Bradley Mills

CS-320-J8975

bradley.mills@snhu.edu

**Summary**

Approach to Unit Testing:

The three main features of our mobile application encompassed Contacts, Appointments, and Tasks. For each feature, a systematic unit testing approach was taken:

Contacts: Testing centered around validation of inputs to ensure that all contact attributes (ID, Name, Phone Number, Address) were correctly formatted and non-null. Emphasis was placed on testing boundary conditions, such as excessive string lengths, as illustrated by tests like testContactInvalidArguments.

Appointments: For this feature, the main focus was on ensuring that appointments had a valid date and description. Boundary conditions were again explored, such as excessive lengths for IDs and descriptions, and invalid date entries.

Tasks: The testing for tasks primarily concentrated on ensuring each task had a valid ID, name, and description. Invalid and boundary test scenarios, like those seen in testTaskInvalidArgument, ensured that the application would handle errors gracefully.

Alignment with Software Requirements:

The testing approach closely followed the outlined software requirements. For instance, the requirement that all contact attributes must be present and valid was addressed by tests like testContactNullArguments which ensured that no attribute was missing or null. Moreover, requirements specifying boundary constraints for attributes were adhered to by designing tests to validate against overextended lengths and incorrect formats.

Quality and Effectiveness of JUnit Tests:

The effectiveness of the JUnit tests can be asserted through the coverage percentage. These tests not only cover the happy path scenarios but also rigorously evaluate the boundary and error conditions, ensuring the system's robustness. The testUpdateContact and testAppointmentInvalidArguments are testament to this, capturing potential pitfalls and ensuring the system behaves predictably.

Experience Writing JUnit Tests:

Writing JUnit tests proved to be an enlightening experience. It encouraged forward-thinking and anticipation of potential system pitfalls. By thinking from a user's perspective, the tests were designed to capture as many scenarios as possible, ensuring the system's resilience.

Technical Soundness and Efficiency:

Ensuring technical soundness was paramount. Lines of code, such as the exception checks in testDeleteContact, ensure that invalid deletions don't compromise the system. To ensure efficiency, specific checks, like in testAddTask, were implemented to quickly confirm the task addition process.

**Reflection**

Testing Techniques Employed:

The primary software testing techniques employed were Boundary Testing, Exception Testing, and Positive Testing. Boundary Testing, as seen in the checks for excessive string lengths, ensured that limits were well-defined and not breached. Exception Testing made sure the software gracefully handled errors. Positive Testing, illustrated by the testTaskGetters, confirmed that under normal conditions, the software behaved as expected.

Other Testing Techniques:

Though not explicitly used, techniques like Stress Testing (to evaluate system limits) and Security Testing (to ensure data protection) are crucial in other contexts. These tests might be more relevant when scaling the application or if it was to handle sensitive data.

Implications of Techniques:

While Boundary and Exception testing are paramount for any application, Stress Testing becomes crucial for applications expected to handle large user loads. Similarly, Security Testing is a non-negotiable for applications dealing with sensitive user data.

Mindset:

Testing required an immense amount of caution. Recognizing the intricacy of code and potential for unintended interdependencies was key. For example, understanding that an invalid task addition might corrupt the entire task list required comprehensive testing around this feature.

Bias limitation was also a significant consideration. As developers, we might be too close to our code, potentially overlooking some flaws. This is where unit tests play a pivotal role, providing an objective measure of functionality. If one were to only test their own code, they might subconsciously avoid paths that they subliminally know to be flawed.

Lastly, discipline is the cornerstone of quality software engineering. Cutting corners might offer short-term speed, but in the long run, it leads to technical debt – where future changes become cumbersome due to past shortcuts. Commitment to thorough testing ensures sustainable, scalable, and robust software development.

In the realm of software development, the commitment to thorough testing is imperative to ensuring quality and security. It not only ensures the delivery of a reliable product but also protects against future complications. By understanding and applying varied testing techniques, and continually reflecting upon our approach, we strive to ensure the pinnacle of quality in our software products.

References

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.

Bechtold, S., Br, S., Merdes, M., Philipp, M., de Rancourt, J., & Stein, C. (n.d.). JUnit 5 user guide. https://junit.org/junit5/docs/current/user-guide/