Alexis Fuerte

CS-320

17 Apr 2022  
 Module 7

I made sure that my approach was aligned with the software requirements when I was writing the JUnit classes by testing for those specific requirements. A good example of this is in the appointment test class, I made sure to go down the list of requirements given to me by the customer. I had to make sure that the date I made the appointment for was in the future, so I tested it using a date in the future. I also tested writing out a description that’s less than 50 characters and made sure that the appointment ID was un-editable. These are things I did in all my testing classes. I think that my Junit tests got the job done, however I do think I could’ve run additional tests with numbers that I didn’t want to work to assert that the code was functional on all sides. I still managed to have a very high coverage percentage (92%), so I think that overall, the code still would get the job done!

I made sure that my code was technically sound by going through each of the red error codes as they popped up, which helped to save me a lot of time in the future. I think this also helped with efficiency, because I was able to save time by fixing those errors early on. Had I had not fixed them, it could’ve caused more errors later in the code, and generated confusion as I tried to fix them. In contact test at line 28, I started my second test in contact. I gave it two different ones to verify that I could make changes with the methods I wrote in contact. This is however the only class that I wrote multiple tests in.

The software testing technique that I used in this project was Junit testing. Junit testing can identify errors in the code early on through creating a test method. There are a few different things you can do, such as repeat the test multiple times, or assert what the output should be, and then compare to see if the expected and actual output match. Junit also offers the before and after methods, which helps to “clean up” after every test.

Some of the software testing techniques that I did not use for the milestones were integration testing, system testing and acceptance testing. Integration testing is used to test how the two classes work together. System testing is used to ensure that all the classes are functioning properly as a whole. Acceptance testing is the last stage in the process and makes sure that the product will be ready to go.

I had to use a lot of caution when writing the code because I wanted to think about how the customers needs come first. All my focus went into checking off the boxes for the customer before considering any additional special details I may want to add to the program. It doesn’t matter how nice the code is if it isn’t what the customer asked for. This caution also helped me consider the quality of my code from the other side. I did try to make sure that every single function I wrote was functional.

To limit bias when I was reviewing the code, I did ask one of my friends to review my code and tell me what they thought, and then compared our conclusions to see if there was anything I may have overlooked. Although there wasn’t anything this time, I have found issues in the past like this, and I’ve found that getting that type of “quality check” has helped me to become a better programmer overall.

It is important to be disciplined in your commitment to quality as a software engineer because ultimately it is your product that companies will pay money for in hopes that you are returning a good product. If you cut corners when writing your code or creating your tests, then you are only causing issues for that company in the future, and making a bad name for yourself as a programmer.

I did not actually use any sources for this other than my own code, so I do not have any citations to list.