Method Selection and Planning

Team 5

Lizzard Entertainment

Gregory Binu
Lydia Eaton
DJ Fox
Daniel Maffei
Michael Papafilippou
Nathalie Rizzo

Software engineering methods
The team has chosen to use an agile methodology for software engineering. This was done as we believed it would allow us to best fulfill the project and stakeholder requirements, due to the collaborative and iterative nature of agile principles, which would allow us to constantly review and update our work to match the brief. We had considered using waterfall methodologies instead, however we felt that the high dependency and lack of flexibility [1] would make it more difficult for us to achieve top marks in the deliverables. The iterative approach used in agile also gives us the ability to adapt our project when stakeholder requirements are changed, further prioritising the requests of the stakeholder to deliver a project that satisfies their needs. Additionally, the continuous integration promoted by agile methodologies was a key part of our decision to implement them, as they reduce the severity of merge conflicts. This is because conflict between two new and complex pieces of code is more difficult to resolve compared to conflict between minor additions to existing code, since the former may require larger portions to be rewritten and a greater time investment.

Specifically, the team plans to use the Scrum framework, which is an agile framework that emphasises the deconstruction of large and complex tasks into smaller, more manageable sub-tasks. These tasks are to be worked on by members of the team in time-boxed periods called sprints, and the work done in them is reviewed in sprint review sessions [2]. We believe this framework is best as it provides a practical approach to implementing agile methodologies. Namely, the team collaboration which is fundamental to agile methodologies is handled by the goals established and assigned in sprints, and the feedback received during sprint review sessions. Moreover, continuous integration can occur through the creation of sub-tasks, as this will allow us to keep delivering small pieces of code and minimise the risk of large code conflict setbacks.

Our method selection to develop the code was responsibility-driven design, as we believed this would make our project the easiest to develop and more scalable for teams that pick it. One of the key principles that informed our choice to use this software development method was the ability to create consistent control styles for objects [3], making it easy to create hook methods and thus making it easy to work with classes. Responsibility-driven design would also distribute game logic between different classes, preventing the creation of a “god object” [4] which would lead to more difficult-to-maintain code.

To support the team’s work we used Git which is a distributed version control system [5] that allows for multiple developers to simultaneously work on the same project by pushing commits and merging branches. For our group work, this is very helpful as it allows us to continuously integrate code by pushing commits when changes are made, meaning that we all have easy access to each other’s work. Following agile methodologies by iteratively improving our project is also made simple, as code can be updated by just switching to the branch and pushing changes, thus making it easy to change existing sections of code to make improvements and adapt to changing stakeholder requirements.

To host our code we used GitHub as it offers the ability to create repositories and easily integrates with Git. Using GitHub allows for greater team collaboration as we can leave comments and reviews on each other’s work using pull requests, requiring multiple team members to verify and approve any changes to the code. GitHub also helps us incorporate the principles of the Scrum framework by making use of GitHub “issues”, where we can split large tasks into smaller tickets to be resolved individually. We had initially considered using Jira for this, as the platform is more geared towards boosting productivity through tickets and offers a greater range of features, however we decided that using issues on GitHub would be easiest as everything would be in one place, as opposed to splitting work between Github and Jira.

For online team communication we used Discord, which is an app that allows users to create servers with dedicated channels. These dedicated channels are the primary reason we chose Discord, as they provide an organised way to hold discussions pertaining to the relevant deliverables. We had considered using WhatsApp as an alternative, however the lack of channels meant that if multiple people were talking about different aspects of the project at the same time, that would lead to the chat becoming congested and limiting productivity. It would also be more difficult to find past conversations, whereas in Discord it is possible to just scroll through the relevant channel rather than needing to go through the entire group conversation as would be required with WhatsApp.

To write up our work we used Google Drive, which is a cloud platform linked to Gmail. The reason we decided it was the best tool to use was because of the ability to easily collaborate on Google Docs at the same time, making it easy to split up work between people working on the same deliverable. Google Docs also keeps a timestamped record of all edits, which could be used to verify that all team members are contributing equally. Due to many team members being more familiar with Microsoft products, it was briefly considered that we would send each other Microsoft Word documents via email, however this was dismissed primarily due to the inability for people to work on the same document at the same time.

Team organisation
During our first few team meetings we assigned team roles to all group members, with the aim of all members completing approximately the same marks of work. We felt that doing this would offer an opportunity for people to share their interests and strengths so that they could be assigned to the relevant work accordingly.

