Method Selection and Planning

Group 4

Isabella Miles
Maciek Racis
Sally Finnon
Sophia Eaves
Wrijurekh Mukherjee
Xavi Murtagh Molina

By evaluating its characteristics (as shown below), we have decided an agile methodology will be the most appropriate for our project.

Agile takes an incremental approach by releasing smaller, but frequent deliverables. Although this requires more comprehensive planning upfront, taking the time to split the project into smaller, more manageable pieces allows for abstraction, and helps us focus on what is important. Saving time in the long run, which is vital for a small project such as ours. Furthermore, smaller deliverables are easier to fairly distribute across the team. As NB. Boehm and Turner point out, Agile methodologies are successful with small groups, such as ours. Additionally, having frequent checkpoints is also helpful. It is not only motivating, but also it makes keeping track of progress more simple. This will help us identify and solve issues quickly, saving time.

The second feature is cooperation, i.e., embedding the customer in the team. Having regular contact with the customer will provide us with contiguous feedback, and it is easier to make improvements during the development process, rather than at the end. As it will disrupt fewer other sections and we will have more time to make the required changes than we would any closer to the final deadline. Consequently giving us the best chance of properly satisfying the customer with our product.

Thirdly, agile is straightforward to follow and document. Due to our condensed time frame, an easy to learn method is essential as less time spent researching the method gives us more time to complete the actual project. Furthermore, simple documentation will be very important when it comes to the second part of the module. Where another group of developers will take over and need to be able to understand what we have done so far.

Finally, agile methodologies are adaptive. Seeing as some changes or problems are bound to occur during the development process (e.g., changing requirements, user feedback or technological problems). A method that allows us to adapt quickly to these problems is vital in ensuring we develop the best possible solution to the user requirements within the timeframe.

The collaboration tools:

We have found GitHub to be a fast and efficient way of sharing code. Using a login, any group member can access the code anywhere, from any device.

For sharing and collaboration on documents, we have chosen Google Docs. Similarly it is easy to use and familiar to all of our group. Additionally, it has a history feature allowing anyone with access to the document to view who has edited the file and when. This built in documentation can act as some evidence of each member's contribution. Which can be shared with the module leaders should any issues arise with regards to the peer assessment. Moreover, the ability to create shared folders makes it more suited for group work in comparison to other word processors, e.g., Microsoft Word.

Creating a comprehensive plan to keep track of our progress is incredibly important, for this we have chosen to use Lucid to create a Gaant chart. It provides us with a good visual representation of our plan. In particular, a Gaant chart allows for lots of checkpoints and dependencies to be added, which is vital for our agile development methodology. The use of colour coding also makes it easy to see who has been allocated to each section.

We are also using Whatsapp for informal communication. It is convenient, and already regularly used by all the team members.

Development tools:
opengameart.org is the largest repository for community developed free 2d assets, giving us the largest range of options for our game possible for free.

VSCode is useful as due to its use in previous software modules at the university, all of our group is very familiar with it. This saves us time adjusting to another IDE such as Eclipse in our short timeframe

LibGDX is the most popular framework for Java game development because it is the most intuitive and easy to use. Particularly for beginners with no formal game development experience, like most of our team.

Tiled is a Java library used to create and edit 2d Maps. It is flexible and easy to edit, additionally it is familiar to most of our group members.

GL20 is a popular graphics library. We have chosen to use it because it is comprehensive and easy to use.

SpriteBatch simplifies and optimises the task of drawing images. which is time saving and efficient for our project.

With regards to team structure, we decided to have a flat team structure without a designated leader or such roles. Everyone appears to be conscientious and committed to delivering the project on time and to a high standard, so we did not feel we needed someone specifically to direct everyone else.

As we don't have a leader, we consciously make sure that everyone participates in meetings; we all share ideas equally and everyone has a chance to contribute. In addition to this, we ensure that someone is taking minutes of the meeting, that we can refer back to, each time we meet.

