Method Selection and Planning

Team 26

devCharles

Ross Holmes
Sabid Hossain
Thomas Pauline
Joel Paxman
Andrey Samoilov
Louis Warren

Software Engineering Methods

The software development framework we used was a modified version of the Waterfall model that meant a stage can be started before the previous stage is completed.

- We chose this over XP as we had one deadline with one task and were not expecting to have the project's requirements changed and also didn't use scrum because:
    - We couldn't assess a complete product at all stages of development which made finding tasks more difficult.
    - due to other modules we couldn't stick entirely to scrum practices, such as daily standups
- We used waterfall instead because this project is short term and the dependencies were direct and continuous.
- Waterfall also consists of considerable documentation, which means minimal team member knowledge is lost if a member was to become unavailable for whatever reason.

The Java framework we decided to use was libGDX.

- We also considered a few others - JmonkeyEngine, LWJGL, GODOT to name a few.
- libGDX was the most:
    - Mature
    - Widely Used
    - Extensible
- LWGJL was another option, but libGDX uses it in its framework and is a higher level method of accessing it.
- It also consists of mainly regular java, which the team was already familiar with.

For our IDEs we decided to use intelliJ and visual studio code. Visual studio code because:

- Everyone in the team used this IDE.
- Its live share feature was incredibly convenient for collaborating together during meetings even whilst not physically together.

IntelliJ was also used as intelliJ is developed specifically for java and so has greater "intelligence" than VScode. We considered using eclipse as an IDE however as it was not as capable as intelliJ and the team was not as familiar with it as intelliJ so we decided it was not worth the team's time to use this IDE. Since GitHub allows for different IDEs to be used easily, we believe that both ideas would be best used in tandem.

Tiled was used as a map creator for libGDX as it is the most supported map editor so this was chosen due to the fact it had more tutorials and was a simpler tool then any competing product.
We then decided that we needed a version control system to upload code and enable every member of the team working on implementation to be capable of developing the code simultaneously.

We decided on GitHub for this purpose.

- Easy design
- Its high market share means that the team could have experience with real world tools required in software development.
- We could sign up for free (as we are students) to GitHub premium. This allowed us to have web based support in case of any issues we encountered.
- Crucial features like being able to have protected branches and having required reviewers. This reduced our risk of:
    - R4: A file is accidentally deleted or corrupted
    - R11: A group member accidentally commits directly to main
- This reduces the chance of us losing changes made by others or introducing merge conflicts as now it needs two group members to review and commit these changes.
- Tracking of who and when team members create changes to their work, which allows for a simple way to track which members have contributed individual parts of the code.

As a team it was decided at the beginning of the project to create a project schedule, with tasks set for each member of the team and deadlines. In order to support this we decided to use a gantt chart software system. We considered several tools, one of which was Asana Gantt chart, however this software was not available for free after 30 days so we decided against it.

We continued researching possible alternatives until we found Monday.com that was:

- Free to students.
- Had multiple user access so that everyone in the team would be able to modify and see gantt charts.

This helped us organise the task and helped us visualise the plan and set clear objectives for when tasks should be completed.
Later in our project we realised that an issue had occurred with monday.com and so we no longer had permission to change/update our gantt charts, this led us to have to reconsider how we could assign tasks. We then decided to use PlantUML as it is an open source gantt software capable of creating the gantt charts we required.

However it wasn't as fully featured as Monday.com so we agreed we required a secondary task management software. GitHub also has a Kanban Board management system (work items are shown as cards, which are moved left to right to show progress) that is fairly capable, and due to the team's now frequent usage of GitHub (and therefore familiarity) we decided to use this system.
Because we planned to have regular meetings, we required a tool to communicate. We discussed several different platforms:

- Slack - some members of the team were unfamiliar with slack and slack had some features barred unless we used a premium service we decided against it.
- Whatsapp - less tools for communication.
- Discord

After some discussion we agreed on discord.

