**Software Testing Techniques**

In the milestones I completed for the Contact Service, Task Service, and Appointment Service, I used several software testing techniques to ensure the functionality of the programs. For the Contact Service milestone, I relied heavily on unit testing to check the functionality of individual methods. For example, I tested if invalid inputs, such as a contact ID that was too long or a phone number with incorrect formatting, would throw the correct exceptions. I also used boundary testing to make sure the program worked properly when values were at their maximum or minimum limits, like testing the longest allowable address or the shortest valid name. These tests helped confirm that the program worked as expected for a wide range of input scenarios.

In the Task Service milestone, I expanded testing to include integration testing, which checks how different parts of the program work together. For instance, I tested how the TaskService class handled adding, updating, and removing tasks. I also used error-based testing, where I purposely gave the program invalid inputs, like duplicate task IDs, to make sure it handled errors correctly. Additionally, I used state testing to check whether the task list updated properly after actions like adding or removing a task. This ensured that the service maintained accurate data throughout different operations.

For the Appointment Service milestone, I continued using unit testing to verify the behavior of individual methods and added negative testing, where I supplied incorrect values like null dates or overly long descriptions to confirm the program responded with the proper errors. I also used sequence testing, which involved checking that certain actions—such as updating or deleting an appointment—could only happen after the appointment was successfully added. These tests helped ensure the program followed the correct workflows and maintained proper functionality.

There are also some testing techniques that I did not use in these milestones. For example, regression testing focuses on making sure new changes don’t break any existing features. This is important for larger projects where features are regularly updated, but since these milestones were small and built incrementally, I didn’t need this kind of testing. Another method I didn’t use was performance testing, which checks how well a program runs under heavy loads or stress. Since the services only used simple in-memory data structures, performance testing wasn’t relevant for this project. Lastly, I didn’t perform system testing, which tests the entire application as a whole, because the milestones didn’t include a complete system or user interface.

Each of these testing techniques has its own purpose and is useful for different situations in software development. Unit testing is best for checking individual components of a program, especially in projects that focus on modular design. Integration testing is crucial for verifying that different parts of a program work together smoothly, which is important for complex systems with multiple interacting components. Boundary testing is useful for ensuring that a program handles extreme or edge-case inputs correctly, helping to avoid crashes or incorrect results. Negative testing improves a program’s reliability by confirming it handles invalid data appropriately. On the other hand, techniques like regression testing and performance testing are better suited for larger, more mature projects where maintaining stability and efficiency over time is critical. System testing is most useful when testing a finished product with all components, including user interfaces, working together.