UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

Method Selection and Planning
Cohort 2 - Group 18 - Octodecimal

Group Members

Izz Abd Aziz

Phil Suwanpimolkul

Tom Loomes

Owen Kilpatrick

Zachary Vickers

Michael Ballantyne

Outline and Justification of our Software Engineering Methods

Due to the short time frame to complete the project, it was clear that multiple stages of the software engineering process would have to be completed at the same time. This fitted an agile approach. It was also reasonable to suspect that plans would change significantly each week as it would be hard to plan how long each task would take when all team members had other commitments and relatively little experience of creating a game or using game engines. Agile approaches prefer shorter timeframes, encourage face-to-face conversations and encourage regular reflection on efficacy [1]. These fitted well to our organisation as we had one to two face-to-face meetings with all team members each week. As these meetings were 2 hours long, they provided ample time to reflect on the previous week, plan for the following week and have detailed discussions about the deliverables.

The main inspiration for our methods and organisation came from scrum. As we were already committed to weekly meetings, it was natural to create weekly sprints. The start of each meeting was then dedicated to a review of the previous sprint to reflect on which tasks got completed, and if any extra tasks were created during it. We would then plan the next week (sprint) with the tasks that needed doing, and assigned team members to each task. The retrospective also allowed for reflection on unforeseen dependencies that had limited progress which allowed for the re-ordering of tasks and prioritisation of what was left. We formalised this all our meeting logbook, which can be found on our team website.

We also drew inspiration from the spiral lifecycle as this produces good documentation control [2] and a large part of the project is based around documentation. This documentation is also reviewed in every loop which was very similar to how we created new plans and reviewed the risk assessment each week. However, we only used certain parts of this lifecycle as adapting it well would not have fitted our size of project and scheduling was extremely important to the project [2].

Identification and Justification of Development and Collaboration Tools Used

The customer briefing placed the constraint of using only Java for the implementation and using a gaming engine written in Java. The requirements for the game included that it would be 2D. Research of various 2D Java game engines was undertaken and LibGDX was chosen as this has specifically been built for 2D games and interlinks well with Git. Other alternatives, including J Monkey and LWJGL, were considered. Many team members liked the look of J Monkey however it seemed much more suited to 3D game development and the assessment brief specified that the game must be 2D. By looking at reviews of LWJGL, it seemed that it was difficult to learn at first so it was felt that this would be an unnecessary delay and inefficient to choose [3]. IntelliJ was selected as the IDE as LibGDX relies heavily on Gradle to assemble projects. Originally, the plan was to use VSCode as all team members had used it before including for Java and it was already on all department machines but this does not interact well with Gradle and would have created unnecessary difficulty during implementation. After finding that IntelliJ was also available on the department machines and would work much better, its use was agreed upon.

Github was chosen for the code base and website as several team members had prior experience and it is the standard tool. Other alternatives were briefly considered such as Apache Subversion but no team members had prior experience with these so it was felt that they would add unnecessary delays and extra work of becoming experienced with using them first. In addition, Github encourages small commits as well as branching and merging which worked well with our agile approach. Github was also selected to host the website as all team members either had prior experience or would be gaining experience through its use in the implementation. Google Drive was used for collaboration for the other deliverables due to the live collaboration features and all team members having access and prior experience. Using Google Docs for the deliverables also had the advantage of version control so any changes could always be reverted if necessary. Being able to add comments easily to

Google Docs also allowed collaboration on documents without having to switch between multiple platforms.

For collaboration such as suggesting ideas outside of meetings, Discord was agreed as the platform to use. Slack was considered but no team members had experience of using it whereas all had experience of Discord. Whatsapp was also considered but Discord was felt to be more suitable as it was better suited for sharing larger amounts of text and channels would allow different deliverables to be discussed in their own areas to help organisation. Using Discord fit well with our scrum-inspired methods as it allowed for very frequent communication between team members. For architectural diagrams, the decision was made to use PlantUML. This was because this works well with Google Docs and so it would be easy to add diagrams to documents but also to change them during the evolution of the document and the project. PlantUML was also used for other diagrams including those in the planning process. This reduced the bus factor as it meant that more people were aware of and had experience with the language being used for the architectural diagrams. Alternatives considered included Mermaid and Graphviz but these did not have the benefits of PlantUML.

As well as the constraint of only using Java for implementation, there was also the constraint of only using tools available on department machines so that if team members weren't able to use a personal device or access the tools personally, they would still be able to access the full project and contribute well. Available expertise was taken into account for each decision.

Team Organisation

Team members were assigned to roughly 15 marks from the 6 deliverables. Most team members wanted to be on more than one deliverable to gain more experience. As the website had so few marks allocated, this was assigned to just one person. However, all other deliverables had a minimum of 2 team members working on them to lower the bus factor. Task assignment started by assigning team members to areas that they had experience of in order to keep the project as efficient as possible. Team members were then assigned to anything they particularly wished to do and finally the gaps were filled.