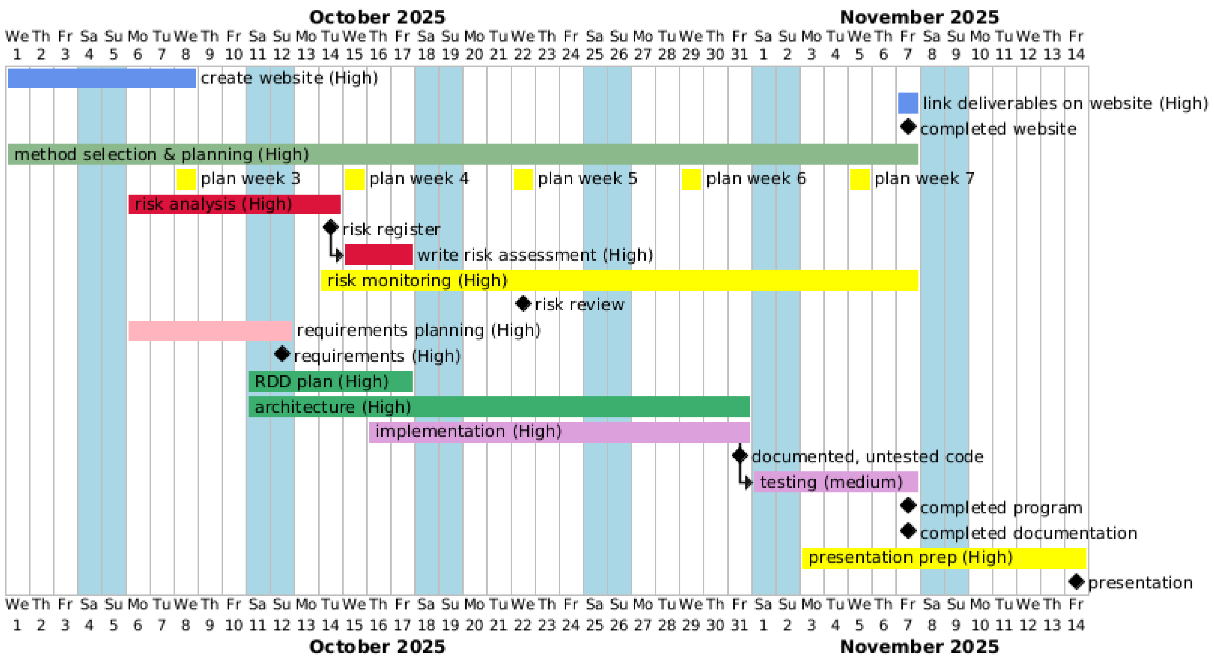
Ultimately, we decided on appointing a team leader who would oversee everyone's work and act as a first point of support when in need of help. For each deliverable, there was one main person who was tasked with most of the marks, along with one or two others who did smaller portions of the deliverable and were in charge of proofreading and editing. For the purposes of the project, we felt this was best as it not only kept the mark distribution fair, but also meant that all work was monitored along the way by both the team leader and the supporting individuals so as to promote maximum quality. Careful attention was placed when making sure that all group members were doing approximately 15 marks worth of work for Assessment 1, which meant that those who had large roles in deliverables with many marks (e.g. implementation) would be tasked with doing less work for other deliverables, and vice versa.

Work was assigned during our first meeting of the week to be completed by the next sprint, which was two weeks in length. Following the principles of Scrum, deliverables were split up into their constituent parts to be worked on, and so for these sub-tasks shorter deadlines would be placed (e.g. by the next meeting in two days). Sprint reviews took place right before the meeting, creating work for the next sprint, as we felt this would allow us to review work and then make informed decisions about next steps based on the outcomes of the previous sprint. Doing this we were able to continuously refine our approach to teamwork, as we became more familiar with each other's strengths and weaknesses and how to best tackle problems as a team.

During sprint meetings, the emphasis would be on checking whether all tasks have been completed to satisfactory standard, whilst also keeping in mind that agile work is an iterative process and hence amendments can be made at a later time. To check the quality of work, the team would have a look at the work of each member, meanwhile the leader would already have had a look at everyone's work in advance and would be able to bring up more concrete points of discussion for the team. Everyone reviewing all pieces of work would lead to very long and inefficient sprint reviews, so the team leader having some points to cover prepared meant we were able to be much more efficient.