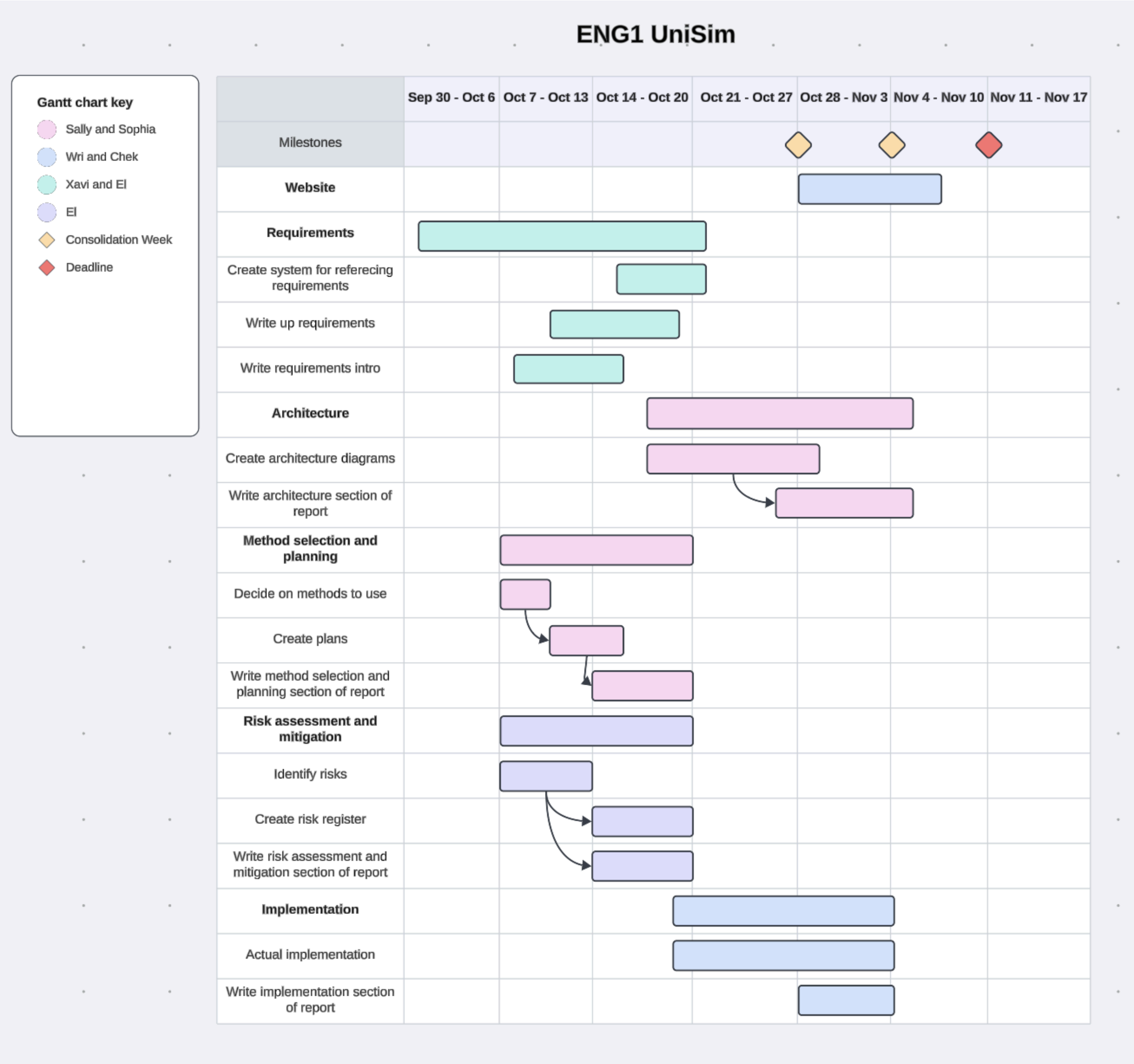
We did, however, assign different deliverables to different people. To do this, we discussed each other's strengths and weaknesses, then divided up the deliverables, also considering the number of marks each deliverable is worth.

For example, Wrijurekh and Maciek's strengths lie in programming, so we assigned them the implementation deliverable, as well as the website, to ensure they were contributing close to fifteen marks' worth of material each. El is knowledgeable about risk assessment processes, so we assigned her the risk assessment and mitigation related tasks. She is also assisting with requirements, as this is another of her strengths, and will ensure she contributes the correct number of marks. Xavi was assigned to the rest of the requirements deliverable, and Sophia and Sally are splitting architecture and method selection and planning equally between themselves.

As well as being appropriate for the team for the reasons stated above, this flat team structure is appropriate for a project with a relatively short time frame, such as this one. Being a manageable scale, the does not necessarily need someone to act as a production manager and direct the project overall. Everyone is able to work on their individual tasks and then come together to discuss the overall progress on the project.

Moreover, there is not a large number of dependencies between deliverables, meaning that each person can work more or less independently on their own tasks throughout a significant portion of the project. As we are able to complete tasks individually and we have assigned these tasks to certain people, we are able to complete the project in a more efficient manner.

