Roderick Fisher

Project 2

I used Unit tests to test the boundaries of the specifications that were given to me. For example, the specification of the Task says that the full name cannot be null or greater than 20 characters. My approach aligned to the software requirements very closely as I tried to test all boundaries that were given. I can confidently defend the overall quality of my JUnit test by the fact that my coverage numbers were over 80 percent on all test. To ensure that my code was technically sound I made sure to write industry standard code that is tested using correct inputs and incorrect inputs. For example, in the AppointmentService class I was sure to write industry standard code that could easily be tested such as,

if(id == null || id.length() > 10) {

throw new IllegalArgumentException("Invalid ID”);

This line of code ensures that the ID cannot be null and the length of the ID can not me greater than 10 characters. I also realized that throwing an exception was not enough to properly test the code. I also had to ensure that the correct exception was caught by the program. There where times when I was testing the code and realized that my expected error was not the error that the program gave me, so I had to ensure to incorporate code that would not throw out false errors. To ensure that my code is efficient I made sure to eliminate unessential operations, avoided declaring unnecessary variables, and properly use if statements to ensure that my code did not have to run more than it needed to.

The software testing techniques that I employed for each milestones were static and unit testing. With static testing, you are testing the program before code is written or ran. Unit testing involves testing individual components of the software to test.

I really didn’t use any system testing in the later modules. I also didn’t use integration testing. System testing validates the complete and fully integrated software product and since we have been dealing with a class by class bases, I have not had the change to do a system test. Integration testing is similar in which it test the entire software program.

Unit testing is practical to ensure that small issues in the program do not become larger ones. Unit testing is also valuable because you test often and can catch errors sooner. Static testing is practical because it helps you find errors in code before time is used writing the code. This ensures that you find errors early on and in turn makes fixing those errors a lot easier.

My mindset that I adopted while working on this project was that I needed to think as a software developer, but also as a hacker who was trying to breach the software or find different ways to brute force the program. It was almost like a cat and mouse game, as I wrote code I tried to break the code right after. Doing this ensured that I was able to correct any errors or any unexpected exceptions that may arise. My only bias that I had to overcome was the fact that it was my code that I was writing so I had to stop myself from thinking that everything was done correctly because Im the one that wrote it so I know it’s correct, and surprisingly that was a good bias to remove and overcome because I encountered a few times that my code or test were not implemented correctly. It is so important not to cut corners as a software developer especially when it comes to testing your program. Before taking this class I thought that testing was done by just using the program and finding errors as they come. I now know just how much goes into writing and testing code efficiently. I also learned the importance of code coverage and how the higher your coverage is the better the program will be. I plan to avoid technical debt by writing industry standard code and not cutting corners when it comes to writing my code and testing it. For example, if you have a project with low coverage the code may need to be refactored later on which take more time and resources.