Team 19 - Method selection and planning Deliverable

Method selection and planning

(1) An outline and justification of the team's software engineering methods

There exist many software engineering methods we could choose from. These are split into three types: (1) plan-driven methods, that have an overall project "roadmap" to guide the project through its lifecycle; (2) agile methods, that focus on dealing with volatility & customer-driven development (3) soft methods, that are employed when the problem specification is not known in detail (usually before SWE is started, to help determine requirements).

Our principal software engineering development methodology is Agile. We chose this because:

- It copes well with volatile requirements, which we expect this project to have.
- It works well in small teams.
- It employs short iteration cycles, with low documentation overheads, focusing on maintaining an inclusive relationship with the client.
    - This allows us to incrementally deliver working prototypes and get immediate feedback for the next iteration.
- We also felt the "standup" concept was incredibly useful, and even adopted it into our overall planning and organisation model.

This contrasts to plan-driven methods which have a large documentation overhead, and as a result are very inflexible.

We note that Agile dictates that face-to-face communication is better than online, but this is unfortunately not possible with the current pandemic.

(1) Developmental / Collaborative tools we have used, and (2) our approach to Organisation

Our codebase is stored in a repository on GitHub, which we felt was the clear leader for version control and enabling team-working on a shared programming project. Additionally, it has inbuilt integration with VS Code -- a working environment many of us use regularly -- and the platform in general has by far the most tutorials and useful resources available.

- Alternatives include Bitbucket by Atlassian or other repository hosting tools / version control tools, but we felt GitHub does both of these things best.

Primarily we collaborate on Discord, using the platform to communicate (both in calls and through text), share files, and self-structure both the assignment and current progress of deliverables.

We believe Discord is fit for this purpose for the following reason:
- We all had existing Discord accounts we could use, and were familiar with the platform and how to use it.
- Discord permits creating a custom server and modifying its structure to best suit the application, with numerous pre-built templates to speed up this process.
    - This allows the server to be flexible and accommodate multiple facets of the engineering process. The centralised approach it facilitates increases working efficiency and coordination, and reduces the complexity of working across multiple applications.
    - For example, we have text and voice channels for each deliverable, allowing uninterrupted communication between assigned team-members.

Possible alternatives we considered included Slack, Facebook Messenger, and Zoom.
- Slack is the closest proxy to Discord, however we felt that our lack of experience with it (and few of us having pre-existing accounts) was enough reason to choose Discord as they are very similar anyway. Furthermore, to unlock all of the useful features Slack offers requires a monetary subscription.
- Facebook Messenger has both voice and video call options, but poor support for file sharing (including previewing embedded images etc.) and no ability to structure a groupchat in any way. Zoom had similar drawbacks.

A planning tool we use is Trello. This introduces the concepts of Kanban-style “Boards”, which can be created for projects and used to keep track of tasks.
- It allows us to keep track of task priority, status, and assigned members, along with a short description and a task deadline.
- The board is split up into “To-do”, “Doing”, and “Done”, which allows us to asynchronously self-organise and determine which [sub-]tasks need to be completed.

We chose Trello since a few of our members had experience with it already, and it is generally well-regarded as a project planning tool due to its intuitive UX.

A possible alternative to Trello would have been Jira by Atlassian, but the free plan of 2GB of storage we felt was insufficient as many attachments will be required when collaborating on, and updating the team about, documentation.

A Systematic Plan for the Project

We highlight that since initial plans will most likely be wrong, our project planning must be iterative and change as our understanding of the requirements evolve (and the technical work progresses). Thus we aim for our plans to drive activity, and for the activity to subsequently inform planning. We will start by identifying project constraints.

Project Constraints

Deadline: Our deadline for this project to be completed is the 25th November 2020. We must have completed all written deliverables, as well as the codebase and finished product according to the specifications described by the customer, and elicited in our requirements analysis / throughout the communication we have held with the customer.

