# CS 320 Project Two: Summary and Reflections Report

Name: Kevin Tippin

Course: CS 320: Software Test Automation

Date: October 17, 2025

Summary

Unit Testing Approach

For this project, I developed and tested three main features: the Contact, Task, and Appointment services. My approach centered on JUnit 5 unit tests designed to validate inputs, enforce business rules, and prevent data corruption.

In the Contact feature, I verified that IDs were unique and non-null, names didn’t exceed character limits, and phone numbers were exactly 10 digits. For example:

assertThrows(IllegalArgumentException.class, () -> new Contact(null, "John", "Doe", "1234567890", "123 Main St"));

Not all tests passed right away—some failed in Contact.java and ContactTest.java due to constructor and field validation mismatches, though ContactServiceTest.java passed without issue. Fixing these failures helped align my logic with the exact software requirements.

The Task tests were my biggest challenge. My initial TaskTest.java was poorly structured, with missing exception checks and inconsistent assertions. After revising it to properly handle null and overly long inputs, the tests passed successfully.

The Appointment tests ran smoothly, especially for date validation. For example:

assertThrows(IllegalArgumentException.class, () -> new Appointment("A01", pastDate, "Dentist Appointment"));

This confirmed that past dates were rejected as intended.

Alignment and Quality

My tests directly reflected the project’s requirements. Each rule—like enforcing phone number length or disallowing null IDs—was tied to at least one test case. Early failures actually strengthened quality by revealing where logic didn’t match the requirements.

By the end, most test classes achieved over 90% coverage, with ContactServiceTest and AppointmentServiceTest reaching full coverage. I kept tests descriptive and modular using @BeforeEach to reduce redundancy and maintain clarity.

Experience Writing JUnit Tests

Writing these tests helped me develop both technical precision and problem-solving discipline. The Contact test failures taught me to double-check validation logic, while the TaskTest errors reminded me to fully understand requirements before coding.

Using assertions like:

assertEquals("Homework", task.getName());

helped confirm logic correctness, while exception-based assertions ensured defensive programming. Once organized, testing became faster and more reliable.

Reflection

Testing Techniques

I primarily used unit testing and boundary testing. Unit testing ensured individual classes worked correctly in isolation, while boundary testing validated edge cases—like maximum name lengths or date limits.

Other techniques, such as integration or system testing, weren’t required for this project but would be useful in verifying service interaction and user-level functionality.

Mindset and Bias

Acting as both developer and tester requires extra caution. The Contact and Task issues reminded me not to assume my code was correct. Writing negative tests first helped me challenge my own bias and think like an external tester.

Bias can easily slip in when testing your own work. By expecting failure instead of success, I found and fixed more subtle bugs than I initially thought possible.

Commitment to Quality

This project reinforced the importance of consistency and quality. Ignoring early test failures would have created technical debt later. To avoid that, I plan to:

Maintain strong coverage with meaningful test cases.

Refactor regularly to keep code readable.

Use automated testing to prevent regressions.

In the end, effective testing isn’t just about catching bugs—it’s about building dependable, maintainable software that stands the test of time.

References

Oracle. (2023). 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.  
 Sommerville, I. (2016). Software engineering (10th ed.). Pearson Education.