Method selection and planning

Group 2 - The Debug Thugs

Bader Albeadeeni
Dan Hemsley
Jennifer Bryant
Oliver Elliott
Mathilde Couturier-Dale
Rosie-Mae Connolly
William Mutch

Methodologies

Methods and Justification

Our team has used a **Hybrid Methodology** throughout the project, with a focus on meeting deadlines, adapting to changes, and producing high quality code and deliverables:

- **Waterfall-Hybrid** for the **Project Structure**. The high-level plan for our project, consisting of the six assessment deliverables (D1-D6), is a Waterfall-Hybrid, visualised in the baseline Gantt chart below. For some of the deliverables, e.g. Requirements and Architecture, the dependencies between them and clear waterfall phases mean we have logical milestones and easy tracking of progress. The project also benefits from the agile approach of the methodology, as making iterations and prototypes allows us to be flexible and hit tight deadlines.

- **Scrum-Inspired** for **Project Management**. Scrum is an agile framework that focuses on iterative progress, which provides flexibility when managing the project structure. Our one-week work cycles act as our Scrum sprints, with the Weekly Snapshots being our 'sprint reviews/planning'. This management strategy allowed us to adapt our original Waterfall plan when the Requirements deliverable was delayed, being able to acknowledge the issue quickly and switch to a more agile approach for subsequent deliverables.

- **Extreme programming** for the **Development** of the project. An agile development methodology that has a focus on high quality code is ideal for the code-dominant LibGDX engine. XP has a fast feedback loop that benefits the project because we implement mechanics and test consistently, as shown by our frequent GitHub commits.

Development/ Collaboration Tools and their Fitness with Methods

The development and collaboration tools we used to support the project and the team working were:

- **PlantUML** allows us to create Gantt charts to visually represent the schedule of the project, and show how different tasks overlap and depend on each other. This supports our **Waterfall-Hybrid** project structure, and can be exported to provide us with Weekly Snapshots. We have also used it to create various structures for the Planning and Architecture deliverables.

- **draw.io** was used to create structural and behavioural diagrams using the Unified Modeling Language. The tool supported our **Agile, Scrum-Inspired** management when we shifted to developing architecture and code in parallel. The diagrams could be edited iteratively, and evolve with the game implementation.

- **IntelliJ IDEA** is compatible with Java 17 and LibGDX, and has the powerful debugging and testing tools needed for **XP** programming practices.

- **Git and Github** allows for Continuous Integration via commits and pull requests, which is needed to support **XP** development. Git is a version control system system that allows us to track changes in our code. Everyone in the team can access the code and view the version history to understand changes that have been made, which is also useful for **agile management**.

- **LibGDX** is a free and open source game development framework that provides the core Java components for game making. It suits our project as it is highly modular, allowing for **agile**, parallel development and design.

- **Google Drive** and an **Instagram** group chat allow us to collaborate on materials and communicate effectively throughout the project. The group chat allowed for a simple way to discuss progress and share 'Daily Scrum'-like updates with the team, adhering to our **Agile Management Methodology.**

Alternatives Considered

- **Methodologies:** We considered using **pure Waterfall** practices to structure the project, but rejected it as it would be too rigid to handle delays in deliverables. We also considered but rejected a **pure Scrum** methodology as it would not work well with multiple large, non-coded, deliverables. As a result, we chose a more flexible, hybrid approach.

- **Tools:** For developing the game we considered Unity, as it is compatible with Java code. However, we decided on LibGDX as it is specialised for writing games in Java , while Unity mainly uses C#. We decided to use draw.io over PlantUML for some of the architectural design diagrams, as its visual interface allows for more collaboration than PlantUML's text-based one.

Team Organisation

Approach

Our approach to team organisation was less structured, and didn't include the classic team roles of leader, researcher, teamworker, etc. Rather, we all worked as a group and the roles evolved more naturally, with people performing the tasks that made the most sense for them to perform at any given moment. We fell into roles suited for us rather than assigning them at the start.

Justification - For the Team

We felt that this approach would be good for our team for a few reasons. The first of which was that we didn't automatically fall into different roles, in that there was no clear person suited for each role initially. We felt that letting people fall naturally into roles over time would allow for better, more fitting roles than if we had just assigned them. For example, people who were better at organisation gravitated to the organisational tasks, meaning that roles were more suited to the people in them.

Justification - For the Project

As mentioned in the justification for the team, our method allowed our team to work in roles more suited to their skillset. WIth everyone working in areas that they are best suited for, it meant that the project could proceed more efficiently. This way of team organisation was suitable for the chosen software engineering methods as well.

Systematic Project Plan

This plan outlines the systematic approach we took to successfully deliver the required components for the ENG1 Assessment 1 maze game. We began by creating a detailed Work Breakdown Structure (WBS) by analysing the assessment paper. We decomposed the project into Deliverables (D), then their integral Work Packages (WP), and finally into small, manageable Tasks (T). This WBS was used to develop the baseline Gantt chart shown below, which serves as our primary guide for the scheduling, dependencies, and priorities of tasks. The evolution of this plan is discussed further below, and tracked via the weekly snapshots on our website.

