Method Selection and Planning

Group 9

Pluto Pioneers

James Lawson, James Rooke, Dema Williams, Nicholas Barker,
Sam Slade, Joshku Gauntlett, Seiya Ikeda

Selection and Justification of Software Engineering Methodology and Development Tools

Our team has adopted an Agile development methodology, notably practices similar to Scrum, which includes “regular meetings, continuous communication with clients, transparency in progress, and consistent retrospective meetings” [1, pp. 148-165]. The choice of this methodology enhances the project's flexibility and strengthens the capability to swiftly respond to changing requirements. Additionally, it fosters cooperation and communication within the team and supports a continuous improvement process.

We utilise GitHub for code version control and task visualisation, Google Drive for the dissemination of documents and deliverables, Discord for real-time communication, and Metro Retro for conducting retrospective meetings to deliberate on project progression and areas for improvement. These tools align with our work processes and the Agile development methodology, particularly in augmenting transparency and fostering efficient teamwork and collaboration.

The use of Git as our version control system plays a pivotal role in balancing the need for privacy with the intrinsic value of transparency within our development team.

Transparency, in our project’s framework, is not about public visibility but rather about the clear visibility of project changes, decisions, and progress within the team. Git enables this internal transparency by providing a comprehensive history of code changes, who made them, and when they were made. This historical record is invaluable for understanding the evolution of the project, diagnosing issues, and facilitating effective collaboration among team members who are working on different aspects of the project simultaneously.

Additionally, the adaptation of Git supports our Agile methodology by enabling iterative development and frequent integration. Despite the project being private, the team benefits from Git’s robust branching and merging capabilities, allowing for seamless collaboration. Each branch can represent a new feature or bug fix, enabling isolated development that can be easily integrated back into the main project without disrupting ongoing work. This approach enhances our team’s efficiency and allows for a continuous improvement process, where the game’s development progresses in a transparent, manageable, and cohesive manner.

For our use of this website, we decided it must be accessible for the duration of the project and free to use. There are many options available at our disposal for this, however, we have decided to use GitHub Pages. We looked at other options, such as Kamatera and Hostinger. Although these seem like good options, they are only free for a limited time and we would run the risk of our website being unavailable if we were to use either of them for this project. The main reason we chose to use GitHub Pages is because it is easily built into the GitHub Project, allowing us to work on both the game and website simultaneously.

Our selection of primarily complimentary tools was driven not solely by budget considerations but also by the need for efficiency and transparency. Upon reviewing various alternatives, these tools were adjudged to best fulfil our team's requirements.

From initial discussions there were 3 clear options for the IDE we would use to create our project: VS Code, Eclipse, and IntelliJ.

Despite us all having previous experience with VS Code, it was determined that we would use one of the other two options primarily because Eclipse and IntelliJ specialise in Java development.

Outlined in the article ‘IntelliJ vs Eclipse: Which is better for beginners?’ [2], Eclipse is the open-source version and is also more customisable. Additionally, eclipse is stated to have a better community to go to for help.

Despite the article stating that IntelliJ needs to be purchased, there is a student plan available which allows us to access it for free. Additionally, out of the box IntelliJ has a greater number of features that will effectively help us with our project.

As such, it was justified that we would choose IntelliJ as our IDE of choice.

We selected primarily available free tools, not merely due to budget constraints but for efficiency and transparency. Other tools and services were considered; however, these were chosen as they best fit the team's needs. GitHub and Discord, in particular, were determined to provide the greatest benefit for both the development process and team communication.

With the Agile development methodology and our chosen tools, we are propelling the project forward effectively and efficiently, deeply ingraining these choices into our workflow and Agile practices, thereby contributing to the fulfilment of project objectives, enhancing team efficiency, and elevating the quality of the final product.

Approach to Team Organisation

Our approach to team organisation has evolved throughout the project, as we became more familiar with it and each other, and learnt what worked and what didn't work. We had to find the balance between too much organisation, which would be cumbersome, and too little, which would result in a lack of coordination [1, pp. 148–165]. We also had to consider that we would not all be working 9-5 in the same office, so couldn't rely on simply observing what other people were doing to stay up to date. Conversely, spending huge amounts of time on documentation and needless processes would be wasteful, as the team only consists of seven members, and the project is not safety critical.

We had at least two meetings each week, which we used for reviewing our progress, planning, and deciding on our next steps. These meetings were held in-person and on campus, as we all felt that this would be the most productive and engaging way of holding them, as well as providing an opportunity for team bonding, and this feeling is backed up by textbooks [1, pp. 148–149].

We used a digital Kanban board on GitHub Projects to keep track of our current work. During our meetings, we identified what work needed to be done, broke it down into specific tasks, and put these tasks on the board. The board had different columns representing the different stages a task may be at, including backlog, ready, in progress, blocked, in review, and done. This was useful because it allowed everyone to see, in one place, the work that needed doing. We assigned the tasks to different people, which was also shown on the board. This helped to make it clear what everyone was responsible for. Breaking down what needed doing into specific tasks made it clear what each actually involved, and when each could be considered done. It also allowed us to easily distribute work.