Systematic Plan

The Gantt diagrams which we created after our meetings can be found here. These diagrams were used as a way to clarify all the tasks that were due, along with the corresponding priorities shown in brackets, and dependencies which are indicated by an arrow pointing from the end of one task to the start of the next. Over time, the plan had fewer dependencies, as people were working on individual subtasks following the principles of Scrum, whereas in the beginning a lot of tasks had to take place before others (such as the stakeholder meeting being a dependency for the user requirements). After the development process began, the main dependency was having a working game prototype we could add features to.



The above Gantt diagram shows the initial plan made at the start of October for our team's deadlines; it was meant to be more provisional and used so everyone had a better idea of what they were doing, and provided a clearer idea of the priorities and dependencies involved in the project. Certain priorities had to be dropped due to time constraints, such as the plan to write unit tests going from medium to low; our team wanted to implement these tests to make it easier for teams continuing our project to develop it, however it became apparent that time would not allow for us to do so.

The most flexible deadlines of the project were those that involved gaining familiarity with libGDX game development; as a team we thought it was important to gain some experience using it and develop some simple games to improve our skills, however these deadlines kept getting pushed back as the team focused instead on the deliverables, which were the priority.

The initial deadlines also did not take agile methodologies into consideration, namely the process of iterative development. To counter this, in revised versions of our plan, there was extra time allotted to this process in order to ensure continuous delivery and improvement.

There were some deadlines that had to be extended due to certain unforeseen circumstances, such as the absence of team members or in the event that members did not do all the work they were assigned in time. These deadlines were generally extended to be completed by the next sprint review session, so the team could review the completed work together. In particular, towards the end of the project when there were several additions the team had to complete before the deadline, many deadlines had to be extended as all members were overloaded with more work than anticipated due to things like merge conflicts. This also meant that, to ensure the completion of the Github tickets, equal time had to be dedicated from all members as opposed to the individuals who were initially tasked with the majority of the work.

Due to different variables such as sickness and delayed work completion, the plan was iteratively changed to reflect these real-world limitations into the following final key tasks:

Task #	☐ Task	Priority	Start Date	End Date	Dependencies
1	Create website	HIGH	October 1st 2025	October 3rd 2025	
2	Add deliverables to website	HIGH	October 1st 2025	November 7th 2025	1
3	Plan and conduct stakeholder interview	HIGH	October 1st 2025	October 10th 2025	
4	Write risk assessment	HIGH	October 3rd 2025	October 8th 2025	
5	Set up libGDX	HIGH	October 8th 2025	October 10th 2025	
6	Write method selection and planning	HIGH	October 10th 2025	October 22nd 2025	

7	Write user and system requirements	HIGH	October 10th 2025	October 17th 2025	3
8	Write and improve architecture	HIGH	October 10th 2025	November 7th 2025 (Core architecture to be completed by October 17th for game prototype)	
9	Make game prototype with one event	HIGH	October 17th 2025	October 22nd 2025	8
10	Add remaining events to game	HIGH	October 22nd 2025	November 5th 2025	9
11	Add timer to game	HIGH	October 27th 2025	November 3rd 2025	9
12	Add music to game	HIGH	October 27th 2025	November 3rd 2025	9
13	Add initial unit tests	LOW	October 27th 2025	November 9th 2025	9
13	Add start, win, lose, and pause screens	HIGH	October 29th 2025	November 3rd 2025	9
14	Add score calculator	HIGH	November 3rd 2025	November 7th 2025	11, 13
15	Add informational messages to events	MEDIUM	November 5th 2025	November 7th 2025	10

References

[1] D. Raymond, “Top 10 Cons or Disadvantages of using waterfall Methodology,” *ProjectManagers.net*, Jul. 28, 2025. https://projectmanagers.net/top-10-cons-or-disadvantages-of-using-waterfall-methodology/
[2] Atlassian, “Scrum sprints Article | Agile.” https://www.atlassian.com/agile/scrum/sprints
[3] P. B. / R. H. G. / 503, “Wirfs-Brock Associates Responsibility-Driven design,” *2004 Wirfs-Brock Associates.* https://wirfs-brock.com/Design.html
[4] A. Garcia-Dominguez, “Responsibility-Driven Design”, vle, https://vle.york.ac.uk/ultra/courses/_116044_1/outline/edit/document/_6367046_1?courseId=_116044_1&view=content&state=view
[5] “git Guides,” *GitHub*, 2025. https://github.com/git-guides