Latoria Judkins

SNHU

CS 320

6/23/24

**CS 320 Project Two**

When performing unit testing approach for the three classes such as Contact, Task, and Appointment service I had to execute a functioning code without errors that would meet customer requirements. For all three classes I used JUnit testing approach and to ensure that my approach aligned to the software requirements I followed the instructions step-by-step. An example is one of the requirements of Task Service was to be able to add tasks with a unique ID, which is shown in the image below.

A close up of a text

Description automatically generated

A screenshot of a computer

Description automatically generated

The overall quality of the JUnit test was effective because I had to ensure that the coverage was 80% or higher. If the code wasn’t 80 % or higher then that let me know that the requirements were not met, and I was provided with errors for the code in small portions that needed to be fixed. The code is technically sound because I tested it numerous times until there were no errors, and the JUnit displayed a successful green bar as display below. The code is efficient because I received feedback from my instructor that my code aligned with the software requirements.

A screenshot of a graph

Description automatically generated

Some of the software techniques that I used are as follows: JUnit testing was used in all three modules which consisted of testing individual portions of the code to make sure it is functioning properly and to uncover errors and System testing which was used to test the entire code to ensure that it meets the client requirements. JUnit testing is a form of white boxing or Java unit testing framework which examines the individual components to verify functionality. System testing or black box testing which ensures that the system meets all client requirements. The other software testing techniques that I didn’t use were Security testing which would ensure that the application doesn’t contain any vulnerabilities and User Acceptance testing is performed by the customer.

The practical use for black boxing is that it verifies how the software performs and white boxing reveals if there are any defects or component malfunctions within the code. Acting as a software tester I approached very cautiously while working with this complex code because I know I had to pay attention to the detail to execute an error free code. I always keep in the back of my mind a small error can lead to bigger problems while wasting time. It is important to appreciate the complexity and interrelationships of the code when testing to ensure reliable and user-friendly product. For example, when testing software, you must remember that there are other components that help execute a successful code so if you make changes within your code, it could affect the behavior or interaction of other components involved. I can imagine that bias would be a concern if I were responsible for testing my own code because I might unknowingly overlook errors. A few ways that can limit bias when reviewing my own code are through peer review and continuous integration (CI). The importance of being disciplined in my commitment to the quality as a software engineering professional ensures effective time management, team collaboration, and continuous growth which helps you in work or personal experiences. It is important not to cut corners when it comes to writing or testing code because it can lead to defects in the code or technical debt.

**Resources**

*Functional Testing: A complete learning Guide - Parasoft*. (2024, April 11). Parasoft. https://www.parasoft.com/learning-center/functional-testing-guide/

Hamilton, D. (2024, January 11). *JUNit tutorial with examples: Setting up, writing, and running Java unit tests*. Parasoft. https://www.parasoft.com/blog/junit-tutorial-setting-up-writing-and-running-java-unit-tests/