Software Engineering Methods and Tools

Our team has adopted an Agile software development methodology for the project. Agile methodologies promote iterative development, collaboration, and flexibility, which are crucial for delivering high-quality software in dynamic environments. Within Agile, we specifically followed the Scrum framework, organising our work into short iterations - sprints. This allowed us to continuously deliver incremental improvements and respond quickly to changes in requirements. Before every sprint we scheduled a team meeting to discuss the progress of the project and suggested changes that were required.

The Agile approach aligns well with the nature of our project, which involves developing a dynamic software solution with evolving requirements. By breaking down the project into smaller, manageable chunks, we could mitigate risks associated with uncertainties in requirements and technology. Additionally, regular feedback loops during sprint reviews ensured that our product met the stakeholder's expectations and remained aligned with the project goals.

To support our Agile development process, we used several collaboration and development tools:

1. Google Docs: We use Google Docs for collaborative documentation, such as creating and sharing project plans, meeting notes, and requirements documents. Its real-time editing and commenting features facilitate seamless collaboration and version control among team members.

2. Discord: Discord serves as our primary communication platform. It promotes real-time collaboration and fosters a sense of community among team members.

3. Git/GitHub: For version control and collaborative coding, we leverage Git repositories hosted on GitHub. This enables concurrent development, branching, and merging of code changes, ensuring code integrity and facilitating code reviews.

Google Docs and Discord align well with our Agile methodology by providing features that support collaboration, transparency, and iterative development. Google Docs enables seamless document collaboration and version control, while Discord facilitates real-time communication and fosters team cohesion. Git/GitHub ensures version control and collaborative coding, enhancing our development process.

During our tool selection process, we considered alternatives such as Microsoft Word for documentation and Slack for communication. However, we opted for Google Docs and Discord due to their ease of use, real-time collaboration features, and integration capabilities, which collectively enhance our team's productivity and collaboration efficiency.

Team Organisation

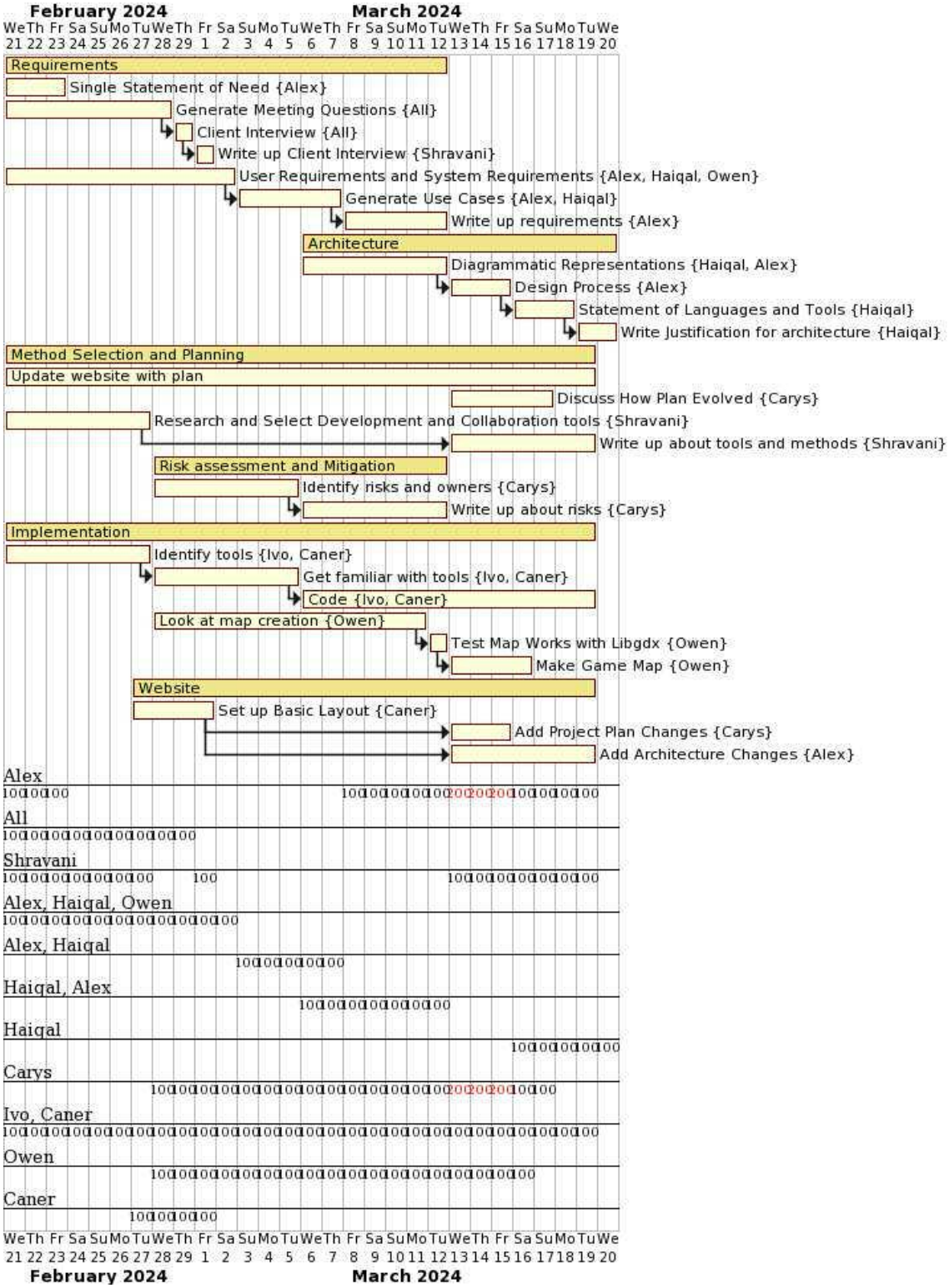
Our team followed a cross-functional organisational structure, where members with different skills and expertise collaborated closely to achieve the project objectives. Each team member was assigned specific roles and responsibilities based on their strengths and competencies, ensuring a well-rounded skill set within the team.

The chosen cross-functional approach is appropriate for both the team and the project due to the following reasons:

1. Efficient Resource Utilisation: By assembling a cross-functional team, we can leverage the collective expertise of individuals from various disciplines, optimising resource utilisation and fostering innovation.
2. Adaptability: In a dynamic project environment, a cross-functional team is better equipped to adapt to changing requirements and challenges. Diverse perspectives enable comprehensive problem-solving and decision-making, enhancing the team's agility.
3. Collaborative Culture: Encouraging collaboration among team members with different skill sets promotes knowledge sharing, skill development, and mutual support. This collaborative culture fosters a sense of ownership and accountability, driving the team towards common goals.

Systematic Plan

This is the final version of the project plan:



The plan changed a lot throughout the project as can be seen on the website. Initially most of the timelines we set were very unrealistic, as we thought certain tasks would take a lot less time than they actually did. For example a week was set for both Architecture and Risk Management, however both of these tasks needed about two weeks to complete. A second thing that changed was the amount of subtasks needed for tasks, while this stayed the same in most cases, the implementation ended up being split into more tasks to allow more people to work on it. Finally the start dates ended up moving around a lot, for example the website was meant to be worked on right from the beginning of the project however, as no one was assigned to the task for the first week it did not end up starting until the second week. The previous versions of the gantt chart can be found on the website: https://ivohadley1.github.io/projectplan.html