**Lessons learnt**

**Coordinating group work**

It is difficult to coordinate group work, especially to share documents with each other. However, we learned that by using modern tools such as GitHub, and cloud services to share document it helps to improve team efficiency. Additionally, we found that messaging services and VoIP programs such as Facebook Messenger and Skype help to make the coordination and communication between such a large group much easier. At any time, we can easily check the progress of our project, and share information to everyone in the group. Also, we can make the group be private; therefore, these tools also keep all files and ideas in a relatively safe place.

**Problems with UML**

UML provides many kinds of diagrams to help plan and design a project. We used use-case diagrams and class diagrams in our project to show the various relations between different components. A use case diagram illustrates a unit of functionality provided by the system. The main purpose of the use-case diagram is to help development teams visualize the functional requirements of a system. Use Case Diagrams make it easy to describe and show the implemented features to the customers even if they are lacking technical knowledge. However, use-case diagrams just can describe the functional requirements of a user, but cannot do anything to describe non-functional requirements such as reliability, performance, and so on. In addition, the class diagram shows how to connect different entities (people, things, and data); in other words, it shows the static structures of the system. Therefore, when a programmer wants to change the code, it is figure out which file should be modified. However, for developers who do not understand UML the design can be difficult to understand at first glance. Because of this, when our group tried to make our UML diagrams, we need to search for a lot of information from the web, and the Fowler's book on UML.

**Working with QT and C++**

While some of our group already had experience with C++ going into the project learning the ins and outs of the Qt API and Qt Standard Library took a while. But in learning how to use Qt it also helped us better understand some of the fundamental characteristics of C++ such as inheritance and OOP. Throughout this semester of using Qt for the first time we learned that:

* Qt is a language extension based on C ++

Qt itself has inherited the C++ fast, simple, object-oriented design and builds on that to provide many other advantages

* Qt has a very good portability

Qt not only can run in Linux, also can run on Windows, Mac OS X and many other different platforms. This means that applications written with Qt can run on multiple platforms with very little or no modification.

* Qt support cross-platform build (Cross-Platform Builds)

Writing software for multiple platforms can tedious and buggy at any time. This is especially when different compilers and platform combinations require several files to be compiled. With the built in Qmake tool, Qt is well positioned to meet this challenge. This tool can be used to generate the correct documentation for each different target platform.

* Qt to use simple and efficient

Qt's has a unified, cross-platform APIs which allows programmers to focus on the core implementation of a software project without having to worry about maintaining and managing multiple versions of the existing application's infrastructure. So Qt developers only need to learn a single API to write an application that can be run on several different systems.