Reflection

Team:

Working in a team has been a positive experience in this project. Everyone in the group has had the same ambition to make this project work, e.g. everyone showed up on time, both after school and on some weekends, and worked to maintain a productive and constructive environment.
Using scrum we have had a good overview over the daily work, and what everyone was working on. With almost daily meetings and most of the project done working on different things but sharing a room, has made it easy to integrate the different parts of the system. If someone needed another persons class to work in a certain way we discussed it in the morning and made sure all pieces fit together and worked as expected o avoid unnecessary extra work.
Programming in a group is different from individual programming in the way that its possible to let ideas be discussed and optimised before putting them into action. Its a way of leaving old programming habits behind and learn new solutions and methods from other group members. It has proven helpful many times to let someone else look at the code with fresh eyes - or simply explain sharing your own thoughts on an error to a group member, you would find the problem.
Git has proven to be an asset in the project by letting each member focus on their part in their own branch, and merge with the main branch when finished, thus avoiding getting time-consuming small conflicts with every push.

Documenting:

All documents has been written on shared documents through Google Drive. Many documents have been assigned to individuals and some has been written by the team as whole. To have the documents accessible on Internet has been a great help by letting everyone read and edit them in real time, without having to make a new version for minor changes. It has also been useful when writing meeting-agendas and documents, and not having to worry about forgetting to share it, or documents disappearing when changing computer. The only problem has been that in some study-rooms the Internet connection has been lost which caused trouble with and we've had trouble writing in the documents - a problem with marginal effects.

Scrum:

To plan and structure up this project Scrum has been used. It has proven to be a useful method - even if some refactoring had to be done.

Daily Scrum meetings has been a good way of knowing what everyone is doing and when they expect to be done with their tasks.
During the first sprint very optimistic time estimations were made, why many of the sprint tasks were still not finished by the end of the week, and had to be transferred into the new sprint log. Knowing approximately how many tasks would be finished every week was helpful when planning which features to implement, and removing the ones that were not likely to be done by the project deadline - thus avoiding starting to write features that would not be finished in time, and instead focusing on the highly prioritised ones.
For every end of each sprint we reviewed the sprint, giving and taking feedback within the team; which were a very effective way to improve both our performance on individual and on team level.

Coverage:

We used Emma for checking our testing coverage, this would prove useful for seeing how much code has been tested and which classes that still needed testing. However it would be useful to see how many lines of code the tests consisted of.
We also used Stan for checking circular dependencies.

The choice of design pattern:

At the start of the project we decided to go for a model-view-controller architecture because that was what we knew best. At the time we didn't know much about android-programming and because of that we had to improvise. when planning the application structure. The further we got into the project the more we realized that an architecture like model-view-controller wasn't really useful since we had to implement a lot of logic in the activity classes (which represent the view-classes) and the DataAccessor class. This rendered the controller classes almost useless and we realized that model-view-controller wasn't the best choice of design pattern for our application. There were two reasons why we didn't choose to change application structure, first of all we didn't know what other design patterns would fit into an android application and second of all we didn't have the time to do another big refactoring of the application.

Refactoring

The app is much relied on the database and its functionality. We started out with a popular approach in Android programming to have a DAO (Database Accessor Object) referenced in the controller, which is called to produce a reflection of its content as model objects. We came to realise this method was not for us. We were forced to edit everything through the controller class, and constantly getting values from the database to keep everything in sync. We thought this made the code difficult to follow and hard to test, so we decided to move the database access to the Account object as a singelton in the model package. This, however made the controller objects seem useless. It is also still some discussion if whether the other model object whould have direct access to the database. But we decided to keep them as they were since at some places we would like to store local copies of these objects.

More thoughts

The project seemed quite small when we first started out, and we thought that we would have plenty of time left for adding extra features in the end, but this turned out to not be the case. It took more time than expected, and we think it was much because of the learning time to trying to find out how to guide our way through the Android system and its limitations and features.