Caleb Sorell  
Southern New Hampshire University  
CS 320: Software Testing and Quality Assurance  
Instructor: Ramsey Kraya  
October 13, 2025

This report provides an overview of the unit testing approach, test quality, and development experience for the contact, task, and appointment services created for Project One. The objective of this project was to validate that each service met its specific functional requirements through an organized and requirement-driven unit testing process using JUnit. The report also reflects on the testing techniques employed and the mindset adopted throughout development.

My approach to unit testing emphasized a requirement-based, white-box testing methodology. Each feature—Contact, Task, and Appointment—was broken down into testable requirements to ensure complete coverage of expected behaviors. Using knowledge of internal implementation, I wrote JUnit tests that validated data constraints, exception handling, and business logic for each service.

For example, the Contact service required that each contact include a unique identifier, a valid phone number, and names that did not exceed length restrictions. I verified these requirements by writing tests that attempted to create contacts with invalid data, ensuring that exceptions were thrown as expected. The Task service required unique, non-updatable task IDs, which I tested by adding duplicate entries and confirming that the system rejected them. Similarly, for the Appointment service, I verified that appointments could not be created for past dates, ensuring the system correctly handled invalid time inputs.

This testing approach aligned closely with the software requirements because every test case corresponded directly to a functional expectation outlined in the specification. The result was a transparent and traceable connection between requirements and test coverage.

The overall quality of the JUnit tests was high, supported by a coverage analysis that reached approximately 98 percent across all services. This coverage was measured using Eclipse’s integrated testing tool (EclEmma), which verified that nearly all code paths, including conditional branches and exception handling, were executed during testing. High coverage indicates a low likelihood of undetected bugs and provides confidence in the robustness of the final implementation. The tests validated both normal and exceptional workflows, ensuring the services behaved predictably under a variety of conditions.

Writing these tests reinforced my understanding of systematic testing and efficient code structure. To maintain technical soundness, I emphasized testing both valid and invalid inputs, particularly focusing on edge cases that could trigger exceptions or unexpected results. For instance, I confirmed that invalid data entries such as null values or overly long strings were handled gracefully. These efforts helped ensure that the software behaved predictably under adverse conditions and that validation logic was both effective and consistent.

Efficiency was achieved through the strategic use of JUnit features such as the @BeforeEach annotation. This approach allowed the test environment to initialize shared objects once per test cycle, minimizing redundancy and improving readability. Organizing test cases into logical groups and using clear, descriptive method names also supported maintainability and clarity. By creating concise and purposeful tests, I ensured that future modifications to the code could be verified with minimal effort.

The project primarily utilized unit testing and white-box testing. Unit testing isolates small sections of code to confirm their correctness, while white-box testing involves validating the internal logic and control paths of the software. These techniques allowed for early detection of issues and ensured that each component functioned correctly in isolation before integration.

Other testing approaches not used in this project include black-box testing, integration testing, and system testing. Black-box testing examines input and output behavior without insight into internal implementation and is typically applied later in the development process to verify user-facing functionality. Integration testing validates communication between components, while system testing ensures the overall application meets its intended purpose in a production-like environment. These methods would be essential in later stages of the mobile application’s lifecycle but were beyond the scope of this assignment, which focused specifically on backend service validation.

Adopting the mindset of a careful and skeptical tester was crucial throughout this process. I intentionally sought ways to “break” the program by testing for extreme and invalid inputs to uncover weaknesses in logic and validation. Recognizing how small defects could cascade into larger system failures reinforced the importance of diligence. For instance, overlooking a simple field length check could later result in data storage errors or user interface malfunctions.

Limiting bias was another key aspect of maintaining objectivity. To achieve this, I treated the requirements document as the single source of truth rather than relying on assumptions about my own code. Developer bias can easily arise when testing personal work, as one may unintentionally favor expected outcomes or neglect challenging scenarios. I avoided this by creating tests that intentionally targeted edge cases and failure points, ensuring the code met the documented requirements rather than personal expectations.

Finally, discipline and commitment to quality guided my approach to both development and testing. Cutting corners in software testing can create technical debt—delayed costs and rework caused by inadequate testing or short-term fixes. For example, skipping validation tests for null fields may save time temporarily but can lead to system crashes or corrupted data later. To prevent such issues, I plan to maintain a consistent testing discipline by practicing test-driven development when feasible, writing clear and maintainable code, and performing regular code reviews to identify potential weaknesses early.

In conclusion, this project strengthened my understanding of structured software testing and the importance of balancing efficiency with thoroughness. The experience underscored the value of disciplined, unbiased testing and reinforced the mindset needed to uphold software quality in professional development environments. By applying systematic testing techniques and maintaining a strong commitment to quality, I can ensure future projects meet both technical and client requirements with confidence.

**References**

JUnit.org. (2025). *JUnit 5 user guide*. https://junit.org/junit5/docs/current/user-guide/

Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner’s approach* (9th ed.). McGraw-Hill Education.