*What went wrong?*

One of the largest issues we ran into was not completely following our designed modularization for the system. From the start we planned on separating the system into front and back end to better focus on the issues of CSV processing and display, opting to have references to data passed between them as to avoid any processing on the front-end. However, as coding went on, the teams became rather disconnected. Combined with a time crunch due to other commitments team members had, the disconnection forced us to incorporate some processing in the front-end in order to make deadlines meet, thus defeating the purpose of separating the two systems.

Another issue we ran into was properly building the program with QT and on different operating systems. As team members used different operating systems and versions of QT to build the project, making sure all necessary tools were on both systems and on the same version when building and compiling the product caused a lot of headaches and slowed progress down quite a bit.

The final major issue we ran into came about due to a lack of automated testing (ie Jenkins). As many of us coded in on our own time rather than at a set time with the team, it was made it difficult to tell if pushed code was breaking the system due to recent changes. Many times some team members would attempt to push their code, only to find unexpected changes to the repository since they last pulled. This caused some confusion with version control and resulted in a few code breaking pushes.

The three lessons that came about from these challenges are:

1. When separating a project into different teams, communication across them is vital
2. Make sure developers are using either an automated build process or the similar environments
3. Automated testing is a life and time saver

*System Documenting*

The majority of the system design was done at the start of the project, well in advance of any coding. Before coding, we designed the system by breaking up the front and back ends as well as agreeing upon how to send data between the two.

Comparing the project to previous projects in which members of the team documented the design after development, creating the ACUITY STAR program was much easier in terms of organization and vision. Having such a clear image of how the system should look and interact with other components made it much easier to split up tasks and code each to specification, rather than justifying code after development. It took a lot of guesswork out of the mix, allowing us to focus on things such as efficiency rather than worrying about how each component needs to interact or what the final product should look like.

*Analysis of Teamwork*

Overall, we feel our team was well organized and delivered a great final product to the customer. While there was some disconnection between the front and back end teams, there were no major issues with team contributions or organization.

With that said, there were still some minor issues such as attendance for meetings or communication across the team. As we are a large team of busy students, issues with meeting attendance are to be expected since we aren’t all free at the exact same times. It was also an issue near the end of the project getting everybody up to speed with the project as we were all busy with other courses and preparing for finals. While we did have an agreed upon meeting time and day, the attendance for this meeting slowly dropped off until the meetings no longer took place. This had an adverse effect on deadlines, causing a time crunch near major milestones.

In retrospect, the team could have been better organized if the weekly meetings were made mandatory and enforced more by the team managers. However, in a university setting, such things are unavoidable and we do not feel these minor issues had any major impact on our project performance.

*Customer Interaction*

Outside of a few questions near the start of the project, we did not interact much with our customer. While infrequent, our communications with the customer were usually very smooth and professional. The outlier to this, however, is the issue we had with our stage 2 submission. Due to some of the problems mentioned before with building the project, the client was missing a required .dll to compile and run our code. The issue was further worsened by our very slow response to the customer when they asked us to help fix the problem. While the problem was resolved eventually, it was a rather poor interaction we had with the customer from a customer management stance.

*Recommendations for Future Students*

Start early, work steadily, and never underestimate the value of a good design. The majority of our issues stemmed from not working steadily on the project over the semester, instead doing the majority of work close to deadlines. As well, one of the greatest assets our group had was our system design we laid out from the start. Having a thorough and clear image of what the final product should be makes it much easier to organize and build such a large project.

*Value of This Project*

This project does a great job of highlighting the importance of design in software development, as well as giving practical experience in collaboration with other developers. Attempting a complex project such as this by yourself is a very difficult task, but tackling the project with others without a clear design would be chaos. Further, the soft skills learned in terms of communication, team management, and simply finding an area of a project to add value to are skills that translate to all areas of life.