Ultimately, this project provided a strong basis in strengthening understanding and exercising Requirements Engineering, System Modeling, Architectural Design, Specification, Implementation, and Software Testing. Additionally, the implementation of a Dependable Attribute, a Project Management concept, and Advanced SWE topics helped to further strengthen understanding of these topics.

The first sprint partially consisted of defining the functional and non-functional requirements, referring to Requirements Engineering. The functional requirements were to create a hierarchy of objects, define the structure of the project including the classes and interactions of such, develop data communication methods and methods to control the movements of the buses including start and stop times as well as the speed of the routes, and to allow the routes to be initialized via text file. The non-functional requirements were scalability and flexibility, in that the program needs to work with different amounts of routes, buses and passengers, and that the program can handle changes in the requirements like different time constraints.

Additionally, in the first sprint, System Modeling was utilized, and an abstract model of the system was developed, using a sequence diagram and activity model. Here, the different interfaces, which include text input, bus overview, and bus controller, were highlighted, as well as the database that dictates the operations of the bus routes. This was implemented by taking requests from the simulator and utilizing an algorithm to realize the activity of the buses, as well as considering location for more precise time estimates.

The Architectural Design emphasized two classes for the operation of the system, those being the Bus Route Overview and the Bus Controller. The Bus Route Overview stores a large amount of raw data about the routes and buses, including their locations, routes, and operating status, as well as the bus's identification. The text input is also included here for testing. A large emphasis was put on the ability for this class's compatibility with inputs of varying size, so that scalability is maintained. The Bus Controller contains the algorithms in charge of managing the bus system, including the speed and frequency of the buses to ensure their timely arrival/departure. Many techniques must be utilized as this is the part of the project with the highest potential for error, due to its difficulty.

Relating to Specification and Implementation, we can see that the Bus Overview is the class that holds the objects and essentially dictates the structure of the objects buses, routes, and stops. A Case Description was utilized for implementation as shown above.

Software Testing for Sprint 1included manual input from a text file since there was no user input at this point, and the input is tested simply with an output to the console. A GUI was

not utilized because the results and their process in the algorithm were printed to the console, to allow for testing in a functional manner.

Overall, Sprint 1 went mostly according to plan, other than a few minor changes needing to be implemented, like the way that distances were calculated; this switched from a doubly-linked list of times that looped to a distance matrix holding a key of the times to each stop and utilize the shortest route based on that. Additionally, this allowed for smaller keys to be centralized into one object. The goal from here for the next sprint was to add the simulator which will generate requests from passengers, allowing for an accurate assessment of the system's handling of a constant amount of passengers.

For Sprint 2, the Requirements Engineering included functional requirements of creating a class for bus simulation, creating methods to test a simulated passenger load, implementing a progress bar to indicate the locations of the buses, and implementing worst case scenarios to ensure timely passenger arrival. The non-functional requirements include scalability, efficiency, and documenting the simulation.

In regards to the System Modeling of Sprint 2, the simulator was added to generate transport requests and passive bus movement, which allowed more realistic testing of the program. Since the simulator is just that and not real people, it uses deltatime to change location based on speed. With this addition, the system is able to process multiple requests in a row because of the passive bus movement.

Architectural Design is affected by the addition of the simulation by the information about bus locations and requests being handled by the simulator instead. The most significant change is that the simulator generates input, instead a manual text input.

For the specification of Sprint 2, the main difference from Sprint 1 is that the system now sends information about requests and bus locations to the bus controller, and the simulator provides the data for the bus drivers and potential passengers. The Implementation is shown in a Case Description above.

The Software Testing in Sprint 2 involved importing routes from the text file, verifying that it was in the program using the console, then reading a manual query and making observations of its handling, by checking the times and locations of the buses. This was further verified by using a constant refresh rate to simulate time passage while continuously verifying its functionality.

Overall, Sprint 2 was successful as the simulation implementation had very little error. The only issue being the "real time" simulation, which was difficult to test as it would take a wastefully large amount of time, so the time scale method was utilized with a 2 minute refresh rate.

The Requirements Engineering for Sprint 3 included functional requirements to implement exception handling, implement a minimal GUI to to display refreshed results, handle incorrect text input, and to display an error if the passenger takes longer than 30 minutes. The non-functional requirements were usability, effectiveness, reliability, and fault tolerance, meaning the program needs to have an intuitive way to interact with, the program must be able to always produce a result for the given operations, the program should be optimized to it's best with a low average of system faults, and that the program can identify and correct errors with the least possible effect on functionality, respectively.

The System Modeling for Sprint 3 introduced the addition of the GUI so users can make their own requests for buses, and refresh the system to find bus times and locations and other bus data.
The addition of exception handling here facilitates the identification of faults and allows for correction of the faults in the program, and errors will be shown in the console for administrators.

The Architectural Design of Sprint 3 compared to Sprint 2 sees the addition of the GUI and exception handler added in the bus controller system, which allows errors to be seen and addressed by administrators.

For the Specification of Sprint 3, there is only two main differences, those being the implementation of the GUI, which means the user checks their bus times and requests through the GUI instead of the console, and the exception handling subsystem, which enables errors to be sent to the administrator to be mitigated. The Implementation for Sprint 3 is emphasized with a Case Description above.

Software Testing was utilized by observing the simulator and its operation status through the console and the GUI. Additionally, bugged code was added to produce an error within a try-catch statement. The GUI was verified by comparing the displayed data with the information that was manually written in the text file.

Overall, Sprint 3 went mostly according to plan with very small deviation from the original plan. By adding exceptional handling it was concluded that the system will run with non-fatal errors. Additionally, the GUI was implemented with less than desired functionality, but still serves its purpose as intended for the scope of the project. The system operates fine under a large load, even with the large amount of commands in the program.