As part of our workflow, each item of work gets reviewed by another member of the team. This provides a number of benefits. Firstly, it improves the quality of the work we produce, as the reviewer can identify potential mistakes or suggest improvements. Secondly, it ensures that more than one team member is familiar with each aspect of the project, which is important in the event that a member becomes unavailable (bus factor).

To enable us to reflect on how we think the project has been going, and to allow us to improve our processes, we held a retrospective after the first few weeks, and will continue to do so periodically. As a result of this retrospective, we decided that we needed to be communicating more online between our scheduled meetings with regards to:

- Helping each other on our allocated tasks.
- Requesting work be reviewed.
- Deadlines.

We also agreed that in order for our meetings to run more smoothly, we should have a clear plan of what we will be doing in them, before we start. We agreed to start marking tasks with due dates to help keep the project moving forward at the required pace. Finally, we decided that if there are tasks where there is not an obvious person that should work on it (for example because of prior experience or because they do not have any work assigned to them), then we would use a random name picker, to help spread the work out equitably.

Systematic Plan for the Project

At the start of our project, we created a work breakdown chart using PlantUML. We did this by reading the assessment brief and product brief and breaking the project down into smaller, more manageable tasks. The purpose of a breakdown chart isn't to provide a timeline to complete the tasks, but rather to understand the key activities. This allows us to have an understanding of what the main hurdles and roadblocks will be before we create our projected timeline for completion of our project. We were more confident in our breakdowns of the sections that we would undertake towards the start of the project compared to the ones towards the end.

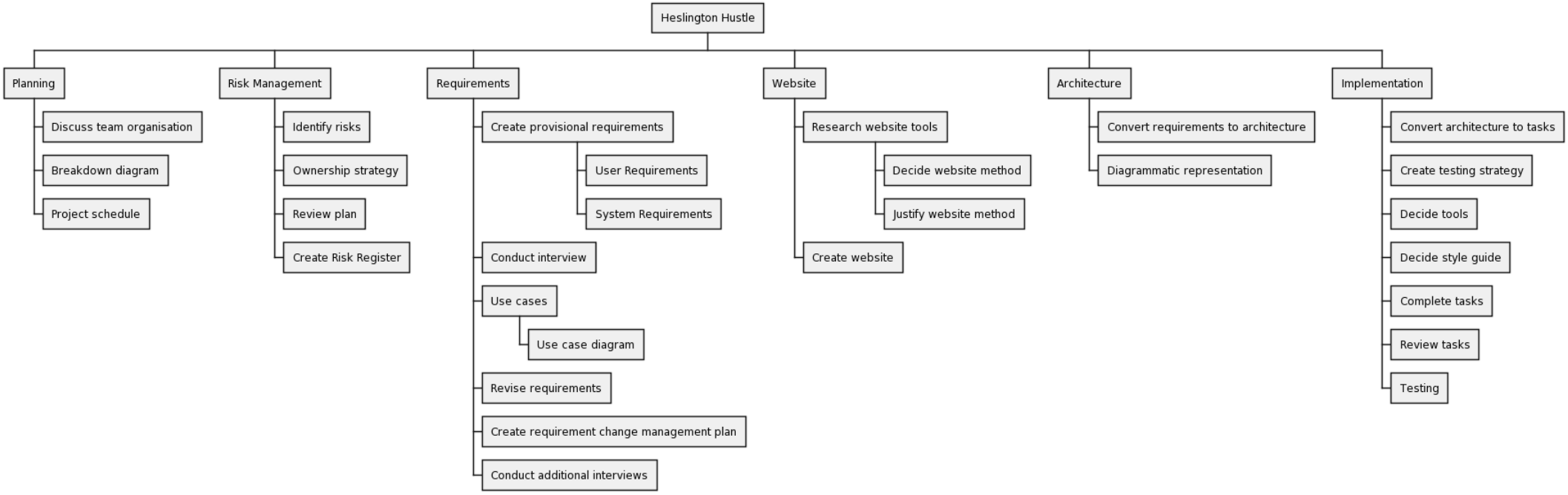

Figure 1. Work Breakdown diagram

After completing our work breakdown chart, we then started work on the Gantt chart, also using PlantUML. The main purpose of this was to 'visually represent a project schedule'[3] from the start of our project to the end date. As it was our first plan, we were not expecting to be strictly sticking with the dates outlined on the Gantt chart, but rather use it as a guide and edit the chart using UML when problems arose. The darker colours chosen show the broad sections, signified as headers in our breakdown chart, with the smaller, more manageable tasks signified with the paler colours.

