Mary Peterson

7/30/24

CS320

Module 4 Journal

**To what extent was your testing approach aligned to the software requirements?**

I believe my testing approach aligned with all the software requirements. For the Task Class, I made sure that the ID String no longer than 10 characters, not null, and couldn’t be updatable. The name String couldn’t be longer than 20 characters and not null. Then, the description field couldn’t be more than 50 characters and not null. The task service is able to update tasks with a unique ID, delete tasks per task ID, and update task field per task ID. The only fields that are updatable are: name and description.

**Defend the overall quality of your JUnit test for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?**

To defend the quality of my JUnit tests for contact service and task service, I’ve made sure to get solid coverage through a series of well-structured test cases. The Contact Service Test Class checks out all the main functionalities, like adding, retrieving, deleting, and updating contacts. The tests cover both positive and negative scenarios to make sure all branches and exception handing paths are tested. Similarly, the tests for Task Service go over the key methods for task management, confirming that tasks can be added, deleted, and updated properly. With such high coverage, I can confidently say that the JUnit tests effectively verify the correctness and vigorousness of contact service and task service.

**How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

A black background with white text

Description automatically generated To ensure that my code was technically sound, I named all my classes and methods according to the requirements. I ensured that the methods @DisplayName() and @Test were declared in the JUnit tests.

**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

Because JUnit uses test cases to ensure coding is efficient, I would need to test a good ID and a bad ID. So, to make sure that my code was efficient, I made two tests to test the effectiveness of my coding.

