Project Review Report

Team management

Our team structure evolved organically over the course of the project as needs arose and we grew into our roles. We ended up with our Team Leader acting as project manager and GitHub control. We added two code managers to handle both the architecture and backend, and the GUI. This allowed us to have 'experts' in all areas and easily allocate members between parts of the project. Our original plan of having a team leader and then a rotating scrum master was flawed. The rotation of the role led to a confusing and unclear team dynamic, some members filling the role more than others. It was decided that the single team leader works but the rotating control was unnecessary. The addition of 'expert' roles consolidated the chain of command in the team. This made it very clear to everyone who to talk to about what issues. These changes happened over the course of Assessment 2, when we realised our approach to risk management was flawed and we needed to be more flexible in our allocation of resources.

Our final structure was as such: Team leader, GUI expert, architecture expert, two general members. The experts and team leader had no extra workload over anyone else. However they agreed to focus their work on that area. Whilst the team leader and general members were able to work on almost every aspect of the project. The team leader again had no extra workload compared to the others but also put time into orchestrating the GitHub repositories and ensuring work was being completed and everyone was comfortable and able to do their best.

Overall, the risks associated with the project didn't evolve a significant amount. The main additions occurred within the start of Assessment 3 when it became clear that there would be significant risk associated when in came to taking ownership of another team's project. These changes were highlighted in our Assessment 3 change report found here. As mentioned above, in terms of management of the risks, the most significant changes occurred in Assessment 2 when we decided on having individual team member ownership for each risk for efficiency purposes.

Alongside this change in structure, it also helped the mentality with team communication. Using our Slack and channels such as: '*Assessment 3*' or '*website*' we kept our communication sorted and specific to SEPR. Failing that, we had a backup of Facebook Messenger for smaller personal messages such as where someone was or if they'd seen an important message. Communication naturally peaked at times of rapid development. Overall the team can be very proud of the communication had at all times, without actual regular conversations this non-hierarchical structure would likely not have been as effective.

Software engineering development methods

Methods Used

Throughout the whole software development process, we used the SCRUM methodology for the Agile model. It was chosen on the basis that we felt that this methodology would be the most appropriate and effective practice for our small team. This was mainly due to the way that the way that SCRUM made use of sprints in order to organise work. A sprint consists of setting tasks for each member of the team, carrying them out and then meeting with the team again to review the sprint and set a new one. Sprints usually last a very short time period, 6 days in our case. Due to the team all living within close proximity to each other it meant that meeting so regularly wasn't an issue. Due to each member having varying commitments in regards to both the course and extracurricular activities, the sprints provided us with great flexibility with time management. For instance the time of a SCRUM meeting can be changed quickly to suit the needs of the team members without incurring any substantial side effects.

Looking back on the whole project, we feel that using this methodology was a success given how effective it was. Whenever a critical decision had to be made or a team member encountered a problem, it meant that it could be brought up quickly at the next SCRUM meeting to be addressed. For example there was an instance where a member of the team didn't know how to make use of a method within the libGDX library. This issue was solved at the next SCRUM meeting with the help of another team member resulting in development to be allowed to continue.

SCRUM was also very effective in this project due to the frequent changing of requirements that we had been presented with. Due to the sprint mechanic of SCRUM, any change to the requirements could be discussed at the next meeting and incorporated into the software design.

Tools Used

A complete list of all tools used and short descriptions of why and how the usage evolved can be found here. But here is a summary of what we deemed to be the most important tools to the group.

Github

Perhaps the most important tool for the safe running of the project. This allowed us to easily maintain traceability and responsibility for the code whilst also acting as a centre for co-ordination. Along with integration to Travis CI for continuous automatic testing. Github allowed us to clearly keep up to date with where everyone was and how the project was progressing. It also provided an issue tracker which became extremely useful when the team were working out what problems there were left to tackle and who had been assigned to them.

Slack

We decided to use slack as a professional messaging client to keep it separate from personal conversations. This allowed the construction of 'channels' for parts of the project i.e. 'website' or 'UI'. Allowing the entire team to see information when they needed it, and keep a history of files and messages. It also ensured no one was being spammed with messages not appropriate to them. This kept communication efficient and transparent between the team.

IntelliJ

A fantastic Java IDE we used more and more throughout the project. Eventually allowing the export of comprehensive UML Class diagrams. We were able to easily share run configurations via the Github and IntelliJ was able to import and use them.