In our initial version of the chart, we used the start date of the first in-person practical, Thursday the 15th of February and the projected end date of the submission deadline, Thursday the 21st of March. Overall, this gave us 5 weeks to complete this section of the project. We predicted that the most time consuming section would be the implementation, and that we would not be able to begin work on it until all the other sections had been completed. The cascading design loosely shows that some work was dependent on other work, although much of the planning and risk management could be worked on alongside other areas.

As the project progressed, we found ourselves mostly sticking to the plan outlined in the original version of the Gantt chart. This was the case until we started to work on the Architecture, and found that figuring out the game logic and class diagrams would take longer than first anticipated. As a result of this, we pushed back the beginning of our implementation by 3 days and removed the dedicated testing portion of the plan. Furthermore, we realised that after completing the implementation portion of the project, we would need at least a day to compile our work onto the website, proofread and make any changes necessary.

This brings us to the final edition of the Gantt chart, where the plan for the project has stayed mostly the same. The final date has been brought forward to Wednesday the 20th of March. But overall, the breakdown plan and initial Gantt chart created in week 1 proved to be very helpful for us as a group to delegate smaller tasks and give appropriate deadlines for that task.

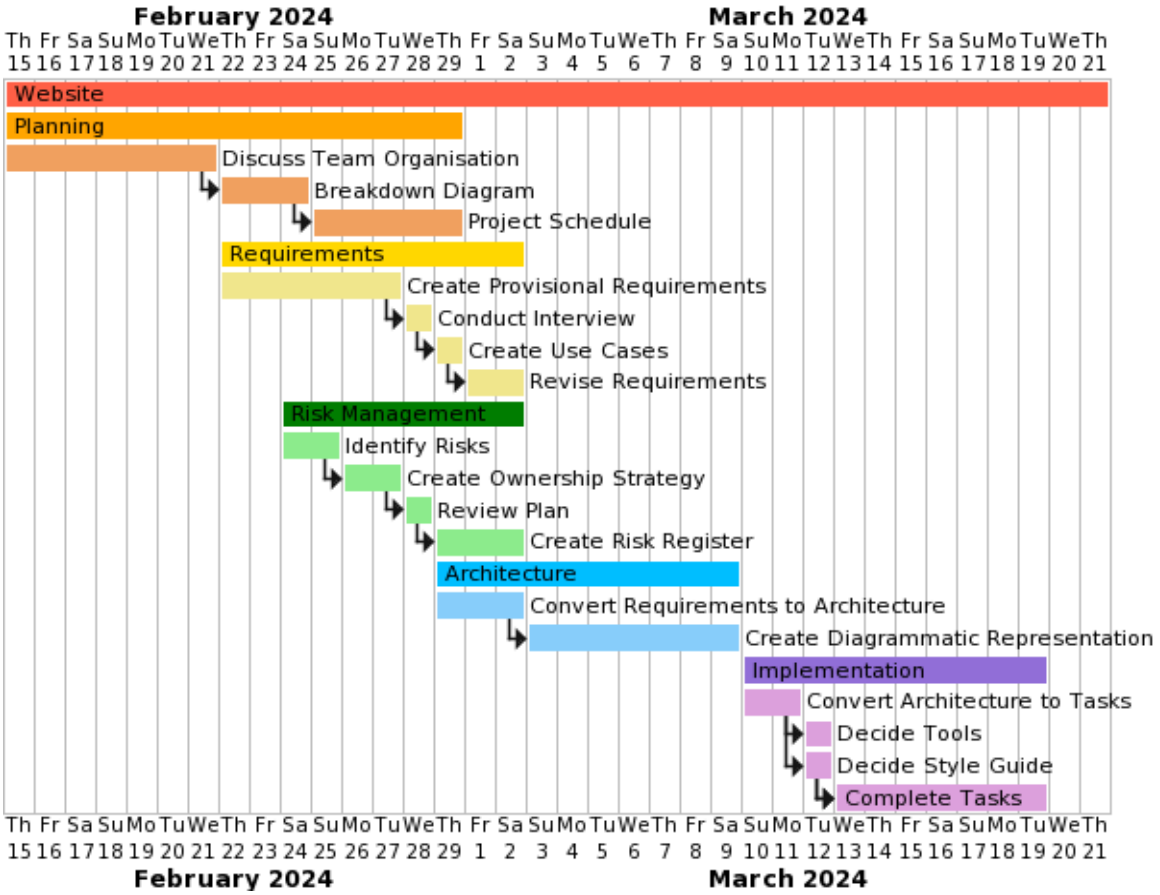

Figure 2. Final Gantt Chart

Reference list

[1] A. Cockburn, *Agile Software Development*. Harlow, England: Addison Wesley, 2002.

[2] 'IntelliJ vs Eclipse: Which is better for beginners?'
https://www.mygreatlearning.com/blog/intellij-vs-eclipse/

[3] PlantUML Gantt Chart
https://plantuml.com/gantt-diagram