Brittney Miller

SNHU

CS 320

Project 2

When going over my unit testing, it was easy to have them aligned to the software requirements as it tested the methods required in the base code. The requirements in Contact Class said the first and last names could not be null or more prolonged than ten characters.

The way I tested this was the Junit test that threw an IllegalArgumentException if the test ran. It was found to be null or more prolonged than ten characters, which would then run the new contact through the system, checking for contact ID, firstname, lastname, phone number, and address. In the same way, we tested the length of the task id to ensure it was less than ten characters. This test was a little different as it would throw a void testTaskIdTooLong method, which would output an error.

Junit testing is challenging for me as it’s a lot of repeating, and I feel there is a better way to run testing. Still, for the project in general, Junit testing is the best choice as it’s not a large project, and you can test as you move through the build instead of waiting until the end to find an error or the test is not running. You can tell your Junit tests are effective if they have 80 percent coverage in testing or if they don’t. You will need to adjust and change to ensure the testing is accurate if they are less than.

I checked multiple times to ensure my code was technically sound. If I ran into trouble, I used the internet to help me find where my code was running into issues or what I might have coded incorrectly. I also used the teacher's feedback to change some things. I also used arrays to help ensure my coding was technically sound. I used ArrayList<String>() to list the strings in my code. I also made sure I set default values as in firstname = (“Brittney”); that way, making changes for me was more accessible, and there would be a default if the name entered was null or more prolonged than ten characters.

When it came to code efficiency, I used the feedback from my instructor a lot. But I also used YouTube tutorials on things I felt were overwhelming or just long and felt there was a better way to navigate part of the coding. I tried to use best practices that are up to date to ensure the code was readable to everyone and made sense for someone who came in to make changes.

I mainly used specification-based or white and black box categories for my software testing techniques. I used black box categories to check valid and invalid input from the user. This is a best practice and an easier-to-understand way of testing functionality and code for errors. White box categories were used more for if, then statements since it’s known for breaking down testing into sections, which is an if, then statement, or loop in coding.

I did not use any experience-based testing techniques in my project mainly because this is used to test exploratory testing, which wouldn’t make sense for my project since the majority of our code is particular and has specific requirements which means exploratory testing which is used to test areas that lack specifications wouldn’t work to the best of its abilities.

Black box category testing techniques test clear-cut cut, precise functionality in code. White box testing is usually used when the outcome of your code is completely understood and has a particular function. Experience-based testing is more used when there is no specific testing. It is for more informal testing.

I didn’t necessarily have a mindset going into this project other than to understand better the testing code I’ve written. In my other classes, we ran it through a tool that alerted us of an error, so it was nice to apply and write a unit test on our own. But it was not without challenges, that’s for sure. I was cautious when reviewing feedback and using best practices since I’m still new at coding in general and testing. I had to be sure I was looking at best practices, employed them correctly, and used the correct testing format, as it was easy to get lost as I worked through. Appreciating the complexity and interrelationships of the code we are testing is essential because it can significantly impact your output's quality and performance. Say I had functioning code that outputs things correctly and how I wanted it to, but as I ran a Junit test, the coverage percentage was low. That means there are significant gaps in my coding that are unreliable and can cause issues, glitches, and bugs and make the code unsafe. While the code still functions and can work, the output is not good and reliable, which can cause problems down the road.

It’s easy not to be biased when looking at the code when you are newer to coding since you want to ensure everything works and works well. So, triple-checking code and Junit testing and providing a high percentage when testing code is easy to aim for. You also want to think about how the code will represent your work ethic and how much you care about your job, the client, and the functionality of it. That is what keeps my bias out of the coding, knowing that a part of my code could have been my thousandth time writing it, and I know it’ll work, but that doesn’t matter. I still want to test it repeatedly until I am cheerful and confidently say it is the best I can produce.

Being disciplined is extremely important in your commitment as a software engineer and your quality. Your output says a lot about you, your employer, and how much you care. It’s important not to cut corners when writing or testing code because we are upheld to standards, and we want to be sure we aren’t compromising quality or performance when it comes to the final project when we are finished. I plan to avoid technical debt by using best practices and tools at my aid that are meant to help us with coding, ensuring safe, functional code, and ensuring everything I output is of the highest quality and represents myself well. I liked learning about the agile method and will use this a lot in the future as it helps keep communication open between the team, the client, and developers and ensures any changes are immediately advised, meaning the team doesn’t spend time on unnecessary items.