During our first weekly meeting, we focused on reading through the whole brief and breaking down the entire assessment into a comprehensive list of tasks, as a work breakdown diagram. This can be seen on the next page. As opposed to assigning leaders for each deliverable, we instead assigned team members to each task based on their preferences and skills. We attempted to distribute tasks evenly, to ensure everyone had equal participation in the project, but this was reviewed in the weekly meetings to ensure no one was being overworked or underworked. More generally however, based on the task distribution, we did end up with around 2 primary members on each deliverable, which can be seen in the deliverables table on the following page.

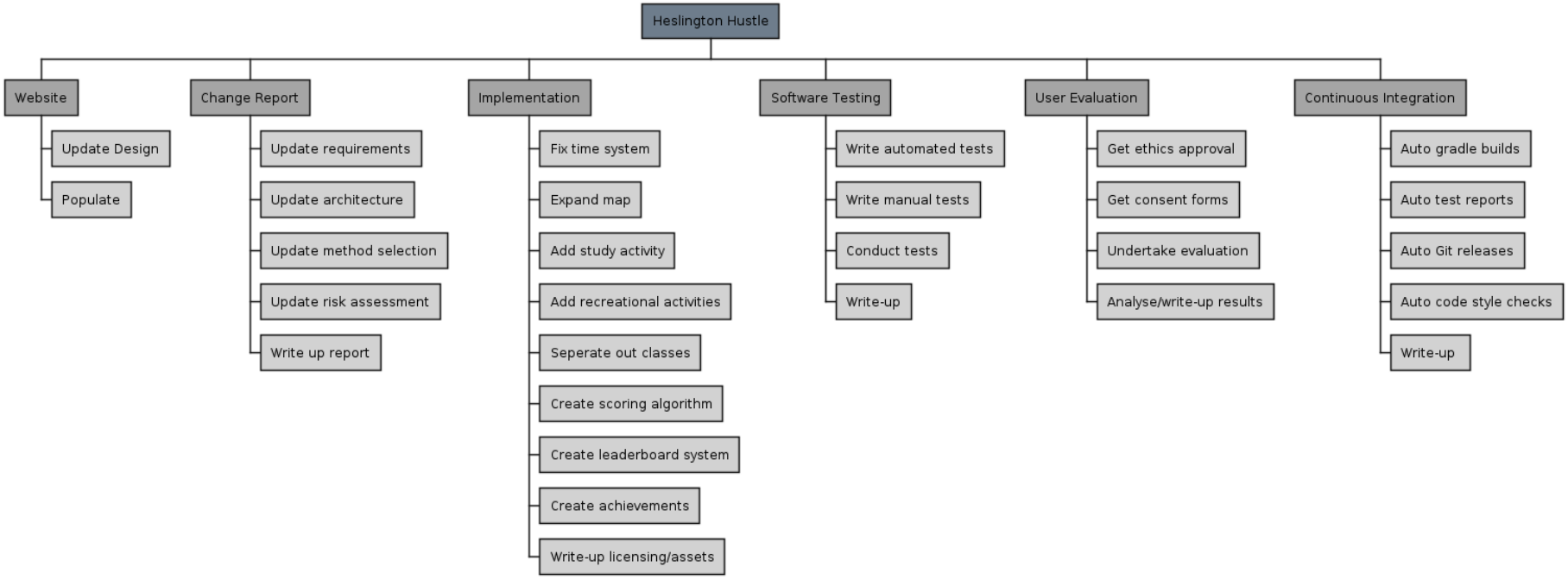
The risk management process required each risk to be assigned to a team member. We gave these risks to the team members who were on relevant tasks relating to them, such as documentation or implementation. We also assigned a team leader, Zachary, who naturally took on an administrative role in team meetings, and he was given risks relating to team organisation.

There was also the role of upper management provided by the customer. This was available if it was needed to solve team disputes or any other issues but wasn't needed. The customer was also the main stakeholder and the only stakeholder who decided requirements. Communication with the stakeholder was first through a formal client meeting to gather more information about requirements. Through assessment 2, we did not hold any further meetings, but instead asked questions over eMail or during practical sessions when clarification was needed.

Decisions were mostly made through unanimous decision as there was very little disagreement. However, where there was any disagreement, the decision was first attempted to be made through the majority opinion. If opinion was equally split, the decision was made by the team leader.

