UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

ENG1 Assessment 1
Group 18 - Octodecimal

Method Selection and Planning

Group Members:

Izz Abd Aziz

Phil Suwanpimolkul

Tom Loomes

Owen Kilpatrick

Zachary Vickers

Michael Ballantyne

Software Engineering Methods

Our group applied the agile software engineering method Scrum. We used an agile method because this project has volatile requirements and short iteration cycles. Agile methods are also best suited to small teams like our group. Specifically, we practised Scrum by carrying out “scrums” at the start of each group meeting, where each team member reported their progress since the last meeting, and their current plans for future progress. This helped the group stay on track with its goals and deadlines.

Scrum also involved dividing development into short “sprints” of work. We reviewed the previous one-week sprint and planned our next sprint every Friday meeting. Working in small sprints allowed us to keep on track with our goals and review our work output at regular intervals. We utilised Scrum over XP, the alternative agile method considered, as XP is primarily focused on code and programming technical practices. This project was focused on the entire engineering process rather than solely programming, hence Scrum fit with our project better.

To develop text-based artefacts our group used the developmental tool *Google Docs*. This was preferred over alternatives considered (such as *Microsoft Word*) due to the group’s familiarity with *Docs*, its ease of use and real-time collaboration features. We used *Google Drive* as a collaboration tool, as this was more convenient than alternative cloud options considered such as *OneDrive* due to its synergetic nature with *Google Docs*. Having a shared drive and the collaborative features of *Docs* fit well with Scrum as it provided an easy location for us to share documents, which was useful when reviewing the previous sprint.

For the development of architectural and organisational diagrams (such as class diagrams and Gantt charts) we used *PlantUML*. This fits our agile method Scrum, as one of the characteristics of agile processes is sufficient and accurate documentation. *PlantUML* allowed us to efficiently create clear, high-quality documentation diagrams. Alternatives we considered included popular drawing tools like *Drawio* and *Lucidchart*, but we chose *PlantUML* for its simplicity and efficiency. Furthermore, the *Google Docs Chrome Extension* for *PlantUML* provided synergy as we could directly create and edit charts within the documents we stored them in.

For our implementation, we used the collaborative version control system *Github*. We chose to use a decentralised VCS over the centralised alternative considered (*Apache Subversion*), as this allowed us to make commits and reversions to our local repositories when we didn’t have access to the central repository. We chose *GitHub* as our decentralised VCS due to its popularity and its encouragement of branching, which we used frequently throughout our project development. To interact with our repository we used *GitHub Desktop*; we chose this over the considered alternative *SourceTree* due both to its simplicity and it being maintained by the same company as *GitHub*. These choices fit well with Scrum as we could work individually during sprints regardless of WiFi access on local repositories. We also could branch and merge whenever needed, allowing individual work to be combined easily before the end of a sprint.

In development, every group member used the IDE *IntelliJ*. We all used the same IDE so that it was easier to resolve issues and share tips while programming. This fit with Scrum, as we could discuss any issues we’ve had with *IntelliJ* in a scrum, and get it resolved before the next sprint. We chose this option as unlike the considered alternative *VSCode*, this IDE was purpose-built with Java development in mind. Although *VSCode* was more familiar to the group, its more general-purpose meant *IntelliJ* had more easy-to-use functionalities for Java programming.

Another key development tool used was our game engine *LibGDX*. We chose this over the alternatives considered, such as *Slick2D*, as *LibGDX* has a large and active community meaning there are ample learning resources available. Another popular game engine we looked at was *jMonkeyEngine*, however the client interview led to us learning the game must be 2D - *jMonkeyEngine* is used for 3D games, making it unsuitable.

For the game's art, we used the website *Pixilart* as our chosen development tool. As a free tool that can be accessed remotely from anywhere, with built in animation functionalities and sprite sheet export formats, it was a perfect fit for the art design of our game. We also used the map editor *Tiled* as a development tool in the project, allowing us to edit our game's tile map and implement the game's background visuals. Creating a tile map, although more difficult to implement than a fully drawn map, would lead to much easier iterative development moving forwards as it can be changed and expanded modularly.