Resources: We are a team of 6, and thus have six people to spread across the various tasks. We aim to reduce our so-called “Bus factor” by ensuring multiple people are capable of working on each deliverable, and are kept up to speed with the current progress through constant communication.
Budget: Our budget is so minimal as to be non-existent, hence we plan to use free and open-source libraries to drive our development. We have identified LibGDX as a suitable game-development library to use in this project.

Project Organisation

We have the following deliverables. Assigned to each are various members of the team. Larger deliverables are assigned more members as needed.

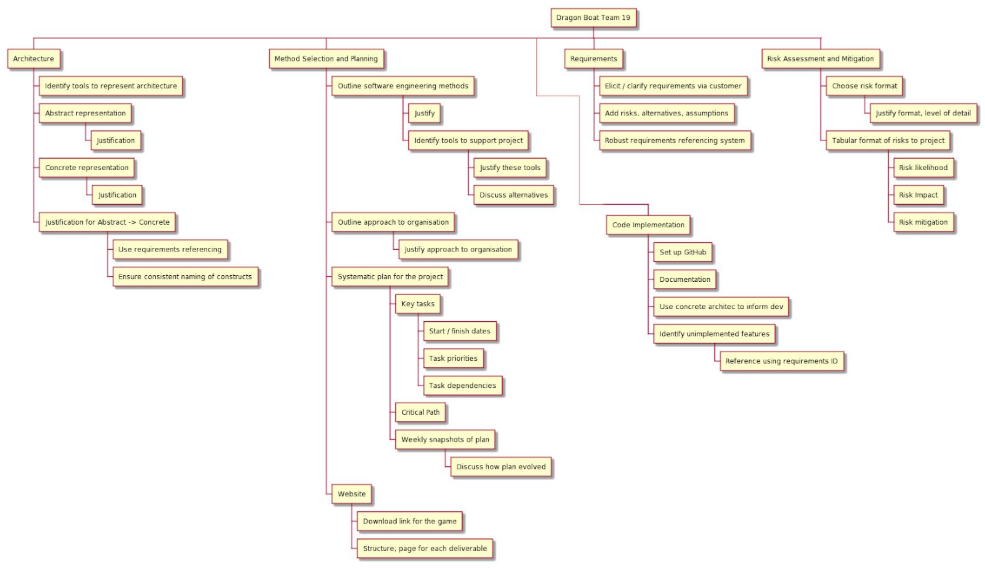
- Architecture -- Justin, Jack (peer reviewed / supported by Andorea)
- Method Selection and Planning -- Dougie, (peer reviewed / supported by Julia)
- Requirements -- Julia, Andorea (peer reviewed / supported by Justin)
- Risk Assessment and Mitigation -- Luke, (peer reviewed / supported by Dougie)
- DragonBoat Race game, programmed in Java -- all members of the team contributing at various points.

Jack was assigned secretary and takes down minutes / a summary of all meetings and discussions held, to ensure that key resolutions reached are recorded persistently.
Luke leads the team on the code base, and acts as Scrum leader. Pull requests must be reviewed by at least one other member of the team (this is enforced by GitHub).
Dougie leads the organisation of the project in general, as he was assigned to the Planning deliverable and as a result was required to identify key tasks & their priorities, thus was well-placed to assign deliverables to available team members.