Work Breakdown

The work breakdown structure was created using the assessment document to split into deliverables which were then further broken down. The product brief was used to break down the implementation deliverable



Deliverables table

ID	Title	Due date	Description	Visibility	Relevant tasks	Primary Members
D1	url2.txt	23/5	Website	Shared	T1	Phil
D2.1	Change2.pdf	23/5	Requirements	Shared	T2.5	Tom, Izz
D2.2	Updated Assess 1 deliverables	23/5		Internal	T2.1 - T2.4	Tom, Izz, Michael, Zack
D3.1	Impl2.pdf	23/5	Implementation	Shared	T3.8	Tom
D3.2	Code	23/5	Implementation	Shared	T3.1 - T3.7	Phil, Zack
D3.3	Executable JAR	23/5	Implementation	Shared	T3.1 - T3.7	
D4	Test2.pdf	23/5	Software testing	Shared	T4	Owen, Michael
D5	Eval2.pdf	23/5	User evaluation	Shared	T5	Tom, Izz
D6	CI2.pdf	23/5	Continuous Integration	Shared	T6	Owen

Tasks Table

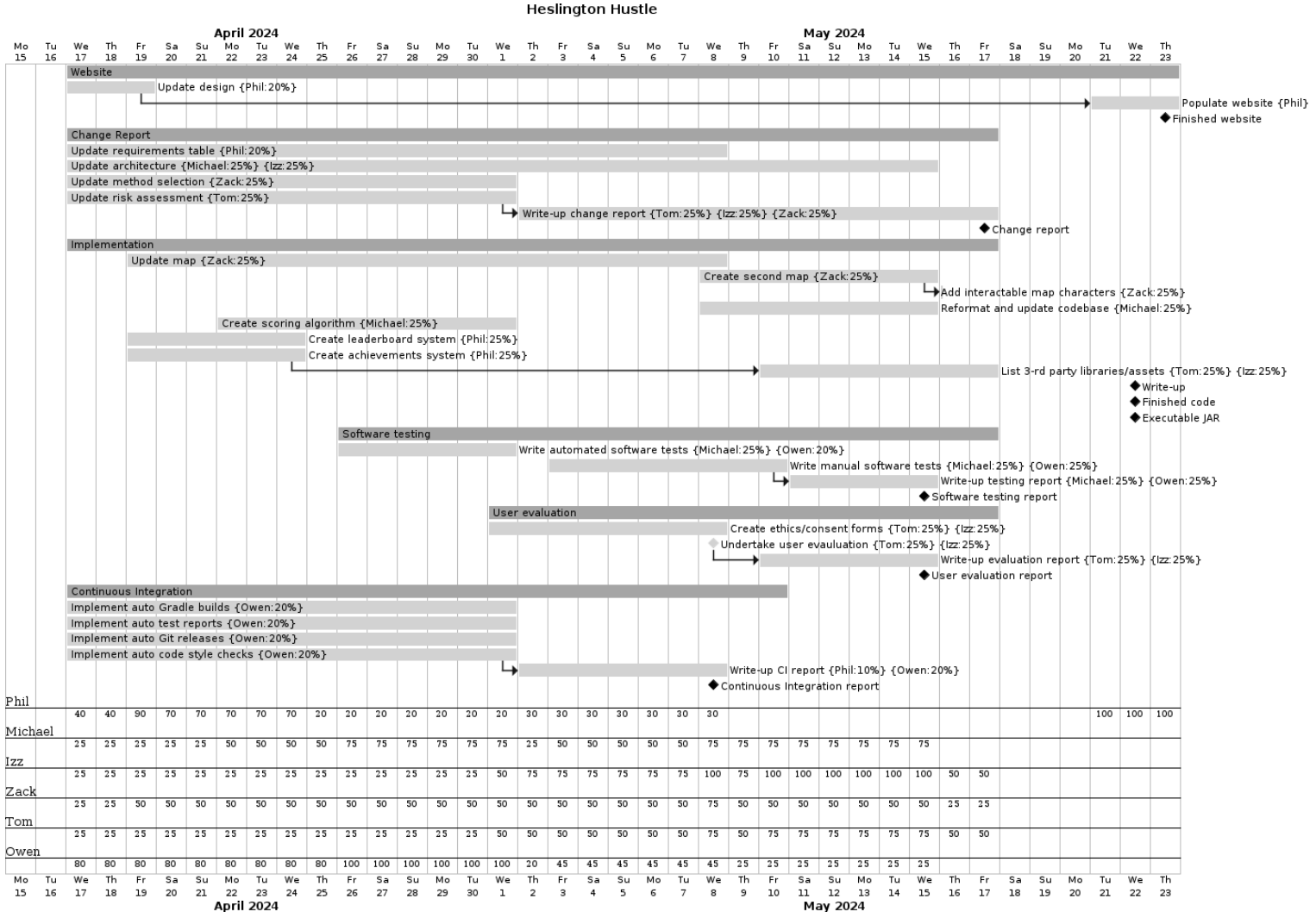
Task ID	Description	Start date	End date	Dependencies	Priority
T1.1	Update website design	17/4	19/4		Medium
T1.2	Populate website	21/5	23/5	T1.1	High
T2.1	Update requirements table	17/4	23/5		High
T2.2	Update architecture	17/4	23/5		High
T2.3	Update method selection	17/4	26/4		High
T2.4	Update risk assessment	17/4	26/4		High
T2.5	Write-up change report	26/4	17/5	T2.1, T2.2, T2.3, T2.4	High
T3.1	Reformat and update codebase	19/4	26/4		High
T3.2	Increase size/complexity of map	19/4	1/5		High
T3.3	Create new secondary map	19/4	1/5		Medium
T3.4	Create scoring algorithm	22/4	24/4		High
T3.5	Create leaderboard system	24/4	1/5	T3.4	High
T3.6	Create achievements system	24/4	1/5	T3.4	High
T3.7	Add interactable map characters	1/5	10/5	T3.2, T3.3	Medium
T3.8	List 3rd-party libraries and assets with licences and discussion of licence suitabilities	10/5	17/5	T3.2, T3.3, T3.7	High
T4.1	Write automated software tests	1/5	10/5		High
T4.2	Write manual software tests to fill in gaps	1/5	10/5	T4.1	High
T4.3	Write-up software testing report	10/5	17/5	T4.1, T4.2	High
T5.1	Create ethics / consent forms	26/4	1/5		High
T5.2	Undertake user evaluation	1/5	1/5	T5.1	High
T5.3	Write-up evaluation report	1/5	17/5	T5.2	High
T6.1	Implement auto gradle builds	17/4	24/4		High
T6.2	Implement auto test reports	17/4	24/4		High
T6.3	Implement auto Git releases after pushes	17/4	24/4		High
T6.4	Implement auto code style checks	17/4	24/4		High
T6.5	Write-up continuous integration report	26/4	10/5	T6.1, T6.2, T6.3, T6.4	High

