Method Selection and Planning

TEAM 14 [ENG1]

1 Software Engineering Methods - Part A

1.1 Methodology

We decided to follow an Agile methodology, as we believed it would allow us to pick up any bugs or errors quickly in the process as well as incrementally improve and add to the game code. An Agile methodology would also guarantee that there would be predictable and guaranteed delivery of features every week which we decided was important due to the limited time available for the project. Following this, we decided to use the SCRUM framework which follows the core principle of continuous improvement whilst also being a good framework for dealing with any changes in the user requirements. We also believed that having a Product Backlog (a ‘to-do’ list for SCRUM) and a Sprint Backlog (the goals of the current sprint) would help keep us on track during the sprints.

Alternatively, we had considered a Waterfall methodology, however the rigid processes and the difficulty in adjusting the product to customer feedback after completion meant that we didn’t believe it would be suitable for our project, especially considering client feedback and communication was important to us. Also, if any bugs or errors were encountered towards the end of the process it would mean having to restart a lot of the other processes to fix them, which wouldn’t have been suitable for our deadline.

1.2 Collaboration/Development Tools

In order to facilitate team cooperation and planning we utilised both Zoom, Discord, Github and Google Drive.

Following the SCRUM Agile Methodology, we had weekly meetings using Zoom calls in order to discuss ideas, allocate new tasks and update the entire group on the progress of previous tasks that were allocated. Alternatively, we had considered using Google Teams but agreed that, as Zoom was more commonly used, it would be a better fit as there would be no adjustment period to using a new platform.

During the rest of the week, we agreed to use Discord to communicate as we could split the server into multiple channels, such as: Java-Progress, Design-Progress and Tasks. We felt that this way we could better separate out what needs to be done and, when you look in a specific channel, all the information would be relevant. We had considered using Slack, but, once again, found that Discord was more commonly used and the group were more likely to utilise a platform they had used regularly than one they hadn’t.

A Google Drive was shared between all members of the team and all work, such as designs and deliverables, were to be stored there for easy collaboration. This way the Software-Development team could access the designs the Design team created without needing to constantly ask. It also reduces the risk that work could be lost as there will always be a copy in the Google Drive.

Github was used to manage the code for the project, it allows us to work remotely and merge any new snippets of code into the project. This way we can all work on different parts of the game at the same time.

2 Team Organisation - Part B

2.1 Team Roles

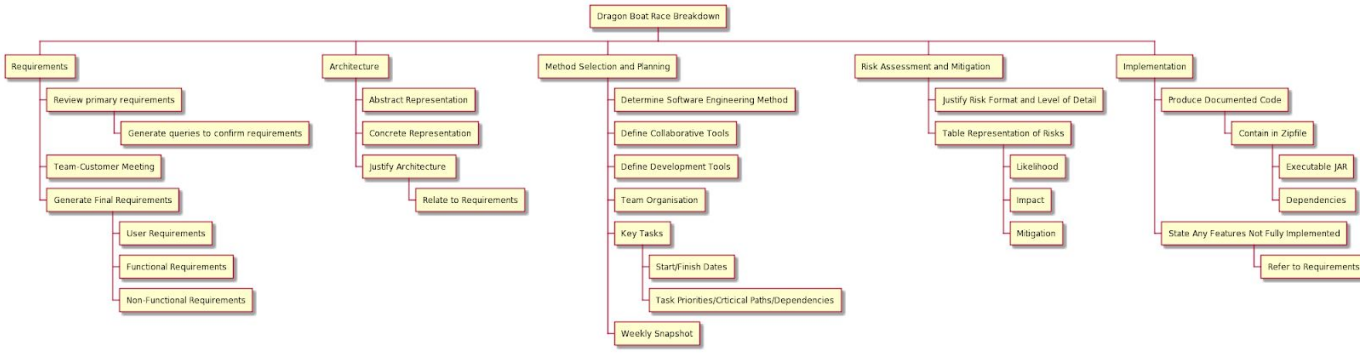
During the first meeting of the group, we outlined a Team structure according to each individual's strengths and what they wish to accomplish during the project. We split the group into two divisions: a Software team and a Design team. The Software team was responsible for the implementation of the project, including the architecture, whilst the Design team would handle any artwork, requirements and other documentation work. This structure meant that there would always be one other person capable of working on someone else's task if the initial person was unavailable, as long as they were in the same

team. For example, at least two people were capable of understanding sections of the game code, which helps provide a high-bus factor. Within the team, any tasks that need to be completed are broken down into smaller chunks and then distributed equally, which keeps the workload fair and manageable.

