Project Two

Beau Zermeno

Southern New Hampshire University

CS-320: Software Test, Automation QA

Instructor: Dr. Angel Cross

June 22, 2025

Project Two

**Summary**

In Project One, I applied unit testing using JUnit to verify the core features of a mobile application: adding contacts, updating contact fields, and deleting contacts. Following a test-driven development (TDD) approach, I ensured that each method in the ContactService and Contact classes had an associated JUnit test. For the add contact feature, I confirmed that each contact was assigned a unique ID and that valid data correctly added a contact to the list. Assertions such as assertEquals() were used to validate behavior. When updating contact fields like the first name, I tested for valid updates and handled edge cases, such as preventing null or overly long inputs. The deleteContact() functionality was tested to ensure a valid ID removed the correct contact and that non-existent IDs triggered appropriate exceptions.

**Reflection**

My testing approach was closely aligned with the documented software requirements. For instance, the requirement stating that each contact must have a unique ID was tested by attempting to add duplicate IDs and confirming only one instance remained. Similarly, input validation requirements were verified by testing boundary conditions, such as rejecting names longer than ten characters.

The overall effectiveness of the JUnit tests was demonstrated by the high test coverage achieved. Every public method in both the Contact and ContactService classes was tested. Both positive and negative test cases were included to ensure the code handled expected and unexpected inputs correctly. For example, a test for an overly long first name confirmed that an IllegalArgumentException was thrown, verifying that validation logic is in place.

Writing the JUnit tests deepened my understanding of both the business logic and the internal code structure. Initially, creating comprehensive negative test cases was challenging, but through practice, I improved my ability to simulate real-world use cases and potential user errors. To ensure technical soundness, I included checks for exception handling, such as verifying that deleting a contact with a nonexistent ID correctly threw an exception. Efficient code was achieved by writing targeted tests that isolated single behaviors, avoiding unnecessary complexity. For example, adding a contact and verifying the contact list size kept the test simple and focused.

In terms of testing techniques, I employed unit testing to assess individual methods, boundary testing to evaluate inputs at the edge of constraints, and exception testing to verify that the application responded properly to invalid data. Techniques not used in this project included integration testing, as there were no interactions between external components, and system testing, which was not applicable given the backend focus of the project. Regression testing was also not performed due to the absence of iterative changes during development.

Each technique has practical applications in different contexts. Unit testing is ideal for validating logic in isolated modules, while integration testing becomes essential when dealing with APIs or interconnected services. Regression testing is critical in continuous integration environments to ensure new changes do not disrupt existing functionality (Testsigma, 2025).

**Mindset**

Throughout the project, I adopted a cautious and detail-focused mindset. Understanding the relationships between the Contact and ContactService classes was key to appreciating the complexity of the system. For example, deleting a contact had implications for the integrity of the internal list and required careful testing to ensure no residual references remained. To limit bias, I approached testing from both a developer and tester perspective. I scrutinized my own code with adversarial inputs, recognizing that developers may unconsciously overlook potential flaws. For instance, my initial implementation of the updatePhone() method failed to reject non-numeric characters, which I only discovered through thorough testing.

Remaining disciplined in my commitment to quality was essential in preventing technical debt. I avoided cutting corners by writing comprehensive tests upfront and maintaining consistent coding standards. In my future work, I plan to use tools like linters and continuous integration systems to enforce quality standards. I will also use static analysis tools to catch potential issues early and focus on writing modular code that is easier to test and maintain. This disciplined approach will help ensure long-term project success and code reliability.

References

Testsigma. (2025). *Regression Test vs Unit Test: Key Differences*. Testsigma.

<https://testsigma.com/blog/regression-test-vs-unit-test/>

‌