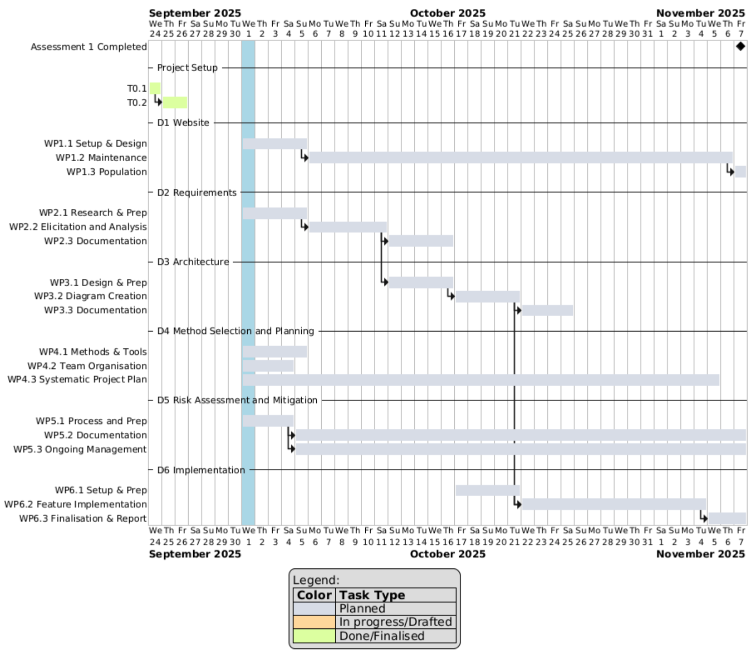
Key Tasks

The table below summarises our six key deliverables at a very high level. It outlines the timelines, priorities, and dependencies decided on at the beginning of the project.

Deliverable	Est. Start	Est. End	Initial Priority	Dependencies
D2 Requirements	Week 2	Week 4	High	
D3 Architecture	Week 3	Week 5	High	D2 Requirements
D6 Implementation	Week 4	Week 7	High	D3 Architecture
D1 Website	Week 2	Week 7	Medium	
D4 Planning	Week 2	Week 7	Medium	
D5 Risk	Week 2	Week 7	Medium	

Baseline Project Schedule

The Gantt chart below visually represents the baseline plan for the project, displaying the timeline, dependencies, and priorities of Work Packages, organised into sections by Deliverables.



The WBS, and weekly snapshots containing the working, more detailed Gantt charts are available on our website: https://thedebugthugs.github.io/#planning

Discussion of Plan Evolution

While the systematic plan on the previous page served as our initial foundation, it evolved significantly from creation. The planning process was tracked as a task, with WBS (T4.3.1) and Gantt chart (T4.3.2) not being made until Week 2. The dynamic evolution of this plan acted as our primary management tool, which was documented in the weekly snapshots. Through this adaptation, all deliverables were completed with a two-day extension of our initial deadline, 07/11/25.

The most significant changes to the plan are discussed below:

- **Initial Slippage in Requirements Documentation (Weeks 3-4)**
    - **Change:** The initial plan for D2 Requirements underestimated the time needed for documentation, which delayed the start of D3 Architecture.
    - **Reason:** The slippage occurred in WP2.3, which is visible on the Week 3 Gantt chart and Snapshot. We identified that D3 Architecture had a critical, unplanned dependency on a finalised T2.3.2 (Requirements Statement), for aiding design and traceability. The Requirements Statement was put as top priority for the next week.
    - **Adaptation:** The delay downgraded our status from "On Schedule" to "At Risk". Our Week 4 Gantt chart illustrates how the entire D3 Architecture process was pushed back by a week, as it could not have been started earlier.

- **Shift from Waterfall to Iterative Development (Weeks 4-5)**
    - **Change:** We made the strategic decision to remove the strict Waterfall dependency between D3 Architecture and D6 Implementation.
    - **Reason:** The Week 4 snapshot shows that the project was "Behind Schedule". Delays in starting some components of D3 Architecture made the linear plan unfeasible.
    - **Adaptation:** The Week 5 Gantt demonstrates WP6.2 (Feature Implementation) and WP3.2 (Diagram Creation) being 'Drafted/ In Progress' in parallel. This iterative approach allowed the project to improve to "At Risk" status.

- **Dynamic Task Prioritisation (Weekly)**
    - **Change:** Our weekly priorities did not come from the initial static table, but on immediate schedule risks.
    - **Reason:** Managing the project needed a clear process.
    - **Adaptation:** Our highest priority became any task that was 'Planned' (gray) but had passed its start date (e.g. T2.3.2). These tasks were moved to the start of the next week (Wednesday, as our main meeting day), to be done immediately. 'Drafted' (yellow) tasks were a lower priority, as these could be revised later, allowing us to focus more effort on the delayed critical path.

- **Finalisation and Deadline Extension (Week 7)**
    - **Change:** The project deadline was extended by 2 days, as shown on the Week 7 Gantt milestone.
    - **Reason:** Provided a necessary buffer to complete all documentation.
    - **Adaptation:** The Week 6 snapshot shows our priorities shifting from feature creation to final documentation e.g. WP3.3 (Architecture justification).

References