Cohort 3 - Group 22: The Wafflers

Eng1 Assessment 2

Method and Planning

Team Members:

Tunahan Sisman
Yousef Omar
Merrill Davis
Tom comrie
Rohan Sandhu
Cameron Pounder
Daniel Redshaw

4a) It was decided that an agile software engineering methodology would fit best for the project, specifically Scrum was chosen. Scrum allows for continuous, iterative development of our game, making it the ideal tool to meet the requirements of our project.

At the start of the process, a product backlog is created, clearly outlining the requirements and features the product needs to include and achieve. In our case this was our requirements elicitation process. Then from this we created a sprint backlog, choosing specific requirements to be met within each sprint. Usually a sprint lasts 2-4 weeks, but due to our restricted time to complete the project our sprints were shorter, each resulting in a new iteration to the game.

During sprints, bi-weekly meetings were held to share progress and highlight any issues that the team have encountered. This ensured that the team was organised and deadlines were met. In addition, it helped with risk management; problems that were raised within the meetings were continually tracked with the risk assessment table, allowing for measures to be taken to prevent or mitigate them. After each sprint we gathered feedback, creating a new iteration of our game with each cycle. This iterative approach was necessary for the development of our game as it enables the incorporation of new requirements into future iterations.

For collaborating on the documentation side of the project, Google services were used due to its collaboration features. A shared Google Drive was created for easy file sharing and access, allowing us to keep track of documents and easily examine each other's work as necessary. Google software also has version control. Although not as crucial to the documentation aspects of the project as the programming aspects, it made tracking changes, and the team members responsible for these changes, much easier. If mistakes were made, each document could be rolled back to an earlier version.

We considered using Notion, a software specifically designed to aid in managing projects and working collaboratively in a team. It facilitates managing deadlines and creating documents, making it ideal for our project. Given that Notion had some interesting and helpful features, we decided to use it. Due to the restricted timeframe we had for the project, we opted to stick with software that neatly integrates with the web-based workflow of the project.

With regards to the programming side of things, GitHub was used, a widely used version control system in collaborative software development. GitHub was the obvious choice, as those allocated to the programming of the game have prior experience with using it and it is widely compatible. Other version control platforms were considered, such as GitBucket. Given the key element of collaboration in the project, the cumbersome nature of GitBucket’s self-hosting would prove an unnecessary endeavour to learn. Ease of use was prioritised to save time. Creating a GitHub repository allowed the team to individually work on separate parts of the game in isolation from one another through the use of branches. GitHub is successful in outlining issues when merging changes made onto the main branch. One rule the team followed was that code must be authorised by additional members of the team before the code can be pushed to the main branch. This stops any accidental merges that may break the system, especially during real time collaboration.

The chosen IDE for development was IntelliJ. One of the core project requirements was for the game to be made in Java. IntelliJ has code review and “code with me” collaboration capability, aswell as built in GitHub compatibility for efficient version control. These features were significant factors in the selection of IntelliJ, providing a strong foundation for programming. Simultaneous coding and remote collaboration would be made easy by this. IntelliJ was also made more favourable by the free educational access provided by JetBrains. These tools are widely used in industry. VSCode and Eclipse were also considered as IDEs, but our game library of choice- LibGDX- has poor compatibility with them.

Communication within the team was of paramount import. A backup whatsapp chat for general conversations was created and then moved the majority of our project discussions to a Discord server. Discord can be very useful with collaborative work. It allows for voice and video calls with screen sharing, meaning meetings can be held virtually. Within the server separate channels were created, for example we had a separate channel for implementation, general messaging, code reviews and assets. This helped us keep the work organised and prevented confusion. Although this separate channel was created for the programmers, everyone could still view it ensuring all group members were up to date with the work that was being completed. There were other options available to us, for example Google Meet or Microsoft Teams, however Discord had all the facilities of these softwares and more, all in one place, hence this is the communication medium we chose.

4b) Team organisation was crucial in this project due to the large number of different tasks needing to be completed. Dividing the work between team members was obviously necessary to ensure each member completed an equal amount of work and to ensure the project was completed on time.

