\

Project Two

Chase Patten

22 APR 2021

Southern New Hampshire University

My testing approach for each of the three features was to try to break the problem down into each bite-sized piece. This allowed me to focus on what the requirements of the program were and what was needed from the code. I could then program the function or portion of code as I needed and then construct tests for those different portions of code. I made sure the requirements for each portion being tested was present in the test.

A specific example I can give for how I aligned my testing approach to the software requirements is in the constructors for the Appointment objects. I made sure that each Appointment had every single configuration for user input that I could. I made sure that there was a default one and one type of constructor that added a new user input each time. This way it built up to a final object with all user inputs thus meeting the requirements. I then tested these objects to verify that they were being created correctly.

I feel that my Junit tests were rather good in coverage percentage. They covered every part of the requirements that were coded in. This way they could be tested against the requirements to see if they give the correct output or perform the correct action correctly. By covering every aspect of the testing, I feel that my coverage area is fair. To ensure my code was technically sound I made sure to follow appropriate coding conventions when it came to spacing and naming conventions. You can see this as an example in the Appointments.java file on lines 4 through 9 where I begin the program making sure that all my variables are declared and named correctly.

To make sure my code was efficient I made sure that no portions of code were performing unnecessary actions. An example would be again, in the Appointments.java file on lines 75 to 82. I made sure that this updateDate function was doing each step correctly without adding too much to it. I wanted to make sure it was verifying that the information being updated was in fact correct and then I used illegal argument exceptions to alert the user of issues. Instead of just using basic print statement to the screen.

The software testing techniques that I used during this project were Unit Testing, Integration Testing, System Testing and Acceptance Testing. Unit testing is when the developer themselves verifies that a piece of code meets the requirements set forth. Integration testing was conducted when I then integrated the units into modules that were designed to perform a specific task together. I conducted my System Testing when I tested the software in a black box style environment and ensured it met the software requirements. Finally, I conducted Acceptance Testing where I made sure that the output was correct and that all requirements were met.

Some types of testing that I did not conduct during this project were testing for performance, security, usability, and compatibility. Testing for performance is when a developer tests for things like load times and how the software is performing with the hardware it is given. Testing for usability is when users take a turn with the software to see how well it performs and how easy it is to use. Finally, compatibility testing is when you test to see how a piece of software will behave in different environments. These can be things like hardware specifications or even operating device or operating system it is running with.

I employed caution as much as possible when testing and coding these tests. As I was learning this on the fly like everyone else, I wanted to make sure I did not get anything wrong. If I did get something wrong, I wanted to be able to easily figure out what I had done wrong and fix it. To accomplish this, I employed as much caution as I could writing the initial code according to the requirements and then building the tests against those same requirements. If I had misunderstood the requirements and how pieces of software behaved with each other it would make it impossible to design and deploy tests that would show the requirements had been met.

I tried to be mindful of my own bias as much as possible. As I understood from this course, in a normal world, a developer would not test their own code. This would help to cut down on bias. In this world, I did not have that luxury. Instead, I tried to be as humble as I could when it came to my own abilities. I also tried to make sure to always keep the requirements in mind when testing. I did not want to get distracted by my own bias and misinterpret what the requirements wanted.

It is important to never cut corners in testing as a developer. One of the reasons being is that people depend on your software. No matter what software it is and what exactly it does, when people use it, they are depending on it to work. To accomplish that you must test often through out the development process. If you cut corners, then your software could misbehave and cause an issue for a user. This issue could be something as simple as long loading times all the way up to something catastrophic, such as loss of data. Keeping this in mind as a developer is always important.