**SUMMARY AND REFLECTIONS REPORT**

Grand Strand Systems – Project One Follow‑Up

Author: Daniel Smith

Date: 8/23/2025

**SUMMARY**

**Unit Testing Approach for Each Feature**

* Contact Service – Used functional unit testing to validate CRUD operations and enforce field constraints. Tested valid and invalid inputs, such as rejecting phone numbers longer than 10 digits, in alignment with the requirement that all phone numbers must be exactly 10 digits (Southern New Hampshire University [SNHU], 2025).
* Task Service – Applied boundary value analysis for title and description lengths. Verified that titles over 20 characters and descriptions over 50 characters were rejected, matching the requirement for strict character limits (SNHU, 2025). Ensured updates did not alter unique IDs.
* Appointment Service – Used equivalence partitioning to test valid and invalid date/time inputs. Confirmed that past dates were rejected and null descriptions were not accepted, meeting the requirement for valid future dates and non‑null descriptions (SNHU, 2025).

Alignment to Software Requirements Each test case was derived from a specific acceptance criterion in the project specification (SNHU, 2025). For example, the requirement that contact IDs must be unique and immutable was validated by attempting to modify an existing contact’s ID and confirming no change occurred.

Quality of JUnit Tests Achieved over 90% code coverage across all service classes. Coverage included both normal and exception‑handling paths, ensuring robustness. Tests validated both successful operations and error handling (OWASP, 2023).

Experience Writing JUnit Tests Tests were isolated and independent. Example from ContactServiceTest:

Code

ContactService service = new ContactService();

service.addContact(new Contact("001", "John", "Doe", "1234567890", "123 Main St"));

assertEquals("John", service.getContact("001").getFirstName());

This ensured a clean state for each test.

Ensuring Technical Soundness Validated expected outputs and side effects. Example from TaskServiceTest:

Code

assertThrows(IllegalArgumentException.class, () -> {

service.updateTaskTitle("001", "This title is way too long to be valid");

});

Confirmed invalid inputs were rejected without altering stored data.

Ensuring Efficiency Used @BeforeEach to initialize services and test data, reducing redundancy:

Code

@BeforeEach

void setUp() {

service = new TaskService();

service.addTask(new Task("001", "Title", "Description"));

}

**REFLECTION**

**Testing Techniques Used**

* Boundary Value Analysis (BVA) – Tested inputs at the edge of acceptable ranges (Myers et al., 2011).
* Equivalence Partitioning (EP) – Grouped inputs into valid and invalid sets to reduce redundant tests (Myers et al., 2011).
* Negative Testing – Provided invalid inputs to confirm proper error handling (OWASP, 2023).

**Testing Techniques Not Used**

* Integration Testing – Would validate interactions between services and the mobile app UI.
* Performance Testing – Relevant for high‑traffic systems but not for this small‑scale service.
* Security Testing – Beyond input validation, penetration or fuzz testing was not in scope.

**Practical Uses and Implications**

* BVA and EP are ideal for validating strict input constraints in form‑driven applications.
* Negative testing is essential in safety‑critical systems.
* Integration testing is critical when multiple services must work together.
* Performance testing is vital for systems with high request volumes.

Mindset as a Software Tester Approached with caution and thoroughness, recognizing interdependencies. For example, updating a contact’s phone number could affect appointment reminders if not handled correctly.

Limiting Bias Wrote tests before finalizing implementation to ensure they were based on requirements, not on the code’s current behavior. This reduced the risk of overlooking edge cases.

Commitment to Quality Avoiding shortcuts prevents technical debt. For example, skipping date validation tests could allow invalid appointments into the system. To avoid technical debt, I will:

* Maintain high coverage for positive and negative cases.
* Review and refactor tests alongside production code.
* Use automated tools like static analyzers and dependency checkers (OWASP, 2023).

Conclusion Effective unit testing is about aligning tests to requirements, anticipating failure modes, and maintaining discipline. By applying BVA, EP, and negative testing, I delivered a robust test suite that supports the long‑term reliability of the customer’s mobile application.

References

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). Wiley.

OWASP. (2023). *Testing guide*. Open Worldwide Application Security Project. https://owasp.org/www-project-web-security-testing-guide/

Southern New Hampshire University. (2025). *Project Two guidelines and rubric – CS‑320 Software Test, Automation QA*.