Project Two Report

Unit Testing Approach: For the Appointment feature, I used boundary testing to validate the correctness of date and times. This helped me ensure that past dates and unrealistic time intervals were caught. For the contact feature, equivalence class partitioning helped me choose between valid and invalid phone numbers. For my notification testing, I used state transitioning tests to ensure the notifications were only triggered under the correct conditions.

To show alignment with software requirements, such as handing edge cases, I ran tests to make sure that our Appointment class was handling future dates correctly. Specifically, through my testInvalidAppointmentDate method.

Most of the code was tested, about 95%. Not only the code, but the edge cases and the critical paths were tested to help with coverage. This also helped to make sure the software ran correctly in diverse conditions.

To make sure that the code was technically sound, error messages were reviewed and test case failures were also reviewed. For instance, when testInvalidContactID was used to give “assertThrows(IllegalArgumentException.class” when an invalid input was given.

To ensure efficiency, I combined multiple related assertions within one test method to keep it compressed and easy to read. For example in ‘testInvalidLastName’ we checked for both null values and excessively long names. This was done to optimize test times. Uncompressed code is time wasting and exhaustive for a developer to read and test.

Testing Techniques: The main software testing techniques used were boundary testing, equivalence class partitioning, and state transition testing. These were chosen based on the nature of the features, boundary testing, equivalence classes, and state transition for features. Boundary testing was used for date/time validations. Equivalence classes were used for fields such as phone numbers, and state transition for features like notifications. I did not use methods such as decision table testing or pairwise testing.

Mindset: Adopting a software tester’s mindset was crucial. I tried to always anticipate where the software could and sometimes would fail. Understanding the intricate relationships between classes is crucial to understand and write good tests that do a good job with coverage. For example, Contact and Appointment being very intertwined in theory, helped me complete and understand how vital it is to make sure both works flawlessly before combining. Bias in code review is real. When it is being written, there is a innate bias in each coder. This can blind some to logical or syntactical errors. For instance, I initially assumed my Appointment date logic and code was sound, but when I tested it was shown to be incorrect and I had to spend more time on it. Being disciplined in commitment to quality is not negotiable. Cutting corners might provide short term gains and save time, but in the long run, errors will grow and accrue technical debt, leading to costlier fixes in terms of time and in resources. Moving forward, I tend to keep a test-driven development approach to make sure tests guide the development process. This also makes sure that technical debt is not stacked on top of itself, and that time and energy is not wasted. I could have done it more efficiently for sure, but as a newer developer the best thing I can do is even test the simple methods and classes to make sure that they run correctly and that when I create more, the program does not collapse upon itself. When a person is hired and finances are on the line, this is very important. Software tends to cause issues around the edges, so boundary testing and having a mindset to do that is very crucial to catch mistakes. For my Appointment class for example, testing the times and dates at the end of acceptable values helped me make sure that the errors and exceptions were thrown correctly. Keeping a mindset to grow and learn new techniques is important. Also, not giving in to hubris and not overlooking simpler code, no matter how long a person has been in the business. You would not put a baby in a baby seat in a car if you have not tested the seatbelt first.