Amarah Kirkman

02/24/02023

CS320 Project Two

Summary

I made sure I had all the specifications needed by the customer or project and tried my best to match my approach with the software requirements for each feature. One good illustration of this is the requirement for specified character values in the contact class names. I was instructed to ensure that the appointment ID was not null, could be updated, and had no more than ten characters in addition to the appointment class. I then verified these were accurate using JUnit testing. I think that was one of the more difficult tasks, learning how to operate and check for JUnit coverage.

Week by week, the quality of my JUnit tests improved overall as a result of the resources in each module's increased contribution to the coverage percentage. The task test had a comparatively low coverage % when compared to the contact test's coverage percentage, while the service test had the highest overall coverage percentage. I could tell that the majority of the functions in the code had been properly covered by the JUnit tests I was running whenever a positive coverage % was displayed.

In order to ensure the technical soundness of my code, I used both strings and arrays. I did a few things to make sure my code worked. My past classes at Southern New Hampshire University gave me some basic coding expertise because they were more general computing courses rather than deep dives into particular languages. I always begin any assignment I've undertaken in the past by matching my tasks to the project specifications. Every time I deviate from this strategy, I either overthink my plan or overlook important details. To make sure my code is efficient; I also think it's important to run it frequently to make sure there aren't too many issues as I go.

Reflection

Component, or unit, testing methods are exemplified by the methods I employed in my software testing. Here, code blocks are tested to make sure they function in accordance with the specifications. I did not employ integrated systems, or acceptability testing, three other categories of testing methodologies, for these assignments. Code that has passed the relevant unit tests is now tested during integration testing to make sure that different components function as intended when they are combined. After passing integration testing, integrated components go through systems testing. This method of testing is intended to evaluate a program or application's overall system functionality. Acceptance testing is done at the end of the project by the final client or intended user, and the goal is to determine if the software is ready for release.

To be completely honest, I thought that "testing" just meant executing the program and looking for bugs at the bottom. I would characterize acting as a software tester as committed as opposed to cautious. We must continually make sure that the final product's quality and functionality are not compromised. We are unable to expedite program completion, and we naturally want to ensure that the final output meets all customer criteria. I can see why someone would think there might be prejudice if you wrote and tested the code yourself. The easiest way to reduce prejudice, in my opinion, would be to allow other programmers to review, edit, and potentially even comment on your code, as well as to constantly be open to change and learning. Since I don't think I'm very good at programming, I can't say that I'm biased toward the code I developed. I think I would be more likely to ask for help and welcome any input or criticism.