CS207 Final Workload and Lessons Learned Report

Team I (Monday 9 am -11 am)
Craig Griffin, Sean Waters, Stuart Lennon, Reece Jones

<u>Functionality Overview</u>

As a group, we managed to implement most of the functionality which was requested in the requirements gathering stage and detailed in the user stories. Listed below is the functionality we managed to implement at each sprint and subsequently the functionality that we did not:

1. During **sprint one** we successfully managed to generate a cryptogram for the user to play and allowed them to perform basic procedures to the puzzle such as inputting a letter and removing a letter. We did not manage to allow the player to select between letters and numbers as a preferred cryptogram type. An issue which caused us to lose many marks at this stage was our lack of Junit testing.

2. During **sprint two** we successfully completed all the user stories from the product backlog. Our software was able to save a game and load it back and track the stats of players of the game. To ensure we did not lose basic marks at this stage we ensured that we had thorough testing for each user story. An issue that came up at this stage was the robustness of our software. If the file containing a player saved game or the file containing the details of all players was tampered with, it would result in our software crashing.

3. During **sprint three** we successfully completed all the user stories. Our software could give the player a hint, the player could view letter frequencies and give up and be shown the correct solution. The player could also compete against their friends on a leader board. We tested most of the functionality at this stage, however, during the demonstration when showing how the hint feature worked a bug that we missed managed to crash the software. Our frequencies were also slightly off because we were taking spaces and punctuation into consideration.

Overall, our game completes the epic. A user can play a cryptogram puzzle. There is room for improvement in our solution however functionally it works.

<u>Workload Distribution</u>

The workload was distributed fairly amongst the team. Listed below are the tasks we identified at each stage and which member of the team completed it:

1. **<u>Iteration 0</u>** (User Stories and class diagrams): user stories were discussed as a team by all team members during a lecture slot and then written up by **Craig** and modified by **Stuart**. Class diagrams were produced by **Sean.**

2. **<u>Iteration 1</u>** (generate a cryptogram, enter and remove a letter):
    1. Create a basic class layout with getters and setters - **Reece**
    2. Implement an adaption of HashMap to allow for One to One mapping - **Sean**
    3. Load and select a random quote from a flat text file – **Craig**
    4. Encrypt quote – **Craig**
    5. Input and remove a letter from the puzzle – **Stuart**
    6. Display cryptogram and empty boxes for the user to enter letters into – **Sean**

3. **Iteration 2** (Load/Save game and track player stats)
    1. Create Model for Player and Players – **Reece**
    2. Save and load a game – **Craig**
    3. Testing of all new features – **Stuart**
    4. Adapt game controller to update stats of a current player - **Sean**

4. **Iteration 3** (Show solution, hints, frequencies and leader board)
    1. Give the user a hint – **Stuart**
    2. Display letter frequencies – **Sean**
    3. Leader board – **Reece**
    4. Show solution to current cryptogram - **Craig**

If doing a project like this again, it would have been hugely beneficial to utilize GitLab issue boards and branching features to more efficiently work on the same project as a team. There were no real changes regarding who was allocated to different tasks, if a person allocated to the task was struggling, they would always receive help from other team members.

Teamwork Strategy

For any successful group project, good teamwork and communication are key. As soon as the finalised group listings were released a group chat was created with all group members. We used Facebook Messenger for convenience but as a team, we agreed that if we were to do that project again, we would use Slack for its extremely handy ability to connect to git and show recent commits.

Decisions were made mostly during the lab slots where we conducted a scrum-like stand-up and a retrospective evaluation of how we got on during the previous week. We discussed what we had been working on through the previous week and decided on who would complete which bits of the next backlog.

A few disagreements came up during the implementation, these include the way in which we should implement a one to one map for mapping the alphabet to an encrypted quote and how we wanted to handle storing player stats. To get around these disagreements each member used git branches to create and implement a feature in their own preferred way. Next, we would meet up as a team and discuss the advantages and disadvantages of each method.

As the project progressed and the volume of code that we had written increased. We decided that it was essential to do some housekeeping and refactor the code as a team to make it maintainable and readable as possible. This involved separating functionality into different classes and making sure that each method had a Javadoc style comment detailing exactly what it did. This made developing code for the final stage much simpler.

Lessons Learned

Overall this project was hugely beneficial, and we learned several lessons completing it. One of the most valuable lessons we learned about was the power of version control like Git. GitLab allowed us to confidentially develop and improve our game with the reassurance that we could roll back and have a working copy at any point. However, as mentioned in the workload distribution section, I don’t feel like we got the full advantage out of using it. Gitlab provides issue boards and CI/CD features which could have been very useful in aiding our teamwork.

Another lesson we learned was the importance of good communication. In the early stages of the project when we were still in the so-called “Forming” stage of our team we were communicating but not really talking about what we were doing. This led to the same functionality being implemented by different people and creating all sorts of merging conflicts. As the project progressed and we became more comfortable as a team in the “Norming” stage communication become much better and as a result, the way in which we completed the remainder of the project became much more efficient.

Finally, completing the project in an agile style was a very good way of showing its advantages in a practical way. Meeting up with a product owner every fortnight and demonstrating what we had done and getting suggestions on how we could improve and adapt the software to better meet the requirements was a great way of implementing software and it was clear to see its advantages of a conventional waterfall approach.