7-2 Project Two

Jonathon Gaspers

Southern New Hampshire University

7-2 Project Two

**Summary**

When creating the unit tests for each of the three features, I would first change my Eclipse view to be horizontally split, then I would have the unit test class at the bottom and the feature class at the top so I can work on them simultaneously. The technique I would use would be to write the documentation for the function and the functions test, then the function, and finally the test. Once I was done writing the new function test, I would run the unit test to make sure it worked as expected. If the test did not work as expected, I would use the Eclipse debugger to walk through the function to find where I went wrong and fix it, so it works as the software requirements stated. An example would be when I was writing the ContactService class, and I kept running into issues setting up the ability to add a new Contact to the list. At first, I had the add contact function as a void function with no return information that indicated it was successful or failed. I turned to use a Boolean function to return true if it added the new contact and false if it did not add the new contact. To test this in the unit cases, I would check if it successfully returned true. The following snippet shows this function:

(Jonathon G. (2021), ContactService.java)

As the Appointment and Task features are similar in function to the application's Contact feature, I recycled the code I used in the Contact feature's classes and changed them as needed for their feature requirements. Recycling the code and my technique of working on the function and then writing the test immediately after helped increase my overall code coverage within my test cases. The following is an example showing similar code, but adapted for the Appointment feature’s add appointment function and its code to show how it is tested:  


(Jonathon G. (2021), AppointmentService.java)



(Jonathon G. (2021), AppointmentServiceTest.java)

In the above snippets, we can see that I recycled the add contact function from the contacts feature and created a test to cover that function by verifying that the appointment was created and returned true. By following this format, creating the function then the test, I was able to verify every function in each of the three required features to make sure they met the requirements provided.

I ensured my test cases were technically sound as well as efficient. To do this, I made sure there were no errors or warnings in the code and by ensuring the three features met the given requirements. I also documented my code; before each function, I would describe the function, inputs it takes, and the possible outputs. For the test cases, I would write a brief description of the test, the expected test result, and the given test result after I performed the test. To help further increase the efficiency of the code, I did not include any extra code. I made sure to organize each of the functions within each class to be in an easy-to-read and understand the order. If I need to go back to the code later, I can easily understand what the code does and modify it to match the future requirements.

**Reflection**

The software testing techniques that I employed for this project are mostly just unit testing. While developing the code and following the requirements, I would do occasional static testing to verify my code followed the project's given rubric and requirements. Unit testing involves testing an individual component of the overall application; in the case of the three required features, I would use unit test cases to test each of the methods to verify they acted as they were intended to if a test failed when I expected it to pass, I would investigate and switch over to static testing. Static testing my code for the three features was a bit difficult at first; in the Contact feature, I was unsure of how to use the assertThrows method, but thanks to help from the professor and the documentation, I was able to figure out the proper way to set up the unit tests using assertThrows. To perform my static testing, I would have a window showing the requirements on one side of the screen, and the code on the other, then I would slowly read through the code while verifying it fit with the requirements. Two other types of testing I performed while working on the project's three features were integration and system testing. Integration testing is testing multiple components of a system when integrated together, and system testing involves testing the whole system. I performed these when I would test the "-Service" classes, using integration testing by testing the "-Service" classes and the individual object classes together (e.g., ContactService and Contact classes). This type of testing, in this situation, would also count as system testing as the matching "-Service" class, and its object class makes up the system for that feature.

There are many different testing techniques out there, and because I only used a select few of them, there is a lot that I did not need to employ for this project. A few examples of techniques that I did not use are stress, vulnerability, and security testing. Stress testing was not needed for this project because I did not need to test if the system can handle large amounts of contacts, tasks, or appointments(Types of software testing, n.d.). Vulnerability testing and security testing were not needed as the project's application did not require finding any potential exploits or other vulnerabilities (Types of software testing, n.d.).

The testing techniques that I listed that I didn’t use are used to help protect software applications like web applications from various exploits and attacks from those who want access to the private data or those who want to cripple the application somehow. Stress testing is used a lot in web development. It helps the developers learn how to increase the amount of users their application can handle without hindering the software's usability for other users when the application is under heavy load. Vulnerability testing will enable the tester to test the application to see if the application has any potential vulnerabilities, giving the tester an idea of how to fix the found vulnerabilities. Both of these tests mentioned fall under the blanket of security testing.

I have previous software development experience so getting into a cautious mindset when writing the project's code was not as difficult as I originally thought it would be. This was helpful because being in a cautious mindset helped me work through the project's requirements and make sure each of the features met the given requirements in the rubrics. I would write a bit of code and then test to see if it ran; finally, I would test the code against the requirements using a JUnit test case. It was important to keep in mind the programs' complexity regarding how the "-Service" classes used the various object classes (e.g., Contact, Task, and Appointment classes) to perform their required tasks.

To prevent any biases when writing the application and the tests for the application, I would write the tests and the functions to make sure they worked as intended. At first, I didn't notice, but I noticed a little confirmation bias when I was writing my test cases when working on the task feature. I had to work around that bias by focusing on the requirements and writing the test cases that matched the requirements a lot more closely than I was originally writing them to be. I had to learn to focus on writing the tests first, then write the function that is being tested. I ended up writing many failing tests then working to make them pass as I worked on the project features. This helped me work with an objective mindset and to avoid any biases in my work.

It is important when working in software development to be disciplined in how you go about writing your code. It is important to first start with a game plan on how you will go about writing the code. It can be as simple as using a pseudocode example on a napkin to explain what you want the application to do and roughly how to do it. Once you start writing the code, it is important not to cut corners so you don't accidentally introduce some catastrophic bugs that could cause the application to act unexpectedly. The best way to avoid technical debt is to think of the best solution to the problem instead of the quickest solution. Sometimes, the quickest solution can cause a team to have to go back and rework that part of the code base, which incurs more technical debt, which could have been avoided.

References

Types of software testing: Different testing types with details. (n.d.). Retrieved February 14, 2021, from https://www.softwaretestinghelp.com/types-of-software-testing/