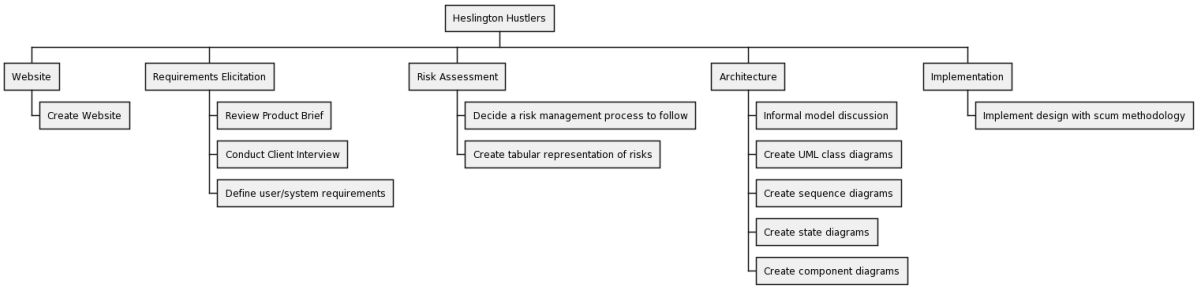
After the initial requirements interview, roles were assigned to team members based on preferences and personal skills. This helped prevent confusion and ensure work was completed to the necessary standard whilst meeting deadlines. These roles separated tasks revolving around project management, taking meeting minutes, programming, UI design, testing, and documentation. This provision of team roles ensured that all components of the game were up to standard. The struggle once this had been done was bringing the individual work back together and ensuring everything made sense as a whole. A meeting was held a few days before submission where we discussed all the work completed as a group and made any changes necessary for submission. To ensure everyone had a clear idea of what they were doing, we assigned people to different parts of the project in our first meeting after the interview. Doing this prevented any confusion down the line regarding what parts of the project were who’s responsibility. It also meant that individuals did not need to research different parts of the project in depth and could mainly focus on their own section and getting it to a high standard. Although everyone had their own roles and responsibilities, everyone in the team still needed to be kept up to date with what was going on in different parts of the project.

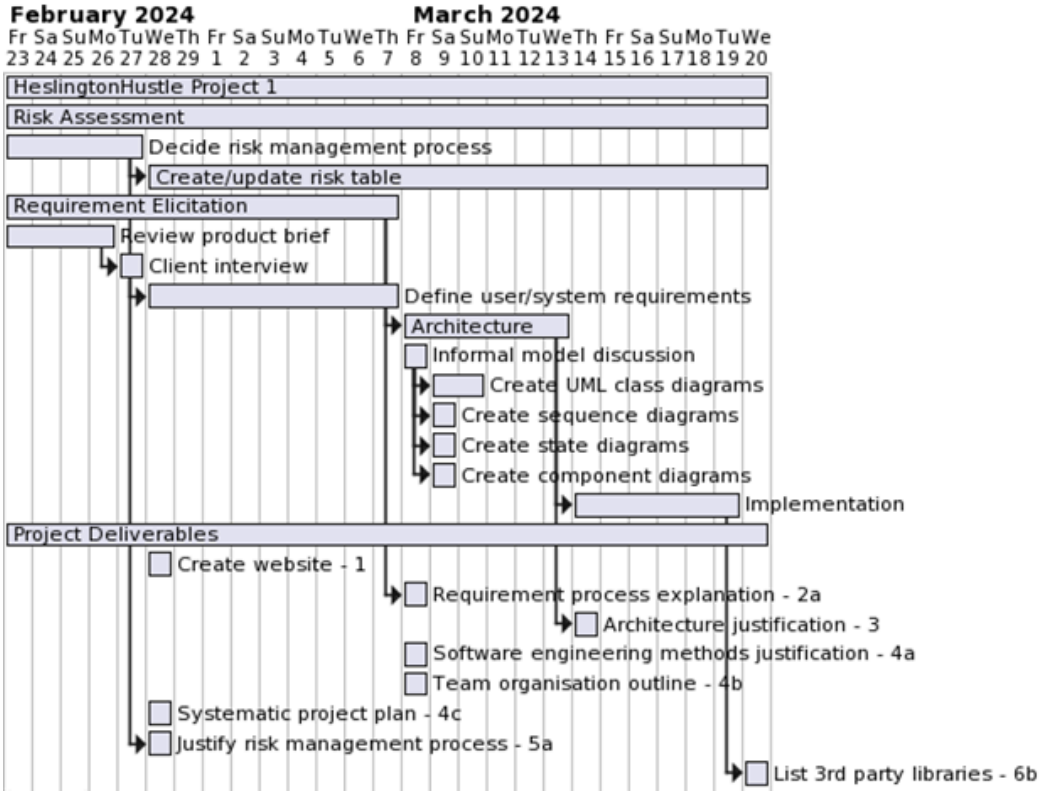
The meetings held were used to discuss such matters. In each meeting, team members would share progress updates, solve problems that may have arisen during assigned tasks and problems that may rise in future, and plan the next tasks going forward. As a group, relevant risks were discussed with their mitigation strategies, referring back to the risk

