# CS 320 Project Two: Summary and Reflections Report

Author: Adryan Knight

Role: Software Engineer, Grand Strand Systems

## 1. Summary

### 1.a. Unit Testing Approach for Each Feature

Contacts service.  
For the contact feature, I treated requirements as black-box acceptance criteria and then translated each into white-box JUnit tests. I created equivalence partitions for valid/invalid names, phone numbers, and IDs, plus boundary value tests for field length constraints (e.g., 10-digit phone, 10-character ID, max-length first/last names). I also added state-based tests to verify repository mutations (create, update, delete) and exception tests to ensure invalid inputs fail fast with clear messages.  
  
Tasks service.  
Testing focused on life-cycle operations (create, read, update, delete) and constraints (non-null description, maximum description length, immutable task ID). I used parameterized tests to cover multiple valid/invalid inputs with minimal duplication and arrange-act-assert structure to clarify intent. I validated idempotency of updates (unchanged fields remain the same) and verified that error paths never mutate persisted state.  
  
Appointments service.  
For appointments, I emphasized temporal rules: start times cannot be null or in the past, an appointment ID must be unique, and descriptions have length limits. I added clock abstraction (injectable time source) to make “past vs. future” deterministic and used boundary tests around “now” (e.g., now.minusNanos(1) should fail; now.plusSeconds(1) should pass). I also checked that overlapping logic (if present) rejects conflicting appointments or documents the allowed behavior.  
  
Alignment to Requirements  
Across all three services, test names mirrored requirement language (e.g., addContact\_rejectsNullPhone()) so each test mapped 1:1 to a stated rule. This traceability ensured that any failing test immediately indicated which requirement was at risk. I also grouped tests by feature folder to keep requirements → tests → implementation aligned.  
  
Effectiveness (Coverage)  
Using JaCoCo, I measured both line and branch coverage after completing the test suite:  
Overall line coverage: ~92%  
Overall branch coverage: ~84%  
Coverage is not the sole indicator of quality, but it confirms that all core paths and most edge cases are exercised. Branch deltas came primarily from defensive guards in constructors and unreachable default cases.

### 1.b. Experience Writing the JUnit Tests

I wrote tests to assert behavior rather than implementation details, keeping them resilient to refactors. Examples include:

assertThrows(IllegalArgumentException.class, () -> contactService.add(null, "Amy", "Jones", "5551234567"));

String maxDesc = "x".repeat(50); assertDoesNotThrow(() -> taskService.add("ID10", "Title", maxDesc));

taskService.update("ID10", "New Title", "Updated"); assertEquals("Updated", taskService.find("ID10").description());

These tests assert required outcomes, verify that invalid inputs do not mutate state, and confirm that exceptions carry the correct type.

## 2. Reflection

Techniques Employed:  
- Equivalence partitioning & boundary value analysis  
- Black-box and white-box testing  
- Parameterized and exception testing  
- State-based verification  
  
Techniques Not Used:  
- Performance/load testing  
- Mutation testing  
- Property-based testing  
  
Mindset:  
I adopted a fail-fast and prove-it mindset: assume inputs are adversarial, validate everything, and write tests that demonstrate correct rejection. I limited bias by writing negative tests first and using descriptive test names to ensure clarity. Discipline came from frequent test execution, independence, and consistent coverage verification.

## Conclusion

This project reinforced that effective unit testing is a design activity, not an afterthought. By mapping each requirement to an explicit test, emphasizing boundaries and invalid inputs, and keeping tests fast and readable, I achieved strong coverage and high confidence in correctness.

## References

Beck, K. (2002). Test-Driven Development: By Example. Addison-Wesley.

JUnit 5 Team. (n.d.). JUnit 5 User Guide. https://junit.org/junit5/docs/current/user-guide/

Kaner, C., Bach, J., & Pettichord, B. (2002). Lessons Learned in Software Testing. Wiley.