**Analyzing and Reflecting on Unit Testing Strategies: Contact, Task, and Appointment Services**

**Joshua Torres**

Southern New Hampshire University

CS-320 Software Test, Automation QA

Instructor**: Robert Nardelli**

**August 17, 2025**

**Summary**

In Project One, I created three backend components for the application, including ContactService, TaskService, and AppointmentService. My focus during testing was to make sure that each service performed its intended functions correctly and aligned precisely with the project requirements. I began testing by isolating each unit to identify any defects early and then confirmed that each component worked as intended within the expected use cases. In ContactService, tests covered creating, reading, updating, and deleting contacts. Using the testAddContact() method, I verified that adding a contact increased the total number of contacts and that the stored information was correct (insert exact line from your JUnit code here). I also implemented tests to block duplicate entries, maintaining data integrity in line with the requirement that every contact must be unique. TaskService testing focused on adding tasks, updating task status, and removing tasks. In the testUpdateTaskStatus() method, I checked that changing a task from incomplete to complete correctly updated the system (insert exact line). The testDeleteTask() method confirmed that tasks were deleted appropriately and that remaining tasks were unaffected. These tests ensured the service satisfied the requirement of accurate task management. AppointmentService testing involved scheduling appointments, retrieving them, and preventing overlaps. The testScheduleAppointment() method confirmed that new appointments were only created if no conflicts existed (insert exact line). I also tested appointment retrieval by date and participant to ensure users could manage their schedules efficiently. These tests verified that the service followed the requirement that appointments must not overlap.

The JUnit tests I wrote demonstrated a high level of quality. Coverage reports confirmed that both standard scenarios and edge cases, including duplicate contacts and scheduling conflicts, were thoroughly tested. Writing these tests reinforced the value of clear, structured test cases, reusable setup methods, and precise assertions. Using @BeforeEach allowed me to consistently prepare test data, improving efficiency and reducing repeated code. Assertions such as assertEquals(expected, actual) and assertTrue(condition) provided definitive validation for each requirement.

**Reflection**

**Testing Techniques**

I primarily relied on unit testing and boundary testing. Unit testing allowed me to evaluate each service separately to ensure it functioned correctly on its own. Boundary testing helped uncover issues at the edges of expected behavior, such as handling duplicate contacts or avoiding conflicting appointments. These techniques were effective for ensuring that each component could operate reliably on its own. This project did not involve other testing approaches such as integration testing, system testing, or stress testing. Integration testing assesses how different components work together, while system testing evaluates the complete application, including the user interface. Stress testing measures performance under heavy usage. While these techniques are important for larger applications or production environments, they were outside the scope of this assignment. Understanding these approaches, however, is valuable for applying them in future software projects when appropriate.

**Mindset**

I approached testing with care and attention to detail, recognizing that the services were connected. For example, deleting a contact could impact associated tasks or appointments, so I verified that these relationships remained consistent. I also worked to limit bias, as testing my own code can create assumptions. Running automated tests and reviewing results objectively helped ensure that gaps were not overlooked. I also considered additional scenarios beyond the initial criteria to guarantee robust and reliable services.

**Discipline**

Maintaining discipline and focusing on quality were critical throughout the project, as neglecting proper testing or shortcuts could result in technical debt and potential future defects. For instance, insufficient testing of appointment conflicts could allow double bookings. To avoid accumulating technical debt, I will maintain thorough testing practices, conduct frequent code reviews, and revise tests whenever requirements evolve. This practice ensures that my code remains reliable, maintainable, and aligned with user expectations.

**Conclusion**

Project One provided an opportunity to apply effective unit testing strategies, confirm that the services met software requirements, and develop a disciplined and cautious approach to testing. By implementing well-organized unit tests, addressing edge cases thoroughly, and reducing bias, I confirmed that ContactService, TaskService, and AppointmentService operated reliably and were prepared for deployment. This experience reinforced the importance of detail-oriented, disciplined testing practices, which are critical for success in professional software development.