Method Selection and Planning

Group Number: 10

Team Name: Decassociation

Group Member Names:
Mohammad Abdullah
Tom Broadbent
Poppy Fynes
Owen Lister
Michael Marples
Lucy Walsh

Method selection and planning

We agreed to use the Agile model as the systems development life cycle (SDLC) for project management. Under this, we were able to better address requirements that changed and we were able to respond when the implementation needed to be revised after consulting with the customer.

The Agile principles prioritise "responding to change over following a plan" and demand that "at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly". This was all crucial while working under the given time constraints and requirement volatility. Agile also emphasises how "business people and developers must work together daily throughout the project". Moreover, the three pillars of Scrum are transparency, inspection and adaptation.

Due to the focus on adaptability and regular communication with the customer, we chose Scrum as our methodology. We worked in sprints between weekly team meetings. During these sprints, members of the team worked on tasks they had chosen. Some tasks (such as programming/implementation) were worked on by more than one person at a time; this allowed each of us to work according to our individual strengths as we perceived them. During team meetings, we discussed the tasks each of us had taken on and how we had progressed. Thus, everyone received feedback and it allowed us to decide what tasks should be done next and by whom.

Google Drive allowed us to create a commonly-accessible space for storing files with its feature to create shared drives. Its integration with Google Docs facilitated working collaboratively on deliverables that required word processing, as well as auxiliary activities like taking notes for the meetings. The service being platform-agnostic by virtue of running in a browser environment made it easy to get started, especially as everyone was familiar with it. It keeps a history of all changes made to a document, which allows team members to see who made changes and when. This allows for accountability and transparency in the development process and makes Google Drive a suitable tool for Agile development teams. We used Google Sheets to create the Gantt charts as part of planning.

GitHub uses Git, a distributed version control system that allows multiple users to work on the same codebase simultaneously; this allows for real-time collaboration. It also allows for easy branching and merging of code, which supports the iterative and incremental nature of Agile development. Pull requests allow for code review and feedback before changes are incorporated into the main codebase. This promotes collaboration and accountability among team members. GitHub organizations were used by our team to manage access to the codebase.

To create our website, we used Github Pages, and to be able to work on the implementation of the project collaboratively we used Github Organizations. Making a website using Pages entailed making a repository called *decassociation.github.io*. With Jekyll themes, we did not have to spend much time writing boilerplate code and had a functioning site quickly.

We arranged our meetings and conducted them on Discord; it allows for voice communication and video (with screen-sharing). Everyone in our team was familiar with the interface and features. Discord allows for the sharing of files, images, links, and code rendered in monospace font with syntax highlighting for Java.

We found IntelliJ IDEA to be suitable for this project. Built-in support for Java and support for Gradle facilitated the onboarding of team members. Moreover, it has the ability to quickly navigate and refactor code, making it easy to make changes and iterate on the project afterwards. The alternatives that we considered were Eclipse (due to marginal familiarity with it from lectures) and Visual Studio Code. However, the former did not come with as many features "out of the box" and the latter may have taken time to integrate with different tools/frameworks so we did not use it.

Our approach to team organisation involved collectively identifying key tasks that needed to be completed and the prerequisites to these tasks, as well as determining a timeframe for when they needed to be done. From that point, members of the group picked the tasks that they wanted to work on – while making sure everything identified will be worked on by at least one person, thus ensuring progress.

This could accurately be described as an organic centralism; it emphasises the central role of (a) collective leadership in guiding us, while also allowing for decentralised decision-making and autonomy at 'lower' levels.

Organic centralism is a method for (political) organisation and seeks to balance the need for a centralised leadership with the need for onboarding individual initiative and addressing individual concerns. It is usually in contrast to traditional centralism which emphasises a strict hierarchy, strict rules and a top-down decision-making process. This organic centralism allowed us to be more flexible and encouraged participation of all members in the decision-making process, while still maintaining a central leadership that sets the overall direction. It also encourages the development of a collective consciousness and the unity of the group and can be applied to software development teams because it aims to create a more effective organisation by leveraging the strengths of all members.

This approach allowed us to have good task throughput; if there was something that needed to be done, it was chosen by a person. This is a benefit given the nature of the project, where a stakeholder might need to see progress regularly. A benefit afforded by this approach (with respect to the team members) is the engagement of individual members with the tasks; we chose the tasks we *wanted* to do.

Our general plan was to do development in 4 stages.

1. Conceptualisation – the team should brainstorm different concepts, gather inspiration and decide on things like target audience and platform. The team should also conduct a requirement elicitation and ask the customer about any specifics.
2. Pre-production – we should develop a more detailed idea of the game, including the game’s mechanics, art style, and interface. The team should also create a game design document that outlines the game's features, mechanics, and overall design. This document will serve as a guide for the rest of the development process.

3. Production – we will start building the game. The team will create the game's assets, such as art, sound, and music. The team will also code the game's mechanics and implement the game's systems. This phase will require the team to work closely together and collaborate effectively.
4. Testing (followed by release). During this phase, the team will test the game to make sure that it works properly and is free of bugs.

In the second week of our project, we discussed our plan. As part of that process, we created a list of tasks (product backlog) that would need to be completed at some point during the development of the game. Since we had decided to use Scrum, some of the time during practicals was spent discussing our next steps and delineating these into tasks (sprint backlog) that members of the team could work on over the next week (our sprint duration).

In the below image, it shows that we had a group of tasks associated with architecture design, and these serve as the design documents outlined in stage 2. Some of these tasks grouped together will be mutually dependent, and others constitute 'subtasks'.

	To do/ doing	Done				
Date started: 16/11/2022						
TASK NAME	START DATE	END DATE	DURATION (WORK DAYS)	TEAM MEMBER	M	T
Gantt Charting				Michael primarily		
Talk to stakeholder				Everyone		
-Acquire information from the hostage				Everyone		
-Use information to formulate plan				Everyone		
Method selection and planning				Owen/ Mo		
Write up requirements in a more cohesive form				Tom/ Owen		
Write risk assessment				Lucy Primarily		
Design architecture				Joint team		
-Make diagrams for how it will fit together/ work						
-Specify objects and indicate relations				Michael primarily		
-Do responsibility driven design						
-Use abstraction of functions						
-Make some vague psuedo code						
Decide on assets						
-Style/ Source						
-Acquire said assets						
-Possibly use placeholders at first and upgrade later						

For example, acquiring assets is contingent on deciding on the style thereof. While this relationship is not explicitly stated on our chart, it is heavily implied and semantically apparent.

For weekly snapshots of our plan (i.e. Gant charts and weekly meeting notes which contain progress made and next steps each week), see our website: https://decassociation.github.io/