register. It was decided to hold the meetings in this way to ensure we were sticking to the Scrum development methodology. By sticking to the Scrum organisational plan, it was ensured that work that needed to be completed was kept track of and kept on schedule with sprints throughout the project. Sprint planning consisted of reviewing the product backlog and tasks.

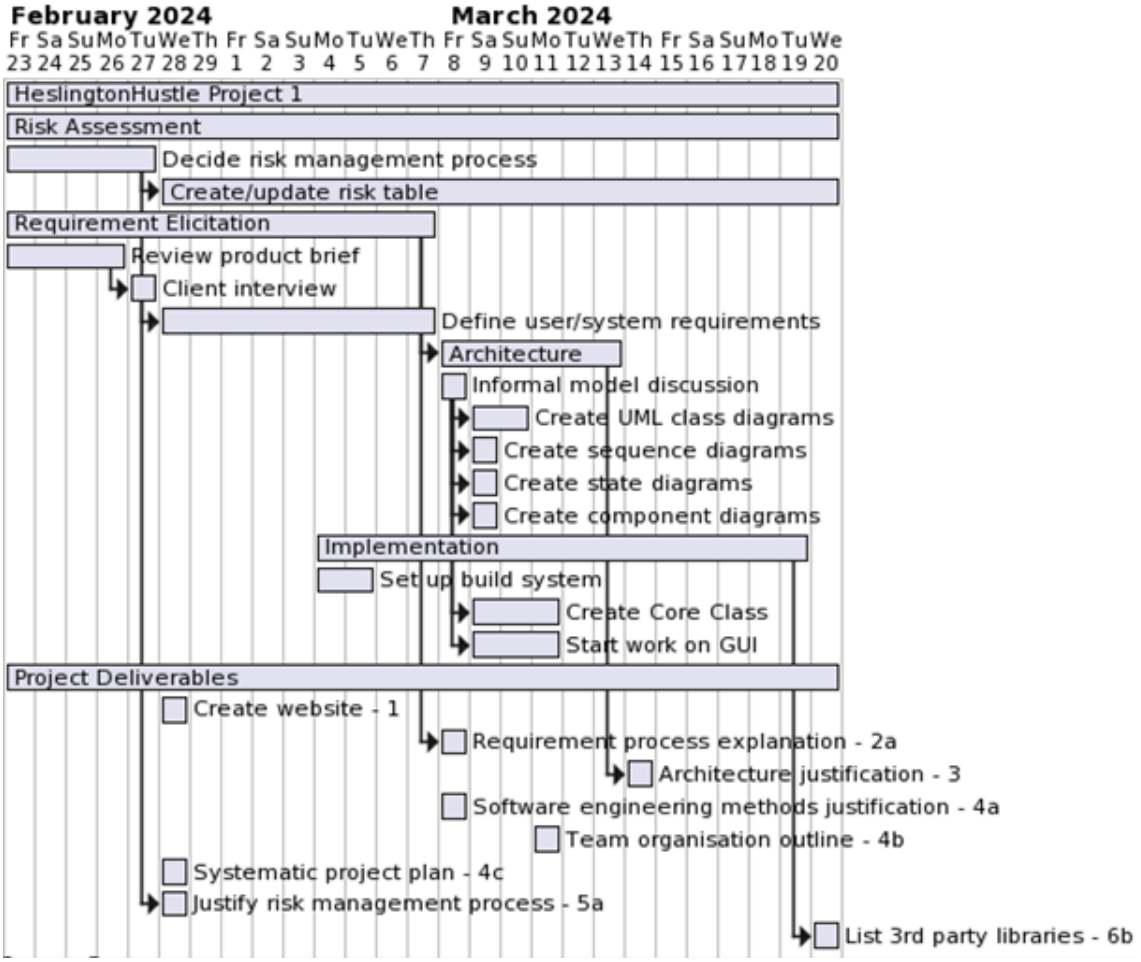
Before the project submission deadline, a meeting was held to review work completed and deliverables. Ensuring cohesion between work submitted by different team members was a priority, and was a big form of quality control undertaken. Maintaining a structured approach to planning, communicating, and completing assignments allowed us to stay on schedule and adapt to the nature of the project and its requirements.

