**Post Work Documentation of the Work Process**

On the first day, the work was split into three from the get-go. Ze Xuan would decipher iText and develop a parser for the PDF, Richard would develop the MySQL database and Ivan would develop the GUI. The VCS program used for the project was GitHub, due to previous experience with it.

Having had no experience with iText, we scoured the internet for resources and tutorials on how exactly to use iText to parse pdf files. This proved to be hard as we had no idea that Acrofields existed and could be easily separated from the rest of the pdf's content. In fact the original plan was to write a parser to get the data, but thankfully we chance upon the idea of Acrofields. We became more immersed in the wonderful online world of free resources after looking through places like google code and stackoverflow, which were really helpful in ironing out the quirks in our mini-project.

As for the development of the GUI, the original plan was to have it all in one Java file. However, having just learnt MVC, we chose to adopt an MVC-centric approach when coding it. This led to the GUI being split into two classes SSEFInterface (Model) and SSEFView (View and Controller). While not a pure MVC structure, the concept remained.

A model should control all sides of the information processing process. As such, within SSEFInterface were the database interface and the pdfParser. SSEFInterface also had methods to check if the SSEF database existed or the user was logged in, to communicate this information to any Views that might be using it. The model classes took a painfully long time to produce due to the multitude of fields that we had to consider. While the logic might had looked straightforward on the code itself, we had to take into account various ways the data could have presented itself, and attend to them accordingly. Otherwise, coding for the model was easy once the architecture of the model had been thought through.

This leads us to SSEFView, which was not so much a View as it was a Controller, serving as a means for the user to input their file and SSEF Code. This information would then be passed to the SSEFInterface to be processed.

With all that’s been said, however, that was only the construction of the individual modules. As it turns out, pulling the three parts of the project took a lot more effort than we expected. Not only did it take awhile to learn how each other’s codes worked, but when the classes finally did work together, it was often in incredibly inefficient ways. We literally used parse a file into a string, output that string into a text file to pass into another Scanner to turn into an SQL query. Valuable time and resources were wasted streamlining the file processing process. For a mini-project, we only used about two hours to fix this problem, but this project was a small program. How much more time would we have wasted, should this style of working translate to a larger project?

In the future, we would probably plan out our individual modules together first to ensure that we all knew what each other’s code did. This way, we would minimize time spent on trying to figure out how our codes worked and can simply abstract the code into just method headers. Also, there would be a better understanding of how the classes would work together, resulting in a more efficient flow of logic within the program that would require less optimization in later stages of programming.