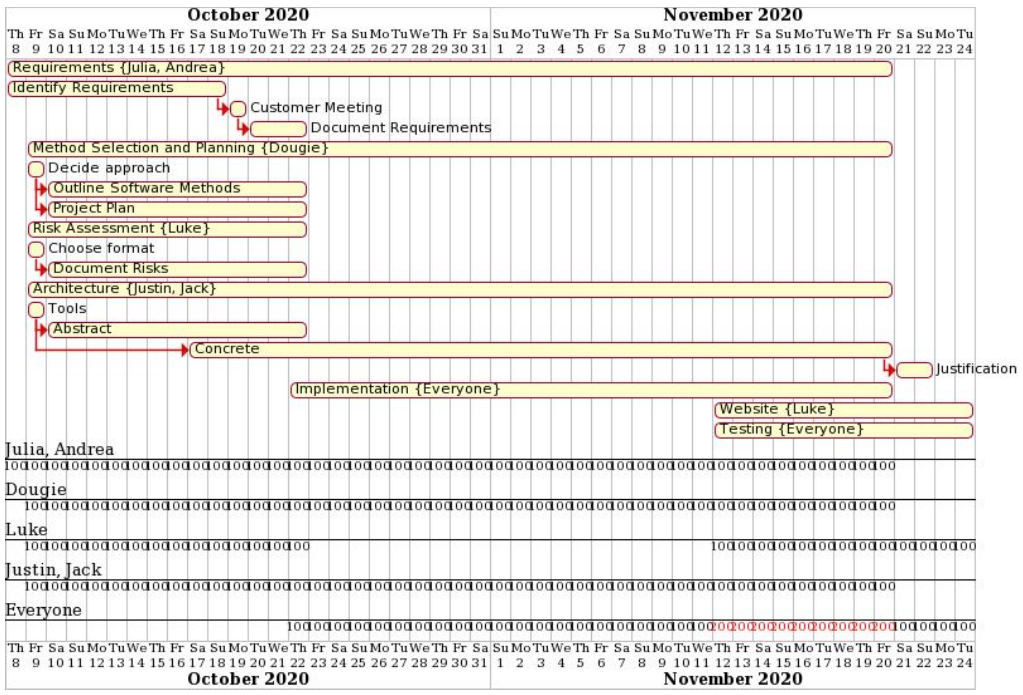
Stakeholders include the University of York communications office (who will use our game for promotional activities), and Dr. Javier Cámara, who represents the customers and final users of the game.
Our agreed method of communication with the customer is voice call, arranged via email 24 in advance. We had weekly slots where we could schedule a meeting, and took advantage of this, posing any questions / clarifications we desired about the requirements (and product in general) to the stakeholders.
Hardware and software resources: the customer has specified that the program should be written in Java to facilitate cross-platform play, but with a primary target platform of standard Desktop PCs with a display ranging between 15-27 inches, and standard peripherals (keyboard and mouse).

Project Schedule

We will break down the work into a number of work packages, which themselves are broken down into key tasks and sub-tasks. This UML WBS diagram is ordered sequentially by task priority.


Dragon Boat Team 19
Architecture
Identify tools to represent architecture
Abstract representation
Justification
Concrete representation
Justification
Justification for Abstract -> Concrete
Use requirements referencing
Ensure consistent naming of constructs
Method Selection and Planning
Outline software engineering methods
Justify
Identify tools to support project
Justify these tools
Discuss alternatives
Outline approach to organisation
Justify approach to organisation
Systematic plan for the project
Key tasks
Start / finish dates
Task priorities
Task dependencies
Critical Path
Weekly snapshots of plan
Discuss how plan evolved
Website
Download link for the game
Structure; page for each deliverable
Requirements
Elicit / clarify requirements via customer
Add risks, alternatives, assumptions
Robust requirements referencing system
Code Implementation
Set up GitHub
Documentation
Use concrete architec to inform dev
Identify unimplemented features
Reference using requirements ID
Risk Assessment and Mitigation
Choose risk format
Justify format, level of detail
Tabular format of risks to project
Risk likelihood
Risk Impact
Risk mitigation

Below is a Gantt chart showing the key tasks, their starting and finishing dates, and highlighting a critical path through the project -- finally leading to project delivery a couple of days before the deadline. Using the Gantt chart, it's also easily possible to identify key task dependencies as each piece of our solution builds from another (aside from, of course, the initial ones).


October 2020
November 2020
Requirements {Julia, Andrea}
Identify Requirements
Customer Meeting
Document Requirements
Method Selection and Planning {Dougie}
Decide approach
Outline Software Methods
Project Plan
Risk Assessment {Luke}
Choose format
Document Risks
Architecture {Justin, Jack}
Tools
Abstract
Concrete
Justification
Implementation {Everyone}
Website {Luke}
Testing {Everyone}
Julia, Andrea
Dougie
Luke
Justin, Jack
Everyone