Crystal Berkhan

August 10, 2023

CS 320

Project Two

**Summary**

Testing is a crucial phase of software development to ensure the software functions correctly and meets the client's requirements. To confirm that the software met the requirements, I utilized the module four guidelines and broke down the classes into smaller sections to work on. For instance, I started creating the Task class and verified that it met the requirements.

In the “TestTask” Junit Testing, I received a green bar that confirmed all seven of my tests executed and successfully ran; however, I did struggle with the “TestTaskServices” Junit Testing and did not receive a green bar. I do believe I might have had something missing from the “TaskService” class that I needed to utilize to successfully run my tests. Two of them did run correctly, but the third did not and I could not figure out why.

void testNameTooLong() {

Assertions.*assertThrows*(IllegalArgumentException.class, () -> {

new Task("42578", "HamburgerManSandwhichTunaFish", "ABCDE");

I did utilize the video from module four to aid in completing the testing portion of this assignment and I believe I understand Junit testing a lot better than the first milestone assignment.

To ensure my code was technically sound and efficient, I named the tests to match what I was testing for to determine if any errors in the code would cause it not to meet the requirements based on the Junit tests. This allowed me to identify and quickly correct any errors I encountered. I also double checked the coverage percentage multiple times to ensure my percentage was at least 80%. I specifically checked each test as well for one hundred percent coverage; however, if I was not able to achieve one hundred percent coverage on each test, I adjusted what I could to ensure the entire testing package of my code reached eighty percent.

*@Test*

void testIDTooLong() {

Assertions.*assertThrows*(IllegalArgumentException.class, () -> {

new Task("4257898563423", "Hamburger", "ABCDE");

});

}

For example, if there had been an error when testing the above line of code, I would have known there was an issue with the code in the Task class for the length of the task ID.

**Reflection**

During the first milestone, I was very confused about completing the testing aspect of my code; however, while completing the second milestone, I found resources that gave me clarification on how to test my code using Junit testing. I utilized the same techniques for milestone three as I did for milestone two. I reviewed the code constantly to ensure I did not have any errors in spelling or syntax but also utilized the Junit testing to ensure the code acted as it was supposed to based on the inputs provided. When there was a red bar to indicate the test failed, I was able to review the issue and determine what I needed to fix to ensure my code ran based on the requirements provided. Another technique I utilized was Boundary value analysis. This technique was necessary and crucial when testing my code because the requirements specifically stated, “The ID cannot be longer than 10 characters.” I ran specific tests to ensure that an ID in the correct bounds was accepted and an ID longer than 10 characters was rejected.

While completing the milestones, I did not use regression or security testing. I completed the classes first before beginning to write my tests and typically there was not much to adjust after running the Junit test. I did not have to go back and adjust the code after running the tests, hence there was no need for regression testing. This testing is practical when requirements are adjusted later into the development of a big project but was not necessarily needed for this project. Security testing is crucial when developing any software; however, I did not utilize this technique while completing the milestones.

After reviewing the requirements for the milestone and the project, I believed the project would not be very difficult to complete. Meeting the requirements when creating the classes was fairly easy; however, I did struggle with running the Junit Tests because I am very new to testing software. I did not realize how important the complexity and interrelationships of the code were until I began testing and this made me write my code carefully for the next milestones. I tried to write my code as cleanly and organized as possible.

Before completing this project, I did think it was a good idea to have developers test their code; however, now I see that the developer can be very biased during this process. I have certainly changed my mind now and do think the code should be tested by someone other than the developer. I believe a fresh perspective on the code from someone other than myself would have been helpful and they possibly would have been able to identify issues I did not. I did try to avoid being biased when completing the testing portion and I tried to create tests that would prove the requirements were being met.

It is crucial to be disciplined in your commitment to quality as a software engineering professional to ensure the code is of high quality and effective. Cutting corners while writing or testing code leads to more expensive and difficult issues to fix later. I plan to avoid technical debt as a practitioner in the field by constantly practicing and learning new ways to make my code more efficient and up to date with the best practices in the industry.

References

GeeksforGeeks. (2023, February 6). *Software testing techniques*. GeeksforGeeks. https://www.geeksforgeeks.org/software-testing-techniques/