting software requirements and developing Junit tests for each feature did not change, but the technical aspects of my code did because of feedback from an experienced individual. From the start, I felt that I had developed a simple yet effective plan to develop good quality code and tests.

One of the first and most important things to understand at the start of the process of developing code is the system's requirements. It is important to plan and map out a rough draft or pseudocode to outline what the final code will look like, and that is what I did. The requirements were not as complex as some other projects may be, so it was an easy task to bullet out all the requirements in just a few points. After reading and truly understanding the requirements I began to start coding. I used the pseudocode and bullet points of the requirements that I had developed previously as a source to guide me in development. My process of developing was to write some code that covered one of the requirements and then I would check that off my list and I did that until all of them were met. For example, in the first feature of the mobile application "Contact Service" the first requirement was that if an Id was to be created in the app, then it needed to be unique and it could not be greater than ten characters in length. In my class constructor I wrote code that would create the Id but if an Id was tried to be created outside of the requirements, an error would be thrown. I used this process for all the input that would expectedly be processed in the app and that had requirements.

For the most part I feel that the quality of my Junit tests for all three of the features was good. The quality of my tests improved the further that I progressed with the application. For each file, I achieved my goal of at least eighty percent test coverage. Some of the files even reached a coverage percentage of one hundred percent. I feel that coverage of at least eighty percent was a reasonable goal to ensure quality testing and when I achieved one hundred percent coverage for most of my test, I knew that I was producing effective testing.

I coded each of the three features of the mobile application "Contact Service", "Task Service" and "Appointment Service" separately. The first feature that I developed was "Contact Service" and I designed the code and test's to the best of my ability to ensure that the code was technically sound. I decided the feature of the product was finished when all my code was executed as expected, and all the Junit tests ran correctly. After each feature was completed, I was able to have my product reviewed and I used help from a better experienced developer to make changes to improve the quality and efficiency of my code logic. Each time that I completed a feature of the application I was able to use feedback to make the next feature better. For example, in the first feature that I developed "Contact Service", I made the Id updatable by creating a setter for the Id. If I would not have gotten feedback about the flaw in my code logic, I would have possibly created a setter for each feature, and each feature would not have met the project requirements. The flaw also proved a flaw in my testing technique. If I had written a test to show that the Id cannot be updated, I would have been able to find the logic error myself.

I used many different software testing techniques when developing good quality reliable tests for each feature of the mobile application. One technique that I used is called Unit testing.

Unit testing is testing individual parts of code separately to make sure that specific part of the code is producing or functioning the way that it is supposed to. An example of Unit testing that I wrote in the “ApointmentServiceTest” Junit test is a test method named “testCheckValues”. The method verifies that every “Appointment” object that is created or is theoretically input from the user is correctly stored by its Id and description considering all value requirements.

Another type of testing that I used when developing tests is Integration testing. Integration testing is a method of testing used to prove that a part of a specific part of code interacts and functions correctly with all other different components of the software. An example of integration testing that I used for the feature “Appointment Service” is in the test “AppointmentServiceTest” named “testAddDuplicateAppoinment”. For the test, I created a new appointment using the “Appointment” class and then tried to add that same appointment a second time and if written correctly the test does exactly what the name implies; it throws an error if a duplicate appointment is added to the appointment list per Id.

There are some software testing techniques that I could have used to better ensure the quality of the mobile application but felt that they weren’t necessary considering the size and complexity of the project. One technique that I didn’t use is stress testing. Stress, or performance testing is testing the process of creating tests that assess a software’s speed, responsiveness, and capabilities to handle large amounts of data. While tests to assess the mobile applications speed and responsiveness may be practical to ensure the quality of the user experience, something like a stress test is less practical because large amounts of data that would corrupt a software’s function is unlikely to be used in this application.

During the process of developing the code and tests for the mobile application, I’ve kept a careful, and precise intention of developing a high-quality product. Although the complexity of the project was not as great as some, the importance of creating and developing with caution and using best coding practices was crucial. When testing it is always important to have the safety of the performance of the product in mind, as well as the safety of the users who will be using the product. One example of testing that was not implemented in the testing that I developed and could be useful and necessary for a product such as a mobile application that deals with data like contact, task, and appointment information could be security and privacy tests. Such information could be private and need protection.

It is important to eliminate bias when designing and testing code. Having bias when examining code tests can lead to a missed test or over confidence in the quality and performance of code. To eliminate bias when creating my tests, I pretended like I was a user using an application for the first time and I had no idea about how any of the code operated on the backend. I imagined every possible type of input or interaction that I feel could happen with the application and determined if there was a test that could be written for it. Lastly, upon completion of each feature of the application I submitted what I completed for review and was open to and used every bit of feedback that I was given. It is helpful to have another experienced person to evaluate your work to eliminate any kind of personal bias that would affect the quality of testing. A lot can be overlooked, or it can be easy to make simple or big mistakes when developing a large amount of code and tests.

As move forward in my career as a computer scientist I learn more every day about how important it is to be precise and detail oriented when developing code. Developing code is truly an art that leaves no room for error. Cutting corners or being careless when developing can and usually does have negative consequences that can even be detrimental depending on the product that is being created. Even if I am writing software for a harmless application or for something as serious as a military operation, I will always develop software with the same careful intention.