The testing approach aligned with the software requirements by testing the requirement cases. Making sure that any Ids, Names, Descriptions were no longer than the available characters and not null. Along with testing to make sure adding, deleting and updating all worked correctly. For example, the Phone number was 10 characters long and not null, and the address was no longer than 30 characters and not null. I know the JUnit test was effective by running coverage and checking to see if all cases had been tested. I ensured the code was technically sound by reviewing it and running JUnit tests. I ensured my code was efficient by using a hashmap to store the data.

The software testing techniques used were Unit Testing: Testing individual components by testing the constructor class, adding and deleting to ensure it initializes correctly with valid input. Boundary Testing: Verifying edge cases for input values by Checking the maximum length of appointmentId, name, and description fields and dates are not in the past. Exception Testing: Ensuring appropriate exceptions are thrown for invalid inputs by ensuring that the constructor throws IllegalArgumentException for invalid inputs, such as null values or invalid lengths. Integration Testing: Checking interaction between Appointment and AppointmentService. State-Based Testing: Verifying the state of objects after operations by checking the appointments map after adding or deleting an appointment to ensure it reflects the expected changes. Negative Testing: Ensuring the system handles invalid conditions correctly by testing addAppointment and deleteAppointment throws an exception when trying to add an appointment with an invalid ID.

Some of the software testing techniques not used were Black-Box Testing: Testing without knowledge of the internal workings of the application. Tests are based on input and expected output. White-Box Testing: Testing with knowledge of the internal code structure. It involves testing internal paths, conditions, and loops. Regression Testing: Testing existing software applications to ensure that new changes have not introduced any new faults. Stress Testing: Testing the software under extreme conditions to ensure it can handle high levels of stress without breaking.

Caution is needed to ensure that all possible cases and inputs were covered. This included typical cases, and invalid inputs to guarantee the robustness of the software. The complexity and interrelationships of the code was crucial for understanding how different components interacted. This helped in identifying potential areas where a change in one part could affect others. Understanding these interdependencies was vital for effective impact analysis, ensuring that changes did not introduce unexpected side effects elsewhere in the system. As a developer, it’s easy to become overconfident in your own code, potentially overlooking flaws because you believe in the correctness of your code. Familiarity with the code might lead to blind spots, where you miss issues that a different tester might catch. It is important to be disciplined and not cut corners because it improves the readability, maintainability, and scalability of the software. Technical debt can be avoided by regularly reviewing and testing code, thorough documentation, and well detailed planning and design.