- Discord allows for:
    - Different channels to be used for different topics and whatsapp would only allow for one group chat.
    - Sharing of desktop screens allows us to have much more productive calls with significantly more collaboration.
    - Voice calls for meetings .
    - Messaging for organisation both to individuals and the group as a whole.
- Overall we decided on Discord due to these reasons and since everyone already had a discord account, so most of the team thought it was the best option.

Because of discord we were able to have regular meetings - towards the beginning this was around twice a week and closer to the deadline this became multiple times a week (up to everyday) - both in person and online to:

- Discuss everyone’s progress
- Adjust the plan based on this
- Set goals for the next meeting

We would do most of the work together during the meetings but some of our work was done independently before or after them.
In order to complete our documentation we decided it would be best to use software such as Microsoft Word. Since every individual in the team had extensive experience with Word we heavily considered using it. We preferred google docs for several reasons:

- Sharing documents was easier
- Live collaboration was simpler
- Our team had more experience with google docs

So whilst it is less capable than word it has significantly better collaboration capabilities. This made it fit with our engineering methods very well which ultimately made us choose this platform over Microsoft’s.

Team Organisation

Originally we didn't formally have a “leader” however after a few practicals Andrey became the leader of our group. To decide who would work on implementation and who would work on documentation it was decided that Ross, Andrey and Louis would start on implementation to focus on learning the technical sections of the project, and as some already had some experience.

That meant the other three team members could focus on:

- Defining software architecture from the already elicited requirements.
- Carry out development planning.
- Identifying any risks and organising their mitigation.

We felt it was suitable to split the group into two teams so:

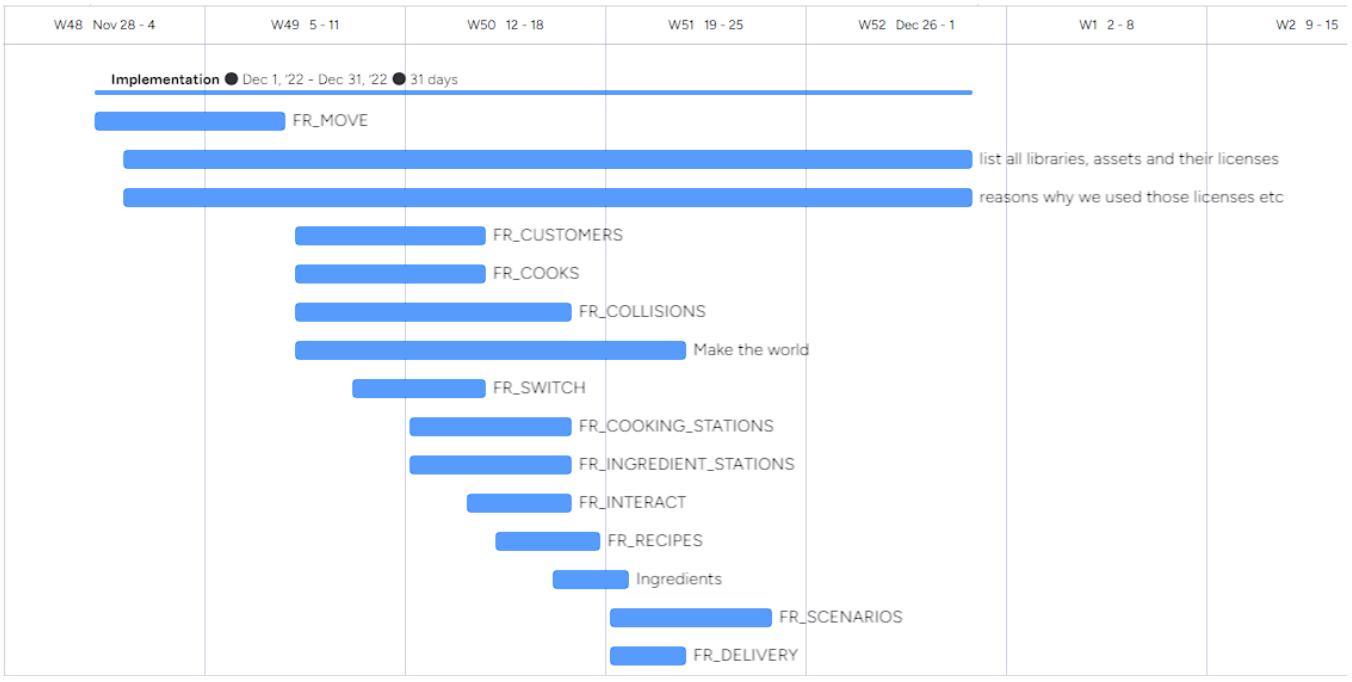
- More focused work can be done on individual strengths and to increase productivity.
- Each team member works on multiple sections of the documentation, rather than every section being the responsibility of one person.
- Different members would complete a base idea for what the section would look like, with other people then assisting by filling out details and ensuring accuracy.