Organisation Approach

During our team's first meeting, we discussed our strengths, weaknesses, and parts of a software engineering lifecycle we enjoyed the most. From this, we were able to immediately delegate team roles and responsibilities, which would be vital in task management going forward with the project. Zachary naturally took the role of team leader - directing team meetings, tracking project progress and delegating tasks. Phil was very comfortable with the technical aspects of the project, jumping in to set up the github repository, website, and overseeing the implementation section. Michael was also confident in the technical tasks, as well as bringing a keen creative mind that aided in architectural design and problem solving. Tom and Owen felt very comfortable completing admin tasks such as risk assessment and method planning, and Izz was very open to contribute to different tasks, helping out others where needed. Establishing our roles and strengths early on was highly effective, as task delegation became trivial and team members confidently stepped into roles that needed filling.

We arranged to hold meetings as a team twice a week - once during our allocated practical session, and once every Friday externally. In each meeting we discussed and recorded individual progress since the last time we met, worked through tasks together, and created a comprehensive set of tasks to complete for the next meeting. We kept a logbook of all our meetings, updated our project plan with our progress, and took a snapshot of this to document its evolution.

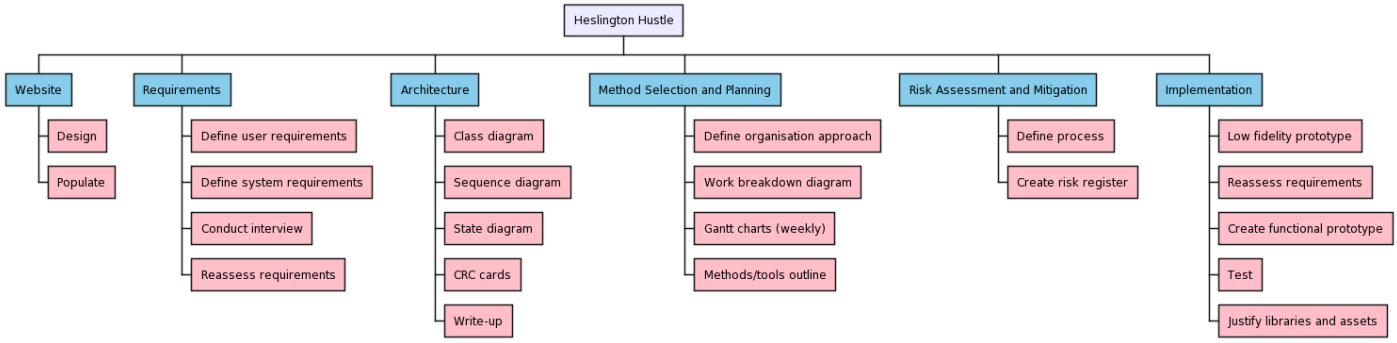
To discuss and collaborate outside of in-person meetings, we created a *Discord* server. This featured different channels relating to all deliverables, as well as a #general text channel and resource channel where all relevant documents and links were stored for easy access.

Within our approach, we were cautious to maximise our bus factor. During risk assessment, we concluded that with only one "primary" team member assigned to each task, we'd have a likely and severe risk that if a member became unwell (or had to leave the project) tasks would be delayed or left incomplete. Thus, we implemented a system where at least one other team member was assigned to each task in a "secondary" role. This person would perform regular reviews as progress was being made in the task and aid in anything that they were asked for help with. This allowed us to ensure that progress wouldn't be lost if we lost a team member, as there was always at least one other person who understood each task, and could move into the primary role if required.

Although no conflicts arose during the project, we had implemented systems in case this did happen. Firstly, the secondary role for each task ensured work quality was kept to a high standard through regular reviews. Secondly, if disagreements over the project emerged, such as conflicting creative ideas, we would hold an anonymous team vote through a *Google Form* - if this resulted in a tie, we would attempt to reach a compromise. In the case of team members being absent to meetings or not completing tasks, we agreed as a one-off occasion this would be disregarded, but multiple instances of such behaviour would result in us following up with the individual to ensure everything is okay, and contacting the module leaders if required.