Work Breakdown:

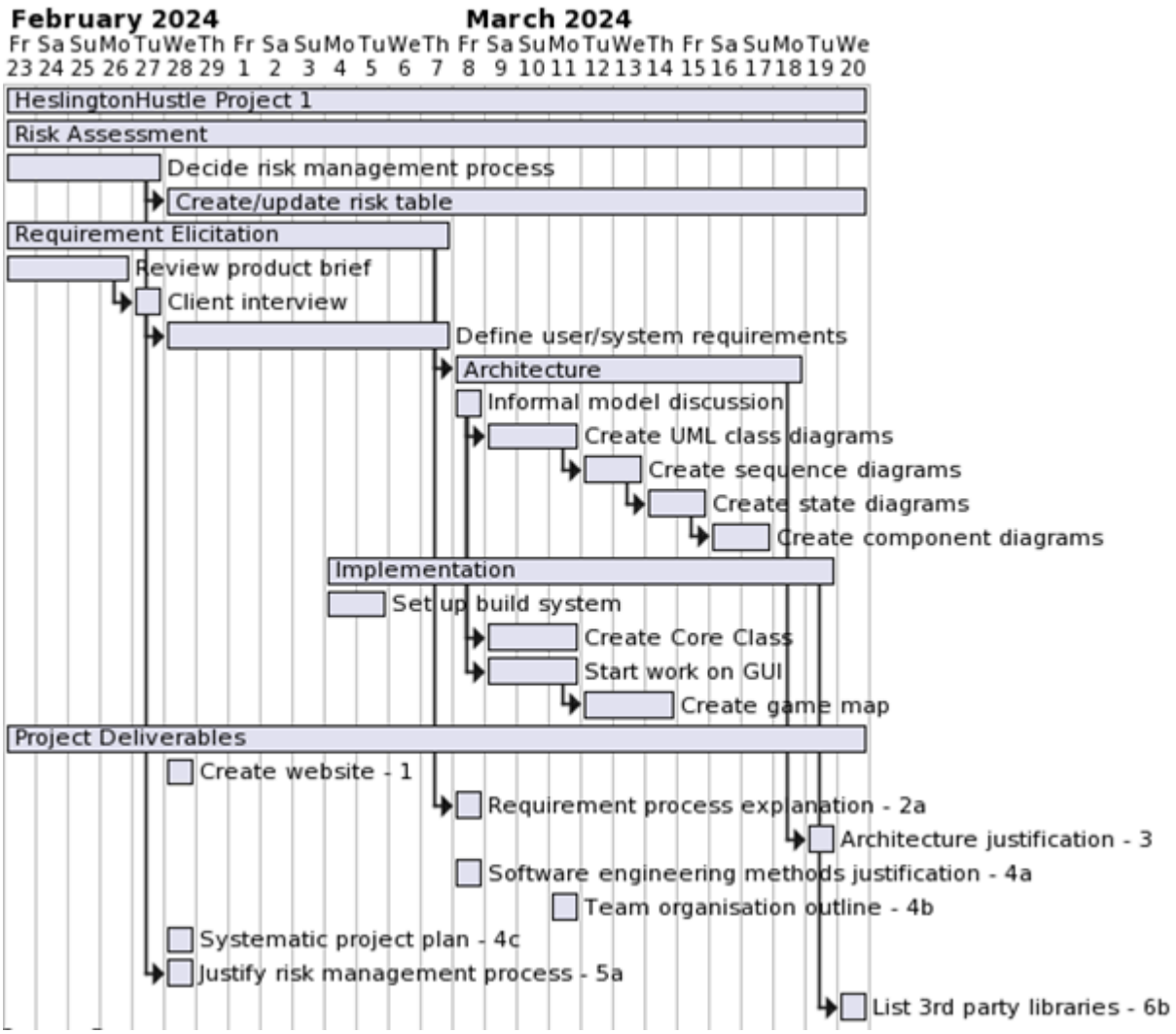




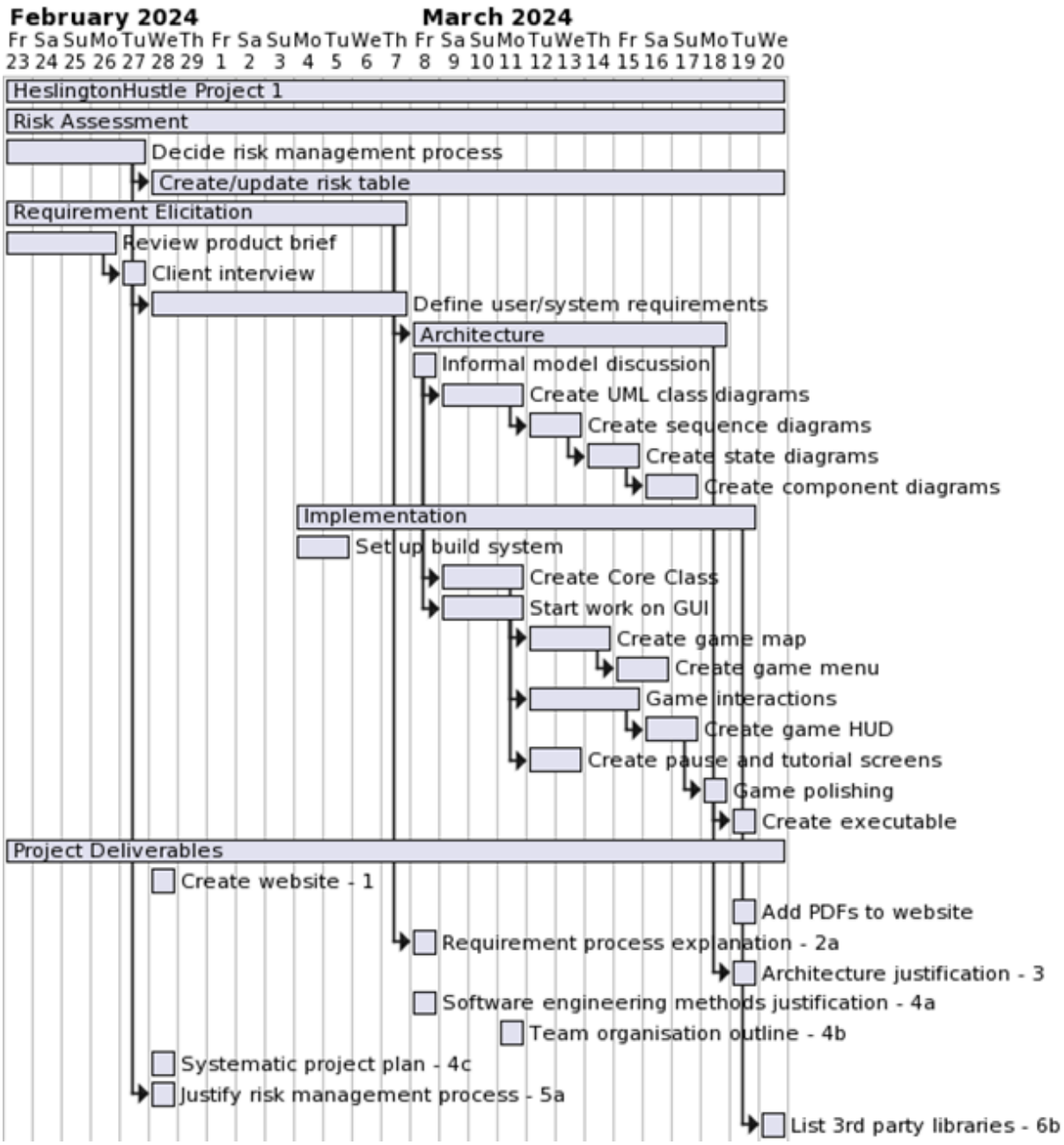
At the start of the project (week 2 and 3) we created a rough plan outlining when we wanted each section to be completed by. At this stage we knew exactly when the requirements elicitation and risk management would need to be done by. As you can see, the Risk Assessment spans across the entire project. This is because as we worked on the project, there were likely to be new risks discovered that would need to be added to our risk management tableau. Requirements elicitation would take until our practical session at the end of week 4. We planned to spend more time on this section as it is arguably the most important. Without well defined requirements the project would suffer.



Our week 4 plan shows updates to the Implementation section. Initially this section came after Architecture, however we realised that these two could be happening in parallel. After our Informal model discussion, both the architecture and implementation team knew the structure of the game so could both work on it at the same time. This was an important change to make, as without it the quality of our game would likely have suffered, due to insufficient time for implementation.



