# 4 | Project Review Report

## 4.1 | Team Management & Structure

Reflecting on our software engineering project, it was clear we maintained a democratic leadership approach to manage our team [1], whilst structuring our team into groups pairing certain strengths together. The team structure consisted of 3 subgroups, each containing 2 people. We maintained a non-hierarchical management approach, as the subgroups allowed the members of the team to apply their strengths to certain areas of the project, to ensure the best possible output. The development method of choice evolved and culminated in the waterfall model, lending itself to our team structure as we were able to tackle subtasks within our groups, then meet to discuss how we would proceed with the next waterfall phase.

Initially, a flat, non-hierarchical management approach was chosen as the management structure for the project; mainly due to the timeframe of the project and partially due to the parity that we perceived it would have with the initial software development method (RAD), where each member could quickly contribute to the project at its beginning, without relying on a single 'manager' to coordinate the process. This team management approach did not evolve throughout the project as we felt it was working to deliver the requirements, despite changes to the risks, requirements, and our knowledge of the software engineering method. As we became more familiar with the strengths and weaknesses of team members, we became able to allocate the right tasks to each of the subgroups, then manage these individually within our smaller teams. We felt this would have a positive impact on our teams behaviour allowing us to share and acquire knowledge more efficiently [2].

In Assessment 2 this approach worked well, as we paired team members who were more confident using Unity and C# with those that had less confidence in these areas. This allowed us to tackle the challenge more effectively as it meant those who were less confident could work more efficiently by asking a subgroup team member for assistance. This maintained the democratic management approach whilst also following our structure using the subgroups to tackle individual tasks and requirements presented in this part of the assessment. We also took the same approach in Assessment 3 using pairs.

The risks and client's requirements of the project changed in Assessment 4 but we did not change our team structure, as the requirement changes concerned only minor extensions to software functionality, and we felt it needless to restructure the team because of this. We again had no designated leader, and continued to use the waterfall method of software development that we had adopted by this stage, using subgroups to progressively develop our final product. As our understanding of the software engineering process evolved we concluded that the way our team was managed could have been improved by having a manager of the subgroups to ensure deadlines were met and ensure that they did not overrun, as overrunning would slow down development when using a waterfall style, with the need for the present phase to be finished before progressing. On the whole however, we feel that the teams structure using subgroups with a democratic management strategy, was an effective strategy to achieve our final result.

## 4.2 | Development Methods & Tools

### Development Methods

Prior to beginning the project, we anticipated that *Rapid Application Development* (RAD) would be the most appropriate development method given the nature of the project (reasons enumerated in assessment 1 deliverables) and the nature of the team (newly formed, a novel development task, etc.).

As the development process began, RAD proved to be a suitable development method by enabling us to quickly prototype a basis for the game [3]. In principal, when there is no pre-existing software structure, a method with little planning can be best to get the project up and running. However, when extending code, new code integrate seamlessly with old code, and more care needs to be taken to not break pre-existing functionality, as is the risk when continuing the development of another project. Hence, we saw the need for a new development method (rather than RAD) with more emphasis on careful integration and less emphasis on adaptability and prototyping [4]; one that may perform better within an existing structure (relevant as we were taking on development of other teams' projects). The team henceforth adopted a waterfall approach, which we saw as suitable candidate to fulfill the changing development needs.

The waterfall method is commonly followed in the presence of a managerial authority. Maintaining a flat management approach meant this was not the case but with only 6 team members to coordinate between, we found this not to be an issue.

### Tools

Our use of tools over the duration of the project remained mostly constant. Quite simply, the tools provided adequate functionality despite the changing needs and risks over the period.

*Facebook Messenger* was sufficient for organising meetings. This, in combination with *Google Drive* acted as a file and team management framework that was wholly adequate given the unchanging size of our team. Supposing that development in stages 2-4 of the project involved between-group-collaboration (rather than project takeover) then we would have certainly opted for a more powerful tool for communication such as *Trello*.

The teams whose projects we opted to takeover development of were both produced in *Unity* so we were able to exclusively use *Unity* for all game development during the course of the project. As a consequence, *Visual Studio* (the chosen IDE) was used for game coding throughout the project as the projects' code was solely C#.

Before development began, *Visual Studio Team Services* (VSTS) was chosen as the means of version control, given its native support within *Visual Studio*, also the IDE of choice. In practice, however, team members opted to use *Github* instead, due to existing familiarity with the service and repository ease-of-access. In addition, it proved troublesome to synchronise Unity's engine files through VSTS, whereas this was easily done through *Git*.

## References

[1] - https://aornjournal.onlinelibrary.wiley.com/doi/pdf/10.1016/S0001-2092(07)66907-5
[2]- http://facultyresearch.london.edu/docs/SIM02updated.pdf
[3] - https://www.bbc.com/education/guides/zp3kd2p/revision/9
[4] - https://scholars.unh.edu/cgi/viewcontent.cgi?article=1288&context=honors