However, there are some critical deliverables to which we have allocated more than one person as a risk mitigation tactic; if one team member were unable to complete a critical task, the other member would be able to pick up the work. For example, we have allocated both Wrijurekh and Maciek to implementation and the website, Sophia and Sally to architecture, and El and Xavi to requirements. For other less critical tasks, another team member would still be able to pick up the work if necessary, if with a little more difficulty.



In order to map out a plan for the project, we created a Gantt chart. We are using this Gantt chart to make sure everyone in the group is aware of what tasks they should be working on and when. It sets out internal deadlines for each task to make sure we are not falling behind schedule and are able to complete the project in a timely manner before the real deadline.

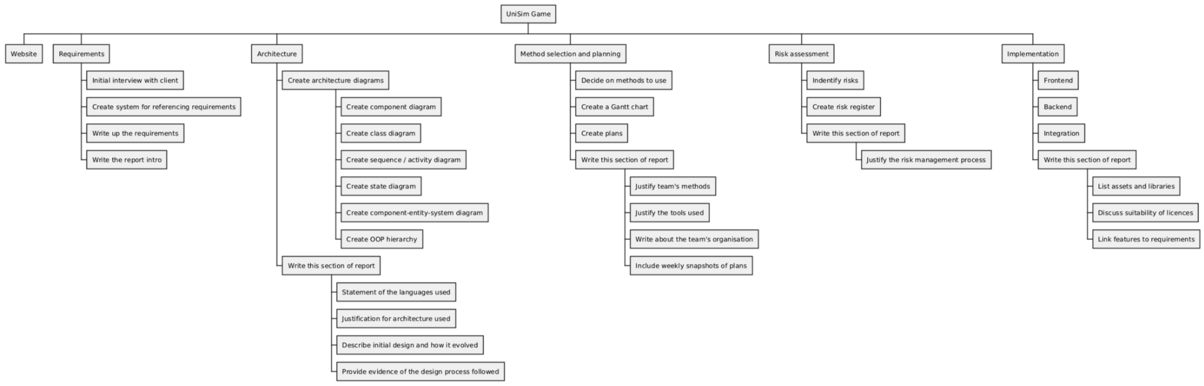
As shown in the Gantt chart above, we laid out the key tasks in the project on the left-hand side and gave each of them a time frame with starting and finishing dates. These dates reflect task dependencies, which are also explicitly shown, as well as our priorities. Moreover, we included a more general time frame for entire sections of the project, such as requirements and architecture. This helps us to see the bigger picture in terms of project time frames.

Another element we added to the Gantt chart to allow us to have a more general overview of time frames are the milestones, visible as coloured diamonds at the top. We added milestones signifying the beginning and end of consolidation week, as well as the final deadline for this part of the project.

We included dependencies where we thought they were needed. For example, we could not create a risk register without first identifying the risks, and we needed to decide on what development methods to use before creating the plans for the project.

The Gantt chart shown above is the final version; we also have a previous version that can be viewed on the website. Our previous Gantt chart only the major project areas, such as requirements and architecture. We realised this was not detailed enough, as there are a number of tasks each section can be broken down into. It was more beneficial to have smaller time frames for a number of smaller tasks than vice versa. Breaking down the tasks further also allowed us to see more specific dependencies.

Additionally, we created a work breakdown diagram to show the breakdown of tasks within the project, and to help break them down into smaller units. The diagram shows major tasks, and all of the smaller tasks that come under the various parent tasks. The tasks shown are not necessarily in any particular order, as that is not required in this diagram.


UniSim Game
Website
Requirements
Initial interview with client
Create system for referencing requirements
Write up the requirements
Write the report intro
Architecture
Create architecture diagrams
Create component diagram
Create class diagram
Create sequence / activity diagram
Create state diagram
Create component-entity-system diagram
Create OOP hierarchy
Write this section of report
Statement of the languages used
Justification for architecture used
Describe initial design and how it evolved
Provide evidence of the design process followed
Method selection and planning
Decide on methods to use
Create a Gantt chart
Create plans
Write this section of report
Justify team's methods
Justify the tools used
Write about the team's organisation
Include weekly snapshots of plans
Risk assessment
Indentify risks
Create risk register
Write this section of report
Justify the risk management process
Implementation
Frontend
Backend
Integration
Write this section of report
List assets and libraries
Discuss suitability of licences
Link features to requirements

The work breakdown aided our team in planning out what tasks should be completed when, identifying dependencies, and thus in the creation of the Gantt chart above.

Although the work breakdown diagram above is the final version, an earlier version can be seen on the website. We found that this version does not break down the, “Create architecture diagrams,” task sufficiently when we reached this section of the project. We therefore updated the work breakdown diagram to specify which diagrams needed to be created specifically.