Method selection and planning - Plan1

Dragonite

Team 21

Omar Omar
Rhianna Edwards
Okan Deniz
Craig Smith
Omar Galvao Da Silva
Joel Wallis

Team 19

Julia Kunikowska
Jack Longmuir
Justin Mendon
Luke Roberts
Douglas Sword
Andrea Zhu

(1) An outline and justification of the team's software engineering methods

There exist many software engineering methods we could choose from. These are split into three types: (1) plan-driven methods, that have an overall project "roadmap" to guide the project through its lifecycle; (2) agile methods, that focus on dealing with volatility & customer-driven development (3) soft methods, that are employed when the problem specification is not known in detail (usually before SWE is started, to help determine requirements).

Our principal software engineering development methodology is Agile. We chose this because:

- It copes well with volatile requirements, which we expect this project to have.
- It works well in small teams.
- It employs short iteration cycles, with low documentation overheads, focusing on maintaining an inclusive relationship with the client.
    - This allows us to incrementally deliver working prototypes and get immediate feedback for the next iteration.
- We also felt the "standup" concept was incredibly useful, and even adopted it into our overall planning and organisation model.

This contrasts to plan-driven methods which have a large documentation overhead, and as a result are very inflexible.

We note that Agile dictates that face-to-face communication is better than online, but this is unfortunately not possible with the current pandemic.

(1) Developmental / Collaborative tools we have used, and (2) our approach to Organisation

Our codebase is stored in a repository on GitHub, which we felt was the clear leader for version control and enabling team-working on a shared programming project. Additionally, it has inbuilt integration with VS Code -- a working environment many of us use regularly -- and the platform in general has by far the most tutorials and useful resources available.

- Alternatives include Bitbucket by Atlassian or other repository hosting tools / version control tools, but we felt GitHub does both of these things best.

Primarily we collaborate on Discord, using the platform to communicate (both in calls and through text), share files, and self-structure both the assignment and current progress of deliverables.

We believe Discord is fit for this purpose for the following reason:

- We all had existing Discord accounts we could use, and were familiar with the platform and how to use it.
- Discord permits creating a custom server and modifying its structure to best suit the application, with numerous pre-built templates to speed up this process.
    - This allows the server to be flexible and accommodate multiple facets of the engineering process. The centralised approach facilitates increased working efficiency and coordination, and reduces the complexity of working across multiple applications.
    - For example, we have text and voice channels for each deliverable, allowing uninterrupted communication between assigned team-members.

Possible alternatives we considered included Slack, Facebook Messenger, and Zoom.

- Slack is the closest proxy to Discord, however we felt that our lack of experience with it (and few of us having pre-existing accounts) was enough reason to choose Discord as they are very similar anyway. Furthermore, to unlock all of the useful features Slack offers requires a monetary subscription.
- Facebook Messenger has both voice and video call options, but poor support for file sharing (including previewing embedded images etc.) and no ability to structure a group chat in any way. Zoom had similar drawbacks.

A planning tool we use is Trello. This introduces the concepts of Kanban-style “Boards”, which can be created for projects and used to keep track of tasks.

- It allows us to keep track of task priority, status, and assigned members, along with a short description and a