Project 2 Document

The testing approach aligned with the software requirements as it was able to test features implemented into the base of the code for the application and not something that could only be tested in a soft deployment or a final deployment of the application. Some examples of this is the hard coded limits on the length of first and last name, the task id and the address are validated to help prevent inputs that are too long that could cause errors.

The quality of the JUnit tests that continued to improve throughout the modules as I had similar coverage in the final testing of both parts of the final project but did continuously improve throughout the testing of the project.

Something that I used to help make the code function as intended was to have intentional use of in code testing such as the limits on all used inputs to help ensure that the code could not be overwhelmed by inputs that are unreasonably long which could potentially cause errors to occur in the code.

One tactic I used to help ensure that the code was kept efficient was to help and cease running if there are found errors by breaking the running code when an input was invalid to help prevent unnecessary processing being used when we know that the program will fail from a certain point forward based on a given error.

The main technique that I implemented during development was functional testing to help ensure that all of the individual parts of the overall code was working properly to then better test when everything was placed together.

Some testing techniques that I did not implement were more experienced-based techniques helping to better stress test sections of codes by putting them through inputs or actions that can be commonly used to break a software for malicious purposes that can then have counter measures created to help defend against.

The mindset that I adopted when working on the parts of the project was to make a conscious effort to continuously test the software throughout the entire development process and to go back and improve the code after a first test to better improve the coverage percentage from the JUnit tests

To help limit bias in the process of reviewing my code I would test individual sections of the code to help ensure that things would act as intended as they were implemented instead of just creating a section of code and trusting that it would work regardless of how simple or complex that the section of code was that was being implemented. One example I came across during development was with the validation of the length of the inputs to limit their length, in my original implementation of the validation it would only accept inputs that were longer than the maximum length which would have been easy to miss if only testing with JUnit and just assuming that I correctly coded the input validation as it was a simple requirement only needing to check length.

Being disciplined during the development of software to better ensure that all requirements that are provided and outlined by the client and, in cases like this, to ensure that all the data being given to and taken from users is safely and properly handled to help prevent leaking of private and sensitive data. Another thing that this will help with is in the overall development in the code to make sure that there is little to no unnecessary processing in code to help optimize the software and to better write code so that if errors are found that they are easy to identify and solve as to avoid having to do large amounts of debugging and testing.