Project Two

Filip Arghir

Southern New Hampshire University

CS320

Kalysa Wilson

10/15/2025

For this project, the development and testing of the mobile applications, I implemented three service features. ContactService, TaskService, and AppointmentService. Each of these services handled its respective core operations for creating, updating, and deleting records, and the main goal of testing was to ensure that these operations behaved properly and robustly. For ContactService testing, I focused more on CRUD operations and ensured strict adherence to data integrity, validation of proper data creation, input constraints, and update operations. JUnit tests such as testAddContact, testDeleteContact, and testUpdateFirstName for proper contact addition, as well as updating and removal. Each test verified constraints for the maximum length of names, proper phone number formats, and contact IDs. I implemented systematic boundary testing with 10-character limits for the digit requirements of phone numbers and proper first and last name fields.

For TaskService, my testing focused on task management operations and comprehensive data validation and provided its own challenges. I used the testAddDuplicateId test to ensure that duplicate task identifiers would throw an exception and used testUpdateTaskName as well as testUpdateTaskDescription to confirm and update fields without affecting the immutable properties. Using 20-character name limits and 50-character constraints to be properly mapped and correspond with each functional requirement for proper alignment between both testing and specifications.

The third service, AppointmentService JUnit tests, was used for validation of time-based logic and to ensure proper validation of appointment IDs, description, and future dates. I did this with my testAddAppointment function and verified that appointments could not be scheduled in the past, and ensured that testAddDuplicateAppointmentId was implemented for unique identifiers for said appointments. The testing methodology addressed the requirements that

appointment dates should not be set in the past, nor should they conflict, which then leads to multiple test scenarios covering past dates, current timestamps, and future scheduling. The 50-character description limit was also tested and validated.

My testing approach, I would say, aligned with the software's requirements by validating each of the proper constraints and behaviors specified in the documentation. The contact class required all identifiers and names to be non-null and limited to proper lengths, and the proper tests were implemented to verify the requirements. The task requirements also asked for immutability and task ID, and character limits for the name and description fields were verified through the testTaskIdNotUpdatable and testSetDescriptionTooLong. The appointment feature also explicitly required that appointment dates be set in the future and validated using the dynamically generated future Date objects in testAddAppointment.

The code and quality of my JUnit tests are comprehensive, with their coverage and a great emphasis on edge cases and validation logic. My coverage analysis in my IDE tests achieved 93% coverage across all three modules. It was 92% for task service, 93% for appointment service, and 95% for contact service. Each class was tested for both valid and invalid inputs, and the proper exceptions were thrown when appropriate. For Task tests, I had an assertthrowsIllegalArgumentException.class () -> { taskService.updateTaskName("12345", null);}); that confirmed validation logic was robust and functioning correctly. I used assertions and exceptions for testing and helped the system behave as expected under all inputs.

My experience with writing JUnit tests gave me a better and stronger understanding of how to think and code like a developer and tester. Technical soundness and completing requirements were at the core of my design. I ensured technical soundness through precondition tests and had a TaskServiceTest that is called taskService.addTask, and before testing deletion or

updates, it ensured that I was operating on initialized data. Efficiency was also maintained by reusing the setup Code before each annotation. I also had helper methods that reduced duplication and maintained test clarity. Having parameterized tests where appropriate helped validate similar scenarios without the need for code repetition.

I also maintained efficiency by ensuring that all tests were independent; therefore, one test's success or even failure wouldn't affect the other tests. This was consistent and repeatable with results each time the suite was executed. I also organized my test data to be as minimal but also in-depth as possible, which ensured proper focus for each case on its specific validation and behavior.

For this project, I primarily used unit testing with boundary value analysis and equivalence to validate inputs, and I was able to check that each module behaved correctly. Unit testing allowed me to isolate each class, and while I performed boundary and equivalence testing, I confirmed that the proper input constraints, character limits, and null validations were enforced. These methods were efficient for confirming that each service functioned reliably on its own.

If I were to expand into a larger application, integration and system testing would be my most likely methods to validate interactions between modules and proper workflow. Integration tests would help check that the changes in one service don't affect others, and system testing can help simulate real user operations.

My mindset definitely changed during the course of this project. Thinking about input validation, immutability, and exception handling are all areas that I interact with on a daily basis when it comes to working on custom-built PCs that have issues, or even laptops, as well as the system we have built, so that the end user may not use the system properly. I was able to view

my code more objectively and see the weaknesses and really assess any issues that I found early. I would highly recommend that avoiding bias or a deep connection to the code is a way for me to see my flaws. Approaching testing as if I were reviewing a classmate's code or a friend's code made me write more challenging tests and verify that invalid data was properly handled. This strategy, while a bit unique, removed the bias I typically hold when it comes to my own projects and reduced any blind spots, and improved my overall test quality and performance.

This project highly emphasized the importance of discipline, which I believe is the best way to avoid technical issues. Any incomplete or unclear tests will lead to long-term problems and maintenance as well. I focused on clarity and descriptive test names to also help with future debugging, as this could be incorporated or reviewed by others. Being able to integrate continuous testing and code reviews into my workflow is imperative. I do understand that now, as well as that simply writing code and then reviewing it isn't the solution to everything, but writing, testing, and then reviewing is a better methodology. This project helped reinforce the idea that verifying correctness is not the only thing needed for secure software, but that stability, security, and proper alignment with requirements are also needed.