Discussion of Plan Evolution

The Gantt chart was updated after each weekly meeting to reflect changes in the plan and variations between the work that was intended to be completed and what was actually completed. The snapshots of this can be found on our website. As we'd adapted a scrum methodology, each meeting started with a sprint review and retrospective to identify what work had been completed and any issues that team had faced. A new plan was agreed for the remainder of the project. Small changes were required each week. These mostly involved extending the number of days required for tasks or pushing back tasks due to unforeseen dependencies.

The final iteration of our gantt chart can be found on the following page
All of the previous charts can be found here.

Gantt Chart


Heslington Hustle
April 2024
May 2024
Website
Update design {Phil:20%}
Populate website {Phil}
Finished website
Change Report
Update requirements table {Phil:20%}
Update architecture {Michael:25%} {Izz:25%}
Update method selection {Zack:25%}
Update risk assessment {Tom:25%}
Write-up change report {Tom:25%} {Izz:25%} {Zack:25%}
Change report
Implementation
Update map {Zack:25%}
Create second map {Zack:25%}
Add interactable map characters {Zack:25%}
Reformat and update codebase {Michael:25%}
Create scoring algorithm {Michael:25%}
Create leaderboard system {Phil:25%}
Create achievements system {Phil:25%}
List 3-rd party libraries/assets {Tom:25%} {Izz:25%}
Write-up
Finished code
Executable JAR
Software testing
Write automated software tests {Michael:25%} {Owen:20%}
Write manual software tests {Michael:25%} {Owen:25%}
Write-up testing report {Michael:25%} {Owen:25%}
Software testing report
User evaluation
Create ethics/consent forms {Tom:25%} {Izz:25%}
Undertake user evauluation {Tom:25%} {Izz:25%}
Write-up evaluation report {Tom:25%} {Izz:25%}
User evaluation report
Continuous Integration
Implement auto Gradle builds {Owen:20%}
Implement auto test reports {Owen:20%}
Implement auto Git releases {Owen:20%}
Implement auto code style checks {Owen:20%}
Write-up CI report {Phil:10%} {Owen:20%}
Continuous Integration report
Phil
Michael
Izz
Zack
Tom
Owen

References

[1] K. Beck, et al. (2001). Principles behind the Agile Manifesto. Manifesto for Agile Software Development. [Online]. Available: https://agilemanifesto.org/principles.html [Accessed: 13 March 2024].
[2] A. Garg, R. K. Kaliyar, and A. Goswami (2022). PDRSD-A systematic review on plan-driven SDLC models for software development. 8th International Conference on Advanced Computing and Communication Systems, Coimbatore, India, Mar. 25-26, 2022, IEEE, 2022
[3] B. Refi (2023, Aug. 3) Java Game Engines: Top Choices For Game Development. Bluebird. [Online]. Available at: https://bluebirdinternational.com/java-game-engines/ [Accessed: 14 February 2024].