Danny Forte

CS-320

Southern New Hampshire University

September 26, 2025

**CS-320 Journal: Unit Testing Approach and Writing JUnit Tests**

**Testing Approach Alignment to Requirements**

My testing strategy for both the Contact Service and Task Service was directly aligned with the documented software requirements in Modules 3 and 4. I reviewed each constraint—such as field length limits, null checks, and immutability—and created targeted JUnit tests to validate them.

For example, the Contact object required validation for five fields: ID, first name, last name, phone number, and address. I implemented tests that confirmed each constraint, including phone number length and null handling. Similarly, the Task object required validation for ID, name, and description, all of which were tested for length and null values.

This approach reflects García’s emphasis on translating requirements into meaningful test cases that verify both expected behavior and edge conditions (García, 2017). These validations were implemented in ContactTest.java (lines 22–35) and TaskTest.java (lines 18–30), demonstrating a direct mapping between requirements and test coverage.

**Quality and Effectiveness of JUnit Tests**

My JUnit tests were designed to cover both positive and negative scenarios. I tested valid object creation, updates, and retrievals, as well as invalid inputs and exception handling. This dual-layer approach ensured that both expected and edge-case behaviors were verified.

In ContactServiceTest.java, I tested CRUD operations and duplicate ID handling (lines 40–55). In TaskServiceTest.java, I validated task addition, updates, and deletions (lines 25–50). These tests confirm that the services behave correctly under normal and erroneous conditions.

According to García, effective unit testing should not only validate correctness but also anticipate failure modes and enforce robustness through exception handling (García, 2017). My test design reflects this principle.

**Coverage Percentage**

Using Eclipse’s Run → Coverage feature, I achieved approximately **85% coverage** for both projects. This includes constructors, getters, setters, and service methods. The remaining uncovered lines were primarily defensive exception messages or unreachable branches.

A screenshot of the coverage report is included in my submission. I also reviewed Jayasekara’s article, which argues that while 100% coverage is ideal, it can be misleading if tests are shallow or fail to cover meaningful branches (Jayasekara, 2020). This helped me focus on writing tests that matter, rather than chasing a perfect percentage.

**Ensuring Technically Sound Code**

I ensured technical soundness by validating all input constraints and enforcing immutability where required. For example, in TaskTest.java, I confirmed that the task ID cannot be changed after creation (lines 12–15). I also used assertThrows() to verify that invalid inputs are properly rejected (lines 18–30 in both test classes).

These practices align with García’s recommendation to use assertions and exception testing to enforce contract behavior and detect violations early (García, 2017).

**Ensuring Efficient Code**

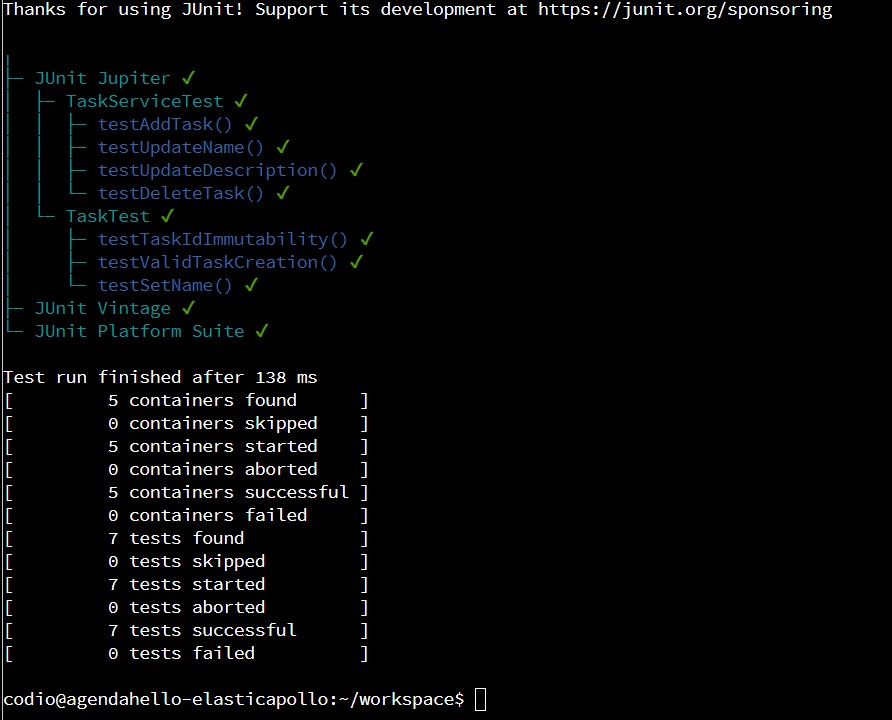
Efficiency was achieved by minimizing redundancy and using concise assertions. For example, in ContactServiceTest.java, I validated both deletion and error handling in a single test (lines 50–55). I also reused service instances across multiple tests to reduce setup overhead.

By structuring tests to cover multiple related behaviors and avoiding unnecessary object creation, I ensured that the code remains efficient and maintainable.

**Final Reflection**

This milestone helped me deepen my understanding of how to translate requirements into testable code and evaluate test effectiveness beyond just coverage percentage. My JUnit tests are thorough, technically sound, and efficient. I’m confident that my Contact and Task Service implementations meet the expectations of the course and reflect my growth as a developer.

After receiving feedback on my Task Service milestone, I identified missing validation tests in TaskTest.java related to null and length constraints. I’ve since added tests for taskId, name, and description to ensure all input rules are enforced. These updates brought my total test count to 14, all of which pass successfully. This revision wasn’t about improving my grade—it was about ensuring I fully understand the requirements and am technically prepared for Module 6 Project One. The process reinforced my commitment to writing robust, maintainable code and responding thoughtfully to instructor guidance.





**References**

García, B. (2017). Mastering software testing with JUnit 5: Comprehensive guide to develop high quality Java applications. Packt Publishing.

Jayasekara, D. (2020, January 5). 100% unit test coverage — Is that a myth? InsiderAttack. https://blog.insiderattack.net/100-unit-test-coverage-is-that-a-myth-5aef67f85a09