Top Right Corner **Project Review Report**

Our approach to team management

As we stated that we would in assessment 1 (https://sepr-topright.github.io/SEPR/documentation/assessment1/deliverables/Plan1.pdf) we followed a Scrum like approach to management. Scrum is an Agile methodology[1,2] meaning that the planning, development and testing phases are interleaved[1], broken into a series of fixed lengths cycles (1-4 weeks in length) called sprints[1,3]. At the start of each sprint there is a sprint planning session[3]. During which the team decides what work is to be completed. At the end of the sprint there is normally a sprint review where teams talk with the client about what has been done and what new tasks have been added to the backlog of work to be completed (the product backlog). Also there are usually sprint retrospectives where teams discuss what went well in the last sprint and what improvements could be made with regards to team working, productivity, communication and alignment with the Scrum process. This is done so that team can learn from their mistakes and increase their productivity whilst making the development process more enjoyable. As we stated in assessment 1 we didn't practice the the Scrum methodology exactly. We didn't have daily meetings as we didn't work on SEPR every day (due to some days being predominantly filled with lectures and other commitments that we have to keep). Instead we utilized Trello (see below) so that everyone can see what everyone else is doing and has done. Also, any potential problems were raised in the group chat and dealt with immediately. We also combined the sprint review, retrospective and planning sessions into a single meeting to take place during the SEPR practicals (as its the the only time that we can guarantee that we're all in the same place at the same time). We didn't place as much emphasis on the retrospective as we should have whilst we now function well together as a team, at the beginning we were poorly organised and at times our productivity stalled. We would have benefited from taking time out to talk about our "alignment with the Scrum process" as in some ways what we ended up doing was more similar to Kanban that Scrum. Kanban involves breaking down the project into a small number of discrete tasks and working on a set number of them until their done, rather than defining a set period of time in which to complete them[4]. In assessment 1 we stated that working in this way, without self imposed deadlines would probably hinder our productivity. This was definitely a correct assessment as in many cases things that were meant to be finished for say Friday would end up not being completed until the following Wednesday. Had we taken the time to reflect on this and been more strict about deadlines we could definitely have completed our work more quickly and avoided unnecessary stress close to deadlines.

We decided to stick with the Scrum approach we had used throughout the project as we were then able to decide on deadlines as a group rather than individually. This meant that we had to then rely on other people which meant we didn't want to hinder the entire group's project by falling behind on our individual parts of the project. Although we didn't always meet the deadlines we initially set, we were much better time managed than we would have been if we had set individual deadlines. Scrum was the best approach for our group as it allowed us to work very well as a team towards the end goal. It allowed each team member to work on their own part of the project, but allowed everyone to work towards the end product at the same time.

We opted for an Agile methodology as we are inexperienced software developers and therefore there is no way that we could completely design a system prior to commencing development (as a plan-driven methodology would have us do[1]). We would almost definitely have had to replan the entire process and redesign the entire system multiple times, which would take up too much of our time meaning that we spent less time actually developing the system (and due to other deadlines and obligations we can't justify the overhead involved in such a process). Many of our ideas (like the UI layout in the first assessment) didn't occur to us until halfway through development (sometimes after we'd got know the libraries etc that we were using a little better) and incremental planning and development allowed us to have these kind of ideas when they happened rather than "trying to force them all out" at once.

Whilst our general approach to management never really changed we did end up with a "team leader" of sorts that would delegate any remaining work after each team member had agreed to take on the various tasks that interested them. This ended up being a necessity as sometimes people just weren't decisive enough and otherwise the meeting would have ended up "stalling" and we either wouldn't have ended up achieving nothing more in that session or moving on, which meant that the work wasn't assigned to anyone and work that wasn't assigned to anyone didn't tend to get completed… Similarly it became clear that one team member wasn't keen on coding but has far better at UI and graphic design than most of us, another was far more musically talented and thus handled all sound effects and music choices, another was far more knowledgeable of the automated testing libraries that we used and thus handled most of the testing etc. So in a way our self-organisation actually led to us having fairly well defined "individual roles".

Software development methods and tools

As stated above and in assessment 1 (link above) we followed a Scrum like approach to software development (please do see above). We stuck with Scrum (or at least our version of Scrum) because we felt it worked, the incremental approach to development allowed us to only design the section of the system that we were currently working which was useful as over time we learned more about e.g. LibGDX (the library used to implement the GUI) and therefore our ideas of what the system should/could look like and how it should function changed. If we had planned it all ahead of time we would have ended up having to change all of the plans many times, which would have wasted a lot of time that we could have spent actually implementing things.

