Christian K. Foster

Southern New Hampshire University

CS 320 Software Testing

Professor Smith

In developing the mobile application, I employed a rigorous unit testing approach to ensure that the application’s core functionalities were implemented correctly and aligned with the specified requirements. This report summarizes my approach to unit testing for adding, deleting, and updating tasks, reflects on the effectiveness of these tests, and discusses the testing techniques used, mindset adopted, and the importance of quality in software development.

My unit testing approach was designed to validate each feature of the application thoroughly. For adding tasks, I used the testAddUniqueTask method to ensure that tasks were added correctly with unique identifiers and accurate details. This test checked if the task list was updated correctly and verified that the new task’s attributes matched the provided input. By asserting that the task was included in the list with assertTrue(TaskService.tasks.containsKey(id)); I confirmed that the feature met the requirement to manage and track tasks effectively.

The test for deleting tasks, testDeleteTask, was focused on verifying that tasks could be successfully removed and that the task list updated accurately. This method involved adding multiple tasks and then deleting one by its ID, ensuring that the deleted task was no longer present in the list. Assertions like assertEquals(2, TaskService.tasks.size()); confirmed that the task list’s size was correctly adjusted post-deletion and demonstrated that the feature worked as intended. Additionally, the test checked that attempting to delete a non-existent task did not alter the list’s size, ensuring robustness.

Updating tasks were tested with two methods: testUpdateTasks and testBadUpdateTasks. The former validated that a task’s description could be updated correctly using a valid ID, confirmed by the assertion assertEquals("Updated task details", TaskService.tasks.get(id).getDescription() );. The latter method ensured that no changes were made when an invalid ID was used, thus verifying the system’s resilience to incorrect updates. Together, these tests ensured that the application could handle both valid and invalid updates effectively.

The quality of the JUnit tests was validated through their ability to cover various scenarios and catch potential issues. The tests demonstrated effectiveness in checking functionalities such as task addition, deletion, and updating. The comprehensive coverage was evidenced by the accurate validation of tasks and list updates, confirming that the features worked as expected and were aligned with the requirements.

Writing the JUnit tests involved ensuring technical soundness and efficiency. Technical soundness was achieved by using precise assertions, like assertTrue(TaskService.tasks.containsKey(id)); to confirm that tasks were correctly added. Efficiency was maintained by avoiding redundant operations and ensuring that the tests were streamlined, as illustrated by the concise assertions used to validate the task list’s state.

The testing techniques employed primarily included unit testing, which focuses on validating individual components of the application. Unit testing is characterized by its speed, isolation, and immediate feedback. It is crucial for catching issues early in development and ensuring that specific functionalities work as intended. Techniques not used in this project, such as integration and system testing, also play vital roles. Integration testing examines interactions between components, while system testing assesses the entire application to ensure it meets overall requirements. Although these techniques were not part of this unit testing phase, they are essential for comprehensive software validation.

Adopting a cautious mindset was critical throughout the project. This mindset ensured that all potential edge cases were considered, as demonstrated by tests like testBadUpdateTasks, which verified that invalid updates did not affect existing tasks. Understanding the complexity and interrelationships of the code was essential for accurate testing and validation. Limiting bias was also important; by approaching tests objectively, I ensured that the functionality was verified without personal bias influencing the results.

Being disciplined in testing is crucial to avoid technical debt and ensure that software remains reliable and meets user expectations. Thorough tests like testUpdateTasks and testDeleteTask demonstrated a commitment to comprehensive coverage and accurate functionality verification. To avoid technical debt, I plan to adhere to best practices, continuously update and refactor code, and maintain thorough testing.

In conclusion, maintaining a disciplined approach to unit testing and being mindful of potential biases is essential for developing robust and reliable software. The insights gained from this project highlight the importance of rigorous testing and a commitment to quality in software development.

References:

*What is software testing - A Complete Software Testing Tutorial with examples*. (n.d.). Www.lambdatest.com. <https://www.lambdatest.com/learning-hub/software-testing>

Bechtold, S., Brannen, S., Link, J., Merdes, M., Philipp, M., de Rancourt, J., & Stein, C (n.d.). *JUnit 5 User Guide*. Junit.org. <https://junit.org/junit5/docs/current/user-guide/>