**What were the software testing techniques that you employed for each of the milestones? Describe their characteristics using specific details.**

The main technique I applied was decision testing, a form of white box testing. Since we are trying to generate code coverage to certify that the specified use cases the software is meant to handle do work, we are using the structure of the project to evaluate it, making it white box testing (Hambling et al., 2019, pp. 115-116). Each week I made sure that every decision tree in the code was covered correctly, such that any IF statement or other decision was followed, meaning both incorrect and correct inputs were tested. It is similar to testing the function of a transistor in a few ways, as we know when the transistor should switch a signal on and off, as well as when the transistor should be overloaded from too high voltage and when it should fail at too low voltage, all of which can be tested.

Past the first milestone I also began to examine static testing on the design documents (rubrics). The first large error I noticed was that there were no cases specified outside of the phone inputs where the input would be rejected for being too short. This is contrary to my own experience, as any user may accidentally hit the enter key at any time, regardless of technical skill. AppointmentService also had no listing for how long its ID was meant to be, but the implication was that it should follow the same ten-character limit the others had, so I implemented this.

Finally, I also used equivalence partitioning and boundary testing to test the inputs for my functions. Since the If statements should fire on certain integer values for their size, I supplied them with counts just above and just below their designed range to test the two rejected partition ranges as well as a correctly sized input to verify it was not rejecting all partitions.

**What are the other software testing techniques that you did not use for the milestones? Describe their characteristics using specific details.**

I did not use black box testing in any of the milestones. As the person who wrote the code, I was unsure how to implement this kind of testing while uncovering new potential errors in the code for this project. I was already getting the desired outputs, and my white box testing showed that all the code was correctly executing on aberrant and routine inputs. Black box testing, however, seems to be one of the greatest tools for testing someone else’s code quickly. It may be very time consuming to evaluate every line of code, and these tests can more easily be written in advance.

**For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**

Decision table testing is from designing a truth table and assigning truth values to them to design test cases (Hambling et al., 2019, Ch. 4). Use case testing originates from designing a user and a use case for that user, then checking the output is what that user would expect. White box testing is knowing the internal structure of the program, then using that in your testing scheme. Black box testing istesting the output of a program from just its input treating the code like a black box. Equivalence partitioning is the act of deliberately segmenting different inputs into ranges based on the expected values that you may encounter, such that they cover the range of “too low” to “too high” with a set of cases that cover “just right”, as working cases in those ranges imply all the other ranges are working as intended. Boundary value testing is an extension of partition logic, picking test cases that exist at the edges of the partitions.

References

Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing: An ISTQB-BCS Certified Tester Foundation Guide* (B. Hambling, Ed.; 4th ed.). BCS Learning and Development Ltd.