By week 5 we realised that we had not assigned enough time to the architecture section of the project. This is something we had outlined as a risk of the project at the start, and it can be found in the Risk Assessment. Although we had initially planned to only spend just under a week on Architecture, we left plenty of time to complete it after the initial assigned time, allowing us to extend the architecture section of the plan to the start of week 6, giving the architecture team ample time to create all the necessary diagrams. At this point implementation was also updated to include the next steps in the programming.



In the final week of the project, we see the complete plan. Implementation has been updated to plan the programming of the game to completion. Using our plans we were able to stay on track with the project deadline, finishing a day early, ready to submit

Heslington Hustle					
Website	**Change Report**	**Implementation**	**Testing**	**User Evaluation**	**Continuous Integration**
- Update website with updated deliverable	- Update the change report as we go	- Review the new client requirements	- Determine tests that can be automatic	- Create ethics and consent forms	- Set up continuos integration in GitHub
- Update website with new deliverable	- Update Requirements	- Implement the new requirements	- Set up test environment	- Create questionaire for users to fill in	- Add support for different OSs
	- Update Architecture	- Implement any further identified improvements	- Create tests to be run on code	- Analyse the final data	- Do Continuous Integration write up
	- Update Method and Planning	- Update the Implementation document	- Do Testing write up	- Do User Evaluation write up	
	- Update Risk Assessment and Mitigation				

For the second stage of the project, a new work breakdown was made to abstract the tasks into smaller, more manageable tasks to give the team a better understanding of what each new piece of documentation would require.

Along with this new Gantt charts were created, providing a simple and easy to follow plan for how the project was to progress. From week 7 to 9, the below Gantt chart was followed:

	Date						
Task	11/04/2024	18/04/2024	25/04/2024	02/05/2024	09/05/2024	16/05/2024	23/05/2024
1. Heslington Hustle Project 2							
2. Choose Project to takeover							
3. Change Report							
3.1 Change report write up							
3.2 Updating Risk Assessment and Mitigation Documentation							
3.3 Updating Requirements Documentation							
3.4 Updating Method Selection and Planning Documentation							
3.5 Updating Architecture Documentation							
4. Implementation							
4.1 Read through and understand code base							
4.2 Implement new requirements							
4.3 Final UI updates and bug fixes							
4.4 Update Implementation Documentation with new 3rd party assets							
5. Testing							
5.1 Set up testing environment							
5.2 Writing tests to satisfy requirements							
5.3 Refactor code to help run tests							
5.4 Testing write up							
6. User Evaluation							
6.1 Creating the consent, response and ethics forms							
6.2 Performing user evaluation on other Computer Science							

students							
6.3 Analysing the data							
6.4 User evaluation write up							
7. Continuous Integration							
7.1 Set up workflow with automated build and test coverage							
7.2 Add artifacts output							
7.3 Add automatic release by tags							
7.4 Add multi-OS support							
7.5 Continuous Integration write up							

After week 9, not all the new features had been implemented, so user evaluation and continuous integration were pushed back, with the intention of splitting the team into two so half can work on each part of the project, displayed in the Gantt chart below. From week 10 until the end of the project, week 13, the following schedule was kept to:

Task	11/04/2024	18/04/2024	25/04/2024	02/05/2024	09/05/2024	16/05/2024	23/05/2024
1. Heslington Hustle Project 2							
2. Choose Project to takeover							
3. Change Report							
3.1 Change report write up							
3.2 Updating Risk Assessment and Mitigation Documentation							
3.3 Updating Requirements Documentation							
3.4 Updating Method Selection and Planning Documentation							
3.5 Updating Architecture Documentation							
4. Implementation							
4.1 Read through and understand code base							
4.2 Implement new requirements							
4.3 Final UI updates and bug fixes							
4.4 Update Implementation Documentation with new 3rd party assets							
5. Testing							
5.1 Set up testing environment							
5.2 Writing tests to satisfy requirements							
5.3 Refactor code to help run tests							
5.4 Testing write up							
6. User Evaluation							
6.1 Creating the consent, response and ethics forms							
6.2 Performing user evaluation on other Computer Science students							

6.3 Analysing the data							
6.4 User evaluation write up							
7. Continuous Integration							
7.1 Setup workflow with automated build and test coverage							
7.2 Add artefacts output							
7.3 Add automatic release by tags							
7.4 Add multi-OS support							
7.5 Continuous Integration write up							