Brandon Bond

SNHU

CS-320

12/9/2024

When creating the packages I would focus on what the requirements were, such as length of characters in a string or parameters for the string that make it false for the requirements. Then I would move onto building the Service class for the packages that would focus around adding, deleting, or updating IDs in that package. This was the case except for the Appointment Service Package that only focused on adding and deleting an ID. After I created both classes then I would move onto creating the test classes. In the test classes, my approach was to follow each line from the class I was testing and create test cases that both follow and break its parameters. For example, in the test cases for the Task Class, the Description string must be no longer than 50 characters and not be null. So, when testing I made sure to make a test case for a correct description, one that broke the description's length, and one that was null, I did this for each requirement to make sure that I could cover all possibilities in the code. After creating the test cases for each of the packages I would check the coverage percentage and make sure that it would cover over 80% of the code that way I would know that the tests were effective in covering the requirements.

The experience that I had while writing the test cases was mostly fun and it was easier to learn and understand than my previous experience with java. When it comes to the code quality, I would make sure to comment on each test case of what the test was for and follow good practices like no duplication on testing and keeping the code organized and readable. When it came to coverage in my testing the task test class is the best example of this. In the Task Test I test for task creation with all parameters within the requirements, then I go and create a test for invalid description where the description is longer than 50 characters which throws IllegalArgumentException, after that test I tested the description a third time for a null description which is not allowed and will also throw IllegalArgumentException. I did this for the description name and task ID in the code, then when I created all the tests, I went back to the class and made sure all the parameters were covered in the test class. This was how I would make sure, along with a coverage percentage test, that the testing was efficient.

Two software testing techniques that I employed in the testing process were functional testing, non-functional testing, and white box testing. Functional testing tests for specific functionality in code, in this case I tested for adding, deleting, and updating an ID in two of the packages and just adding and deleting in another package. This testing allowed me to see if the software would work when the requirements were met. Non-functional testing tests the code's behavioral aspects, which would fall under the string parameters testing. I tested the requirements by individually testing too many characters, null, and correct number of characters for the ID, Name, and Description for each package. White box testing tests the internal structure of the system, which could include coverage percentage. I tested the code coverage for each package after finishing them to ensure that the tests covered the code. Other software testing techniques that I did not use was testing related to changes. Testing related to changes is testing done after finding a defect in the code and fixing it, then you would test the code after the fix for any other issues. I did not have any code that needed to be fixed due to a defect, other than wrongful code that would not test properly in the first place; therefore, I did not need to retest due to bugged code.

The uses for these tests will find different importance in different situations. I would like to talk about the usage of these tests from a video game perspective. Recently I have been playing Old School RuneScape again and when it comes to a bigger functionality of code such as an MMORPG like this then all these testing techniques are of high importance. For the sake of space, I will only talk about one functionality. The account creation functionality is important and would use all testing techniques mentioned. Functional testing would be used for testing the creation of the account and the two-step verification of that account, Non-functional testing would be used for testing the parameters of the account name, email, and username, white-box testing would be used for testing the interface of the account creation and the data storage of the accounts information, testing after changes would be used when fixing any bugs found previously and this testing would be used for security purposes.

When I started the course I was already in burn out and my mindset was in the wrong place, but as we got into week 2 or 3 I started getting more excited for the class and put myself into a testing mindset which really helped me push through not only this class but also my other class at the same time. In terms of software testing, I would try to always test every aspect as I did not want to miss any issues that there might be. It is important to understand the complexity of testing the code so you do not take a simple approach when testing it, allowing you to find more issues there may be in your code. As an example of this, the simplicity of some of the requirements could be overlooked like only allowing for a certain number of characters, I personally overlooked this in the original milestone but fixed it in later testing.

I try to limit bias in my testing by adopting a mindset that something is wrong, and I need to find it. This is not a healthy way to do it, but it just takes experience to get better at testing. There are a few reasons why I think that bias would be an issue when testing your own code. I think that for some people arrogance and pride could affect testing, when you do not think you did something wrong then you will be less likely to test for issues, for me this is not a problem because there is an underlying issue. Another reason bias could cause an issue is being too close to your code and not being able to see it all in perspective, this is an issue because it can cause you to miss some things. I always think that a second eye will be better for the code and the end user.

The importance of being disciplined in committing to quality code is to make sure that your code is uncomplicated which allows for an easier end phase. When you test later, or you are trying to fix an issue that is found in testing it calls for easier fixes when the code is clean and up to a standard. Another benefit of ensuring quality code is fewer mistakes made in the first place, which will lower the workload of testing and repairing any code. When corners are cut then it causes more end work like fixing and testing which, if it were done properly in the first place, would not need to be fixed. I plan to do things to the best of my ability the first pass through the code in a professional setting to ensure less work and stress in the end run of the development process.

Sources:

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.