Critical Reflection

The design we were given for the system had many significant flaws particularly in consistency of ideas and in security protocol. The main security issue was that when a user logged in to the token machine, their entire user account was passed through and stored locally in the machine. The presented a problem which required a large-scale redesign of the account variables and methods. For this reason, a Proxy design pattern was used to hide this, and only the account ID was returned; thus, allowing a more secure account system. There was then a function created which allowed specific data to be accessed from the account by passing through the accountID integer and a string which tells the function what it needs to return. This subsequently meant that some other functions were not used such as GetCardId as they were made obsolete by the new template function:

(GetXByAccountId from the CustomerAccount class)

public T GetXByAccountId<T>(int accountId, string x)

{

var accs = ReadFromBinaryFile<List<CustomerAccount>>(@"Accounts.txt");

           foreach (var account in accs)

           {

               if (accountId == account.\_accountId)

               {

                       switch(x){

                           case "balance":

                               return (T)Convert.ChangeType(account.\_balance, typeof(T));

                           case "cardid":

                               return (T)Convert.ChangeType(account.\_cardId, typeof(T));

                           case "fullname":

                               return (T)Convert.ChangeType(account.\_fullName, typeof(T));

                           case "username":

                               return (T)Convert.ChangeType(account.\_username, typeof(T));

                           case "paymentoptions":

                               return (T) Convert.ChangeType(account.GetSavedPaymentDigits(), typeof(T));

                           case "savedpaymentmethods":

                               return (T)Convert.ChangeType(\_savedPaymentMethods, typeof(T));

                       }

               }

           }

           return default(T);

}

Many of the classes within the class diagram were lacking basic variables which were required for the class to function correctly. Some examples of this are classes like LanguageList which needed to contain a List<Language> variable, and the scanner location becoming a Station rather than just a string. This allows the scanner to have access to a list of tickets which are valid which is crucial to the scanner’s ability to open and close the barrier that it is attached to.

Another issue with the design was the way that the payment system had been implemented. This process had been detailed inside one of the sequence diagrams.

In terms of scheduling of work and dividing of the work load the group has worked together communication via a WhatsApp group chat so that everyone stayed up-to-date. This was useful as it allowed the group to see when message had been read by all the other members and cut down significantly on many of the problems from the design assignment during the first semester resulting from inconsistent communication. The basic implementation was worked on largely as a whole group with one person coding on a large screen that everyone could see and the others discussing changes and ways of implementing the design. While this was useful as it meant that everyone was aware of what was happening with every piece of implementation, it did mean that development was often slower than it would otherwise have been if everyone was working on their own section of the implementation.

The individual code implementations were handled in a largely inefficient manner. The way it happened was that the backend and language extensions were completed alongside the development of the basic implementation and then the AdminGUI and creation of ticket validation/ barrier functions were created later. In hindsight, the backend and the ticket functionality should have been done with more urgency as these affected many other aspects of the program. Whereas the AdminGUI and languages extensions were much more surface level changes which dealt with how data was displayed.

We handled the issue of version control through a program called Tortoise SVN which works through a link with GitHub. Once this was up and running it worked largely as expected with each member of the group having their own branch to work on which was merged into the trunk at each milestone. A small hiccup with this at the beginning however, was that we were doing most of our work sessions together and in the library, which would have been fine except that the library did not have Tortoise SVN installed and when we requested it we were told that it would not be possible. Despite this, the library computers did gradually all get Tortoise installed on them which made life with version control much simpler. A side effect of the coding approach we were using and that we were unable to use Tortoise initially was that the commit logs did not accurately dictate the actual divide in work as it was always only one members account which was committed from.