**Project 2**

Jasmine Villarreal

CS 320: Software Test, Automation QA

Professor Farley

October 26th, 2025

**Summary:**

Project one was a hands-on way to practice software testing and quality assurance in a simulated development setting. The Main objective was to design and build 3 service classes: ContactService, TaskService, and AppointmentService. Each class includes its own set of JUnit tests. This project focused on creating reliable functionality, improving code efficiency, and ensuring full test coverage, which are all key concepts in the software development cycle.

The Contact feature introduced a simple structure for managing user information using a unique contact ID, first name, last name, phone number, and address. The Contact Service class handles creating, updating, and deleting contact records while still maintaining data integrity through validation checks. In these checks, it verified limited string lengths, verifying that inputs are not left null, and limited the 10-digit length for phone numbers. The JUnits for ContactService and Contact checked both valid and invalid inputs using assertions to help confirm that the exceptions were thrown. These tests followed black-box testing principles, focusing on user input and output rather than just internal logic, which helped simulate how an actual end user would possibly interact with the application.

As for the Task feature, it built the foundation for managing application tasks. The Task Class required a taskID, name, and description; no section needed to be left null or overly long. The TaskService class helped maintain a list of active tasks and included ways to add, delete, and update tasks. Testing for this section used multiple assertions to help verify that each method behaved as expected, covering normal, boundary, and invalid cases. This section of the project reinforced how defensive programming and thorough validation make applications more reliable.

The Appointment feature introduces the use of time-based logic. This helped to manage appointmentIDs, dates, and descriptions, while also validating that appointments made could not be scheduled in the past or any section left null. Unit testing targeted boundary cases, such as dates right before the current time and long descriptions. For example, one of the tests checked if a user could make an appointment for the past, which enabled the application to follow realistic scheduling rules. It considers both output results and how conditionals branch handles date and input logic.

**Reflection:**

This project offered valuable insight into what is expected in professional software testing as a Software Developer. By integrating JUnit into the development process, I experience how automation testing enforces accountability and consistency within the codebase. Each test becomes an excellent safeguard against regression and a good way to verify that the system evolved without introducing new bugs into the application. Being able to write tests alongside implementation improved code readability and helped with my confidence in my code. It encouraged me to use clean and maintainable design patterns.

I used several testing techniques throughout the project. Black-box testing was used to evaluate how each class responded to various input conditions without considering implementation. For example, when testing the addContact or addTask methods, the objective was to verify that invalid inputs would trigger exceptions rather than how those checks were written. Boundary testing was also essential for confirming enforcement of length limits on names, phone numbers, and descriptions. These tests helped to validate constraints, such as 10 characters in phone numbers or 50-letter descriptions. White-box testing was used in the debugging process, and it was used in the date validation logic, where understanding how conditional statements executed was necessary for accurate verification.

While this project used several types of testing, like black-box, white-box, and boundary testing, it did not use integration or system-level testing. Including these types of testing in future projects can help simulate real-world workflows and help ensure that multiple services work correctly together. Regression testing would also be valuable to use, as it will confirm that any changes to the application do not break existing functionality. Although these techniques are not needed for the current project scope, understanding them helped me recognize when to use different types of methods in software quality assurance.

This project helped to reinforce key concepts that are essential for effective testing. Having discipline was crucial, as even minor errors in test logic can lead to inaccurate outcomes. The approach to input validation and exception handling was vital and recognizing the importance of avoiding confirmation bias by testing invalid and edge case inputs to uncover weaknesses in the code.

It was crucial to develop a quality assurance mindset as it means that you understand that testing is about maintainability, scalability, and confidence in software behavior. By utilizing JUnit for test coverage, it illustrated how small, isolated tests can help establish reliability, leading to faster development cycles and safer deployments.

This project has prepared me for what the expectations are going to look like, with the importance of effective code and testing alongside it. I will apply these principles in my future projects by using continuous testing tools, automating regression tests, and using broader system test strategies to build up my software quality and performance.