Project Review Report

Stakeholders: Richard Paige, University of York Communications Office
Team: Barney Morgan, Cameron Smith, Harry Berge, Jake Phillips, Matthew Wilkie, Rob Weddell

Team Management Review

It is clear that the groups approach to team management has evolved in parallel with the project's completion. To begin with, we were a group of six strangers with little prior experience at project management. But now, with the finished product behind us, each individual member of the team has developed and grown alongside the assignment and can confidently complete team projects to the highest possible standard. To begin with, we decided it was in the best interests of the project to follow the typical and popular industry standard of the 'Scrum' method of agile development. This choice was well documented within our original planning document [1] and much focus was placed on the flexibility of the approach. From the start, the team decided that the strict meetings required of a traditional SCRUM approach would not be viable for our individual schedules. We reduced the recommended number of meetings down from four to two per week and neglected to assign roles to individual team members. This was because we believed that flexibility was the most important aspect of successful team management.

In retrospect, we were very wrong. The bi-weekly meetings were never strictly held and by having no strict team roles, the members did not have much focus on the task. In reality, at the beginning of the project, each team member was slightly lost and essentially ignored the plan we had laid out in order to pursue their individually chosen tasks. Consequently, we firmly believe that the initial deliverable was lacking in quality due to the fact there was too much ambiguity in task delegation which lead to lots of time being wasted. From this, we learnt that the team needed clear, precise guidance in order to work effectively.

Following this, the team decided to make large alterations to the plan. Firstly, we assigned a team leader whom we felt comfortable delegating implementation choices to. This helped us achieve a focused and unified approach to the project's completion. In addition to this, stricter roles were assigned to each member to compliment their individual strengths. For example, the stronger programmers were tasked to focus entirely on the coding of the project. By doing this, we could delegate tasks effectively and also made it easier for members to take on extra tasks whenever it was appropriate. These roles were derived from our newfound understanding of software engineering roles from the lectures on team-working [2].

Despite the benefits of this new approach, it became clear that one team member was not contributing equally as a result of their other commitments to the university. This meant that they could not get their delegated task finished on time. As a result of this, the completion of the map was delayed which consequently caused a bottleneck in development as it held up the critical path in our gantt charts [3]. Despite this, by following the risk mitigation techniques outlined at the start, the problem was quickly resolved. However, it was agreed by every team member that this delay had a noticeable impact on the quality of the project (a simplistic, temporary map had to be made for testing) and that the only fair way of dealing with this was through reducing their mark in the team evaluation. Whilst it could be argued that this approach was rather harsh, it resulted in a huge boost in their workload, and has given said team member a stern push to create some very high quality work which has made an overwhelmingly positive impact on the project.

Lastly, the team made a large mistake in the project selection phase for assessment 3. We rushed into a decision of choosing a game we enjoyed the concept of the most, rather than considering the documentation, quality of code, and software engineering practices followed. This lead to an excess of hours being spent overhauling the game from the ground up, and, as a result, stagnated other areas of work whilst waiting for the implementation to be completed. Despite still being successful in the end, the team learnt from their mistakes and upon choosing a project to take over in assessment 4, we completed

adequate research into project selection to ensure a smooth transition. On reflection, this was a positive learning experience as it provided evidence that: although a project may look successful on the surface, if proper project management techniques are not followed internally it can make a project very hard for others to work with. We believe this is a very important lesson to have learnt going into the industry and it succinctly summarises everything we have learnt throughout the module.

Software Engineering Development Method Review

The software engineering development method we selected was the 'Scrum' method of agile development, and we used and evaluated the following tools for team management and development: Google Drive, GitHub, LibGDX, LucidChart, ProjectLibre, StarUML, Facebook messenger / Google Hangouts, and Tiled. The tools and methods selected have remained consistent throughout the project, with only a few additions to be made throughout the project such as TravisCI, a continuous integration tool, mentioned in our previous assessment documents.

As mentioned above, we initially planned to have two meetings per week in accordance to our slightly altered Scrum model: once to plan the week's sprint (workload) and once to reflect and review the work completed during that sprint. Originally, we believed this would yield great results and ensure that work was completed punctually. And to some extent, it did help keep the group on track. However, as initial motivation dwindled and external responsibilities arose, these meetings became less and less frequent.

Over time, the group tended to sway more towards online communication through Facebook messenger, Google Drive, and Github rather than holding physical meetings due to the lack of availability. Because of this, individual deadlines became less concrete, and team members began to complete their designated tasks in their own timeframes. It could be argued that despite the numerous positives, when viewed in this light, online communication actually hindered the software engineering methods of our team. By removing the physical aspect, some team members got lazy and did not complete tasks on time as they faced no direct consequences for their delays. It also meant that future task delegation was not discussed in full detail, but rather summarised in a short paragraph online with members not being 100% sure what their tasks actually entailed. This risk was raised in the risk documentation, and the team attempted to rectify their mistakes by enforcing a singular meeting every week and keeping track of attendance.

The largest obstacle our team faced was ensuring the chosen methods and tools were transferable during the transition to a brand new project in assessment three and four. The risks, considerations and struggles of this are well documented in the change management [4] documentation and follow the IEEE standard [5]. To summarise, we decided it was very important to minimise the amount of time needed to learn new software as our time could be much more effectively spent applying our current knowledge. However, upon reflection, the team placed too much focus on ensuring the tools were transferable and not enough focus on the quality of the other team's code. This mistake was rectified upon the transition to assessment four as both factors were taken into account.

As the project progressed, it was decided that the team required additional tools to work effectively, for example, we decided that adding TravisCI [6] to our list of tools would have a positive overall impact on the project, and save time overall. In general, upon deciding to add a new tool to the project, there was a consideration as to whether the time spent learning the new tool was worth the improvement in overall project quality. For example, learning how to model realistic 3D ships may have added another layer of quality to the game, but the time spent learning and implementing them would take much too long for it to be deemed worthy. Thus, there was a distinct tradeoff between optimism and realism: some team members spent lots of time investing themselves into learning complex software when the current software already did an adequate job. However, this dilemma of being overly ambitious was adequately documented in the original risk documents and thus generally avoided.

References

[1] Element of SEPRise!, Updated Method Selection and Planning [Online]. Available: https://sepr4.github.io/web/submission/assessment2/updated/Plan2.pdf [Accessed 1 Jan. 2019]
[2] SEPR Module 2019, Teamworking Lecture [Online]. Available: https://vle.york.ac.uk/bbcswebdav/pid-2846239-dt-content-rid-7570814_2/xid-7570814_2 [Accessed 15 Apr. 2019]
[3] Element of SEPRise!, Gantt Charts (Assessment 2-4) [Online]. Available: https://sepr4.github.io/web/submission/assessment4/updated/Gantt.zip [Accessed 15 Nov. 2018]
[4] Element of SEPRise!, Assessment 3 Change Management [Online]. Available: https://sepr4.github.io/web/submission/assessment3/Change3.pdf [Accessed 15 Nov. 2018]
[5] Summary of IEEE Software Testing Standard [Online] Available: http://www.cs.otago.ac.nz/cosc345/lecs/lec22/testplan.htm [Accessed 30 Nov. 2018]
[6] TravisCI Website [Online]. Available: https://travis-ci.org/ [Accessed 3 Nov. 2018]