**CS -320 Project Two: Summary and Reflections Report**

**Jason Sedgwick**

**October 20, 2025**

Throughout the development process, testing was made a priority by unit testing each component unit. For the ContactService unit, I successfully created and tested add, update, and delete operations. Boundary-value testing was applied to the IDs to make sure the limitations on the number of characters (10 or fewer) were implemented correctly. This was also done to verify that the phone numbers were input correctly and matched the required length, and names were entered. Names could not be null.

Next, I successfully created and implemented Unit testing in the Task Service component. I implemented code for equivalent partitioning tests and negative tests to check that the ID, name, and description constraints were working correctly. During these tests, I verified that invalid and valid inputs were tested and that exceptions were raised when necessary.

Finally, the AppointmentService component was tested for proper date formats and to ensure that the appointment ID, the description, and the date were compliant with all the rules put in place. Invalid and Valid conditions were tested for these fields, and these tests were completed successfully in Codio.

The JUnit tests I implemented all passed successfully in Codio, and all major methods were covered. Inputs, both valid and invalid were tested properly to ensure that no unexpected behavior would occur under different entry conditions. Boundary-value tests were also implemented, and I was able to confirm, through this testing, that exceptions were properly raised when necessary. Therefore, I am confident that the code meets the requirements and expectations of the customer. I also ensured that my code was technically sound by using clear assertions assertEquals and assertThrows to ensure the exceptions were handled correctly. My code was kept efficient by limiting code repetition and making the components easy to extend.

Throughout the testing process, as I explained in the beginning, I used several testing methods to verify the reliability and integrity of the application. Boundary-value testing was applied to test the lower and upper limits of input values for IDs and phone numbers. I used equivalence partitioning to address consistency concerns using valid and invalid input ranges. Negative testing was used to ensure that the appropriate exceptions were raised when a user input invalid data.

Other testing techniques not used were integration testing, which shows the interactions between modules. System testing is used to verify the performance of the entire product and the system. Regression testing was also not necessary for this project. Regression testing is used to see if updates cause failure in existing features. Although not used in this project, these testing techniques would most likely prove valuable later in the project development.

I briefly covered some of the practical uses for both the techniques used and not used. The unit tests, such as the boundary-value, negative, and equivalence partitioning, are all valuable tests used in early development. These tests allow a developer to test specific behavior and pinpoint specific errors. The regression, system, and integration testing are all later-stage development testing used to test a combination of all components to make sure they interact with each other effectively and efficiently.

I think as a developer, it is just as important to keep an open mindset as it is to keep oneself unbiased in their assessments of their own code. It is easy as a developer to say, “it’s not my code, it’s user error”, but if that’s the case, a developer needs to be able to back such statements up with proof. This is one reason why testing correctly, efficiently, and thoroughly should be the standard for developers. It’s important not to get lazy and skip testing, as this can help avoid legal and reputation issues down the road.

**References**

GeeksforGeeks. (2025, July 23). *Software testing – Boundary value analysis*. GeeksforGeeks. <https://www.geeksforgeeks.org/software-testing/software-testing-boundary-value-analysis/>

GeeksforGeeks. (2025, July 23). *Equivalence partitioning method*. GeeksforGeeks. <https://www.geeksforgeeks.org/software-engineering/equivalence-partitioning-method/>

GeeksforGeeks. (2025, July 23). *Negative testing in software engineering*. GeeksforGeeks. <https://www.geeksforgeeks.org/software-engineering/negative-testing-in-software-engineering/>

GeeksforGeeks. (2025, July 22). *Regression testing*. GeeksforGeeks. <https://www.geeksforgeeks.org/software-engineering/software-engineering-regression-testing/>

GeeksforGeeks. (2025, July 20). *System testing*. GeeksforGeeks. <https://www.geeksforgeeks.org/software-testing/system-testing/>

GeeksforGeeks. (2025, July 20). *Integration testing*. GeeksforGeeks. <https://www.geeksforgeeks.org/software-testing/software-engineering-integration-testing/>