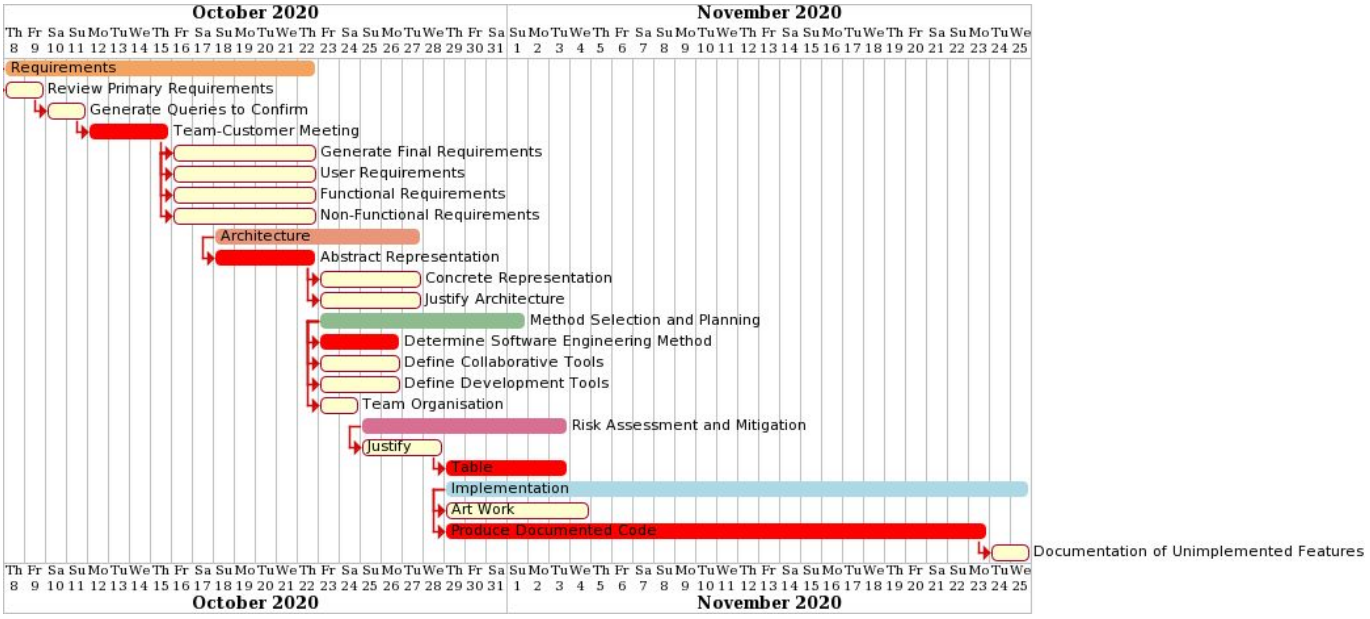
3 Project Plan - Part C

3.1 Key Tasks

We broke down each section of the project into key tasks using a Work Breakdown Diagram with PlantUML:



In the first Sprint we created an initial Gantt Chart to lay out what needed to be done within a certain time frame and help us better see a critical path and any task dependencies. We felt that a Gantt chart would help support our Scrum methodology by giving us a clear view of what was required for the entire project as well as helping us to break down the tasks for our sprints.



3.2 Task Priorities and Dependencies

Priority tasks are coloured in red in the Gantt chart. We considered each stage of the project to determine which tasks needed to be completed first during a sprint and decided upon the following tasks:

- Team-Customer Meeting - as we would need to know any further information or requirements before we could start proper work on the Architecture as the requirements would define what the software needs to be accomplish
- Abstract Representation - as the Development team need a groundwork of what needs to be included in the code and the Design team will also need to consider any artwork or documentation
- Determine Software Engineering Method - as a methodology will define how we lay out our project and our time, in our case we chose the Scrum methodology
- Table - we believed a risk assessment table was crucial so we can potentially act upon any risks before we got too far into the project or implementation stage

- Produce Documented Code - as this was what the client requires and needs to be completed before the deadline, it was set as a high priority

Using this information we also highlighted the task dependencies:

- ‘Generate Final Requirements’ is dependent on the ‘Team-Customer Meeting’ finishing, so we can get any extra requirements that weren’t initially mentioned
- ‘Abstract Representation’ is dependent on finishing the ‘Generate Final Requirements’ task as the Development team need to know exactly what is required by the code to create their blueprint
- ‘Concrete Representation’ and ‘Justify Architecture’ both rely on finishing the ‘Abstract Representation’ as they are extensions of the ‘Abstract Representation’
- ‘Art Work’ depends on finishing the ‘Generate Final Requirements’ task as any additional drawings or the clients design choices need to be considered
- ‘Produce Documented Code’ relies on finishing ‘Generate Final Requirements’, ‘Concrete Representation’ and ‘Art Work’ tasks

3.3 Critical Path

A critical path is a sequence of activities that, if they were delayed, then the completion of the entire project would also be delayed. Considering we have a total of 48 days to complete this project, we identified the critical path as:

1. Review Primary Requirements
2. Generate Queries
3. Team-Customer Meeting
4. Abstract Representation
5. Art Work
6. Produce Documented Code
7. Documentation of Unimplemented Features

The ‘Produce Documented Code’ task is the longest task we have to complete, however, it relies heavily on the tasks beforehand (listed above). Therefore, if any of the previous tasks are delayed then so would the ‘Produce Documented Code’ task and this could delay the entire project.