We stated that we were going to perform test-driven development and we did to some extent in the first assessment (see link) but ultimately ended up writing a lot of our tests after the code itself. This was partially due to our lack of knowledge of Junit (the testing library used see https://sepr-topright.github.io/SEPR/documentation/assessment2/Test2.pdf) and because in some cases we weren't disciplined enough and just wanted to rush ahead with the implementation. We would have benefitted from properly performing test-driven development as we could have found (and fixed) bugs more quickly and easily. Also in assessment 3 the automated tests we received were poorly written and much of the code wasn't tested so unit tests had to be written after the code itself. Similarly in assessment 4 we copied most of our tests from assessment 3 so they also weren't written before the code itself (see https://sepr-topright.github.io/SEPR/documentation/assessment4/Testingreport.pdf).

A Scrum board is a way to visualise the current state of the sprint backlog[5] (the work to be carried out during the current sprint). There are columns for: requirements to be implemented, tasks that have not yet been started, tasks that are in progress and tasks that are completed. We used Trello (https://trello.com/) to create and maintain a virtual Scrum board. A physical board would have been impractical as we would have nowhere to keep it and we all from work different places and thus wouldn't all be able to access or maintain it and therefore it would have been harder to track who was doing what. Unlike the online version we used and kept on Trello where anyone could access the board anywhere. As well as make use of their added features such a due dates for the completion of work, and the ability to create checklists and see an overall completion of the current scrum.

We used the JUnit automated testing framework (http://junit.org/junit4/) which is supported by Eclipse[6] (the IDE we used) and JMockit (http://jmockit.org/) to mock out dependencies. We never stopped using JUnit as we didn't find any other libraries that were better and didn't want to have to spend time learning to use any others with no benefit.

We used Git to manage our version control (with our repository hosted on GitHub), Google Drive and Google docs to share and work on our documentation and a Facebook group chat to handle communications outside of meetings. We stuck with git throughout as most other teams were using it, making it easier for us to pick up their projects and for them to pick up ours. Google drive and docs made it very, very easy for us to share documents and even work on them simultaneously without having to continually send each other updated versions (which would have made it difficult to merge changes etc).

We also used Gradle (https://gradle.org/) to handle build automation and testing (making it easier for others to build and test our code). We started to use Gradle shortly after the start of assessment 2 once we learned that it would make it far easier for others to fetch all of our dependencies and build, test and run our code using a single command such as "gradle run" that would fetch all dependencies, build the code and then run it.

We used Travis CI (https://travis-ci.org) to run tests against our code base after every new commit to the project. This helps to reduce the number of bugs that make it through our software. This allows us to easily see any errors in the commit and stops any extra errors being made before the code can be changed back to a working form. We started using Travis (at the start of assessment 4) as overtime it became apparent that certain members weren't always rerunning the tests when they made changes, having the tests run every time we made a commit ensure that this would no longer occur.

Bibliography

[1] I. Sommerville, *Software engineering*, 10th ed. Boston: Pearson, 2016, pp. 72-88, 652.

[2] M. James, "An Empirical Framework For Learning (Not a Methodology)", *Scrummethodology.com*, 2016. [Online]. Available: http://scrummethodology.com/. [Accessed: 31- Oct- 2016].

[3] "Scrum Guide", *Scrumguides.org*, 2016. [Online]. Available: http://www.scrumguides.org/scrum-guide.html. [Accessed: 31- Oct- 2016].

[4] J. Bowes, "Kanban vs Scrum vs XP – an Agile comparison - Manifesto", *Manifesto*, 2015. [Online]. Available: https://manifesto.co.uk/kanban-vs-scrum-vs-xp-an-agile-comparison/. [Accessed: 31- Oct- 2016].

[5] J. Bowes, "Agile concepts: the Scrum Task Board - Manifesto", *Manifesto*, 2014. [Online]. Available: https://manifesto.co.uk/agile-concepts-scrum-task-board/. [Accessed: 31- Oct- 2016].

[6] "Help - Eclipse Platform", *Help.eclipse.org*. [Online]. Available: http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2FgettingStarted%2Fqs-junit.htm. [Accessed: 31- Oct- 2016].