Systematic Project Plan

The assessment brief contained an overwhelmingly large volume of tasks and information. Hence, in our team meeting the following day, we prioritised the creation of a comprehensive project breakdown. As the project progressed, we learned content and researched into software engineering, gaining a better understanding of how to complete tasks such as “Architecture”. This led to further iterations to the work breakdown diagram, adding necessary deliverables such as “State Diagram”. All iterations can be found in the weekly snapshots section of our website. The final iteration of this chart is shown below:

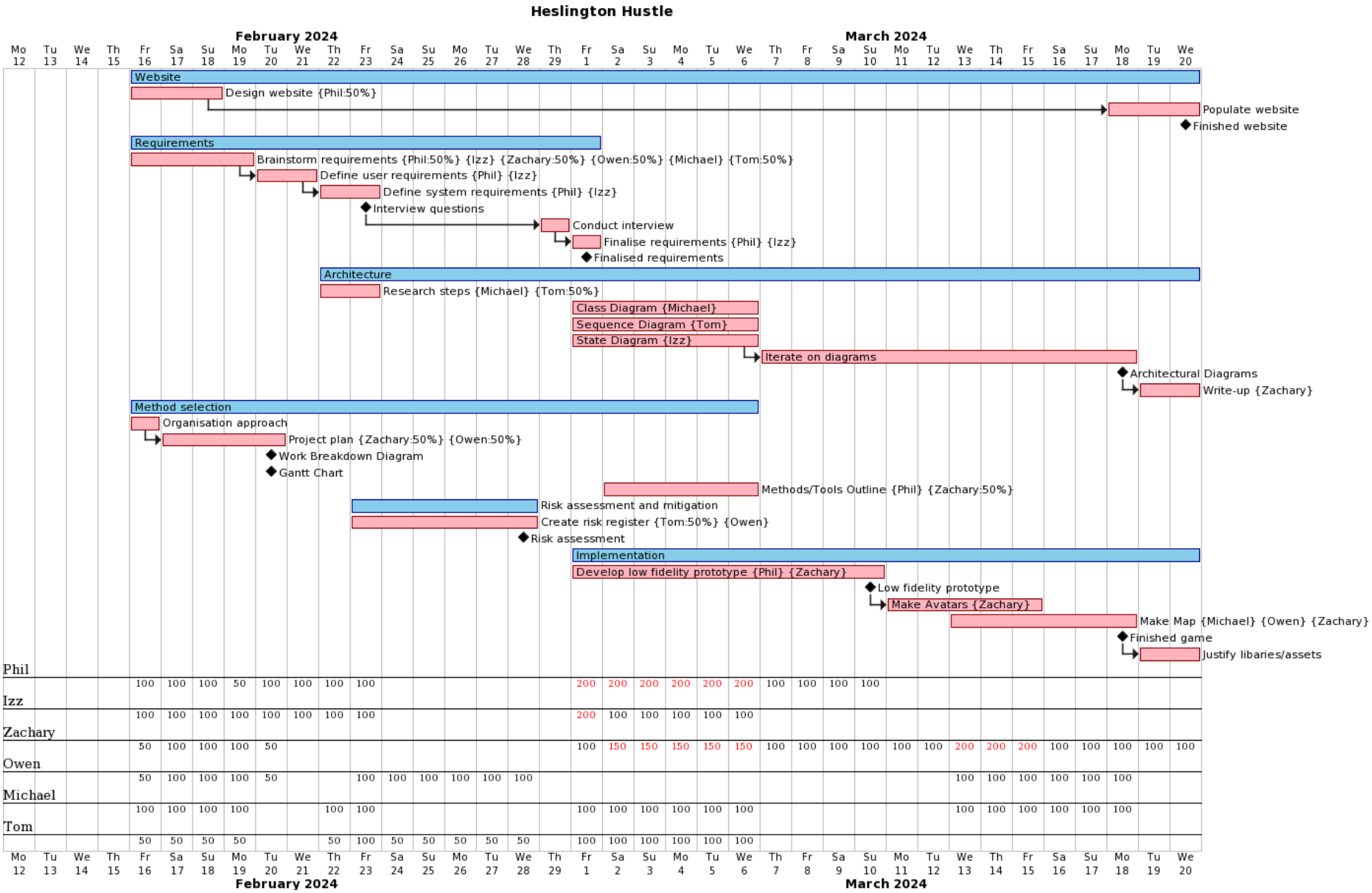


Although relatively simple, this chart allowed our team to comprehend the scale of the project and start looking forward to the deliverables each member may like to adopt.