This method also increases the bus factor, because we would then have multiple people working on each section at once so progression would be as consistent as possible. To maintain a high bus factor Louis also did some documentation so that if an issue arose (such as several team members being unsure of what to do) then he could fill in.

We considered different organisation systems such as all members work on implementation, however this didn’t happen in the end due to the massive amount of learning required to become capable of contributing a sufficient amount of work for the time invested so instead we spread out the required learning, to make sure less human hours would be spent on figuring out how to do each section.

Systematic Plan

To make our initial plan we had a meeting with the entire team where we discussed what needed to be done and gave dates when we wanted each task to be done by. We met around twice a week at first and as the deadline got closer these meetings became more frequent. All our meetings were noted on a document that keeps track of when the meeting was, who was there and has a brief overview of what was discussed during the meeting.



This was the initial plan for progression on implementation of the project. This consisted of:

- A list of functional requirements in chronological order based on dependencies of what they required to be completed.
- Tasks which required being continuously updated in order to be completed, for example "List all libraries, assets and their licences."
- More broad ideas, such as "Make the World" which don't fit into either of the other two categories.
- An organisation method that:
  - Allowed every team member to have clear tasks and deadlines
  - Work could be spread out over a large period of time, therefore preventing the requirement of "crunches" or bursts of work towards the final deadline.
  - Included a mindfulness of every team member's individual availability so that people would be able to realistically achieve their work within the deadlines.

Implicitly shown on this graph are the priority of tasks, as previous tasks must be completed before other tasks can be completed. For example movement must be implemented before customers, cooks or collisions can be implemented.

A difficulty of giving timeframes with this method is the unknown development times each step could take. There is a fairly large amount of time allocated to FR_MOVE for example, and we quickly realised that it was much simpler than we expected, but things like FR_CUSTOMERS took much longer than expected as they were much more complicated than previously thought.

However, as we found it was difficult to organise free time over the holiday and some people were difficult to get in contact with, we decided it would be best to reorganise on returning to university with the plan of every task being completed as soon as possible, and when they were completed to move immediately onto the next available task (without incomplete prerequisites). This does not mean that no work was done over the course of the holiday, just that it would be easier logistically to continue back in term time.

To assist with this we made an emergency document detailing each task to be completed - within which people would assign names to and then reassign to tasks to be completed that were not yet done - and use GitHub's project management system in which tasks were sequentially:

- Created when a new task was found.
- Taken up by anyone free or who wanted to do it.
- Put up to review upon completion.

GitHub was used throughout the project, but became the main way to manage tasks when the Gantt chart became less useful for doing tasks as quickly as possible.

The benefits to this method were:

- It was easy to get a grasp on how far people had gotten in work.
- Simple proof-reading
- Easy to raise an issue for help with bug fixing.
- Multiple people can be assigned to tasks both at the start and any time during the progress.
- An integrated method of seeing which members contributed to or completed which tasks.

A plausible downside to this method is the lack of an included given timeframe for each task. This could cause issues in large projects over long periods of time, where each task could have many dependencies.

In this project this downside could be considered fairly minor, as there are plenty of tasks without complete dependencies and these could be completed during the time of waiting for others to finish the task which you need to be done. Minimal downsides and the benefits outweighing them means that we considered this method most suitable for this project.