Team Management and Structure

We have followed an agile SCRUM framework throughout the project which has obviously very largely influenced the development of our team structure. Our initial approach to team structure was somewhat informal due to our lack of familiarity with one another, and instruction from Sommerville [1], “Agile development teams are always informal groups”. In order to remain *cross-functional* as “the best agile teams should be” [2], team members were each assigned a range of roles based on each person’s self-assessed strengths, as well as personal enjoyment/interest [roles found here]. No implementation was required in Assessment 1, but this helped highlight who amongst us was more suited to documentation and management roles.

During assessment 2, some roles were swapped (as we predicted would be the case) and people began to more confidently find their natural position in the group. This, however, was not without some admitted complication. The team suffered from a problem of individuals being consistently late to meetings and deadlines. This impacted the workings of the group, as the remaining members did not feel respected and could not complete work which relied on the uncompleted work of others. In order to combat this, further risk mitigation was carried out [see here], which partly resulted in some reallocation of responsibility. We referred back to the teamwork methodology aims that we had decided upon at the beginning of the project [see here], namely, “clear view of how an individual’s work and actions impact the progress of the whole team”.

By the time assessment 3 began, we were more aware of one another’s workflow strategies and had a working structure. The group was informally divided into three sub-groups with differing focusses, but still an enforced overlap of aim and communication. These groups were: documentation including testing and risk (Chloe and Tamour); lead software engineering to oversee development (Jordan and Samuel); further software engineering who operate at the instruction of the lead team (Goli and Jack). We decided to stick with this structure into assessment 4.

Midway through assessment 4, our project was interrupted by the effects of covid-19. This meant that each member of our team was facing unseen problems in their personal lives which obviously had an impact on our workflow - our risk mitigation efforts could only do so much. We had experience working distanced before, over the Christmas break, but did not ever plan to need to function this way for such an extreme length of time, on top of everyone’s personal struggles. We were able to use this experience from the Christmas break (one of the weaker points of our teamwork as before mentioned) and the fact that an agile methodology is adaptable to distanced working, to form a base for our approach. To support this, we studied [3] “Distributed Scrum: Agile Project Management with Outsourced Development Teams”, particularly the “distributed scrum of scrum” model as this reduces dependencies between teams and fit well with our preexisting pairs structure. However this served mainly as a source to center our approach as in practicality the size and lack of time zone variation made our collaboration easier than that of larger global companies.

Software Engineering Development Methods and Tools

In response to the constraint that the project must be completed using Java, we chose to work using LibGDX [reasoning discussed in assessment 1] and decided to only choose projects from other teams which also used this framework. Luckily, this was applicable to the vast majority of teams in our cohort. In order to limit the broadness of the software which we needed to learn for development, we avoided projects which relied heavily on Box2d. We believed that this way would allow us to create a higher quality product as we could really focus on meeting our requirements in simple but effective ways.

The majority of the tools which we used did not change over the course of the project, so the justifications are the same as found in previous assessment deliverables [here]. This is because any issues which the team came across were largely not technical but structural as described above. Our use of Jira really helped us to combat these organisational issues.

In terms of development structure for assessment 4, we had decided to make use of additional techniques and features GitHub offered in order to help form a more efficient development structure. Using GitHub Actions, we had set up a system that does not allow commits to be accepted if their tests do not successfully build (i.e such that every time a commit is pushed, GitHub will compile and run the LibGDX tests for it). The pros to this were that it made it easier to ensure written code works at least from a unit testing perspective. Since everyone worked on their individual branches, they wrote the tests for their own units and therefore ensured their units worked alongside all others. The downside is that sometimes it became a habit to no longer build your tests before pushing code, which would have made for finding errors in code before assuming they were fine and pushing them to the repository.

An additional step we took was to enforce acceptance rules for certain branches. Our planned branch structure was to merge finished branches into a “dev” branch, which would require at least one other team member to review the commit before it’s accepted. Once changes were successfully merged in dev (which would require manual changes), they would be merged into “master” (which some would consider the ‘production/prod’) branch which required two reviews. Any commit in the master branch would be considered a release version of the game. While this seemed like a good idea at first, when it came to getting closer to the deadline and more commits were required it became an issue for team members to rely on another team member to be available to accept any commits into dev/master - this availability also links to the distributed team issues which are discussed in the section above. Overall, this rudimentary CI/CD pipeline was a good first attempt at what most professional teams work with, although it missed some features such as planned code reviews, and staggered release dates (i.e making sure there is a big release/dev commit every week).

Because of this, we consider our software development practises to be bordering on Level 4 of the [4] Capability Maturity Model for a team of our size and formality. We have “ tested the suitability of the process in multiple environments” given the changes caused my covid-19 and have therefore challenged our capabilities.

Bibliography

[1] I. Sommerville, Software engineering. Boston [i pozostałe]: Pearson, 2018.

[2] J. Sutherland, Scrum the art of doing twice the work in half the time, 1st ed. New York: Crown Business, 2014.

[3] S. J and V. A, "Distributed Scrum: Agile Project Management with Outsourced Development Teams," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, Waikoloa, HI, USA, 2007.

[4] IT Governance, "The Capability Maturity Model (CMM)," [Online]. Available: https://www.itgovernance.co.uk/capability-maturity-model. [Accessed 04 2020].