Following this, we constructed a thorough Gantt chart outlining rough task completion dates, dependencies, and ownership for every task in our breakdown. This Gantt chart would prove to be pivotal over the course of the project as a constant reference of progress. We tried to be pessimistic in our projections to allow the team to stay ahead of schedule, but also to not rely on artificial overhead to complete overrunning tasks. Instead of distributing all tasks initially, we decided to revisit the Gantt chart every week to update it according to our progression, and distribute upcoming tasks amongst the team during our start of meeting scrum. We took a snapshot of the plan before updating it every week to keep a record of how the project evolved over time. These can be found on the “weekly snapshots” page of our website.

The final version of our Gantt chart can be seen on the following page. It is formatted as outlined below:

- Blue rectangles represent work packages, these are the six deliverables outlined in the assessment.
- Pink rectangles represent tasks, organised under the work package they belong to.
- Group members are allocated to tasks as shown in curly brackets e.g. {Phil:50%}.
- Black diamonds represent deliverables, arrows represent dependencies.


Heslington Hustle
February 2024
March 2024
Mo 12 Tu 13 We 14 Th 15 Fr 16 Sa 17 Su 18 Mo 19 Tu 20 We 21 Th 22 Fr 23 Sa 24 Su 25 Mo 26 Tu 27 We 28 Th 29 Fr 1 Sa 2 Su 3 Mo 4 Tu 5 We 6 Th 7 Fr 8 Sa 9 Su 10 Mo 11 Tu 12 We 13 Th 14 Fr 15 Sa 16 Su 17 Mo 18 Tu 19 We 20
Website
Design website {Phil:50%}
Populate website
Finished website
Requirements
Brainstorm requirements {Phil:50%} {Izz} {Zachary:50%} {Owen:50%} {Michael} {Tom:50%}
Define user requirements {Phil} {Izz}
Define system requirements {Phil} {Izz}
Interview questions
Conduct interview
Finalise requirements {Phil} {Izz}
Finalised requirements
Architecture
Research steps {Michael} {Tom:50%}
Class Diagram {Michael}
Sequence Diagram {Tom}
State Diagram {Izz}
Iterate on diagrams
Architectural Diagrams
Write-up {Zachary}
Method selection
Organisation approach
Project plan {Zachary:50%} {Owen:50%}
Work Breakdown Diagram
Gantt Chart
Methods/Tools Outline {Phil} {Zachary:50%}
Risk assessment and mitigation
Create risk register {Tom:50%} {Owen}
Risk assessment
Implementation
Develop low fidelity prototype {Phil} {Zachary}
Low fidelity prototype
Make Avatars {Zachary}
Make Map {Michael} {Owen} {Zachary}
Finished game
Justify libaries/assets
Phil
Izz 100 100 100 50 100 100 100 100 200 200 200 200 200 200 100 100 100 100
Zachary 100 100 100 100 100 100 100 100 200 100 100 100 100 100
Owen 50 100 100 100 50 100 150 150 150 150 150 100 100 100 100 100 100 200 200 200 100 100 100 100 100
Michael 50 100 100 100 50 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
Tom 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
50 50 50 50 50 100 100 100 100 100 100 100 100 100
Mo 12 Tu 13 We 14 Th 15 Fr 16 Sa 17 Su 18 Mo 19 Tu 20 We 21 Th 22 Fr 23 Sa 24 Su 25 Mo 26 Tu 27 We 28 Th 29 Fr 1 Sa 2 Su 3 Mo 4 Tu 5 We 6 Th 7 Fr 8 Sa 9 Su 10 Mo 11 Tu 12 We 13 Th 14 Fr 15 Sa 16 Su 17 Mo 18 Tu 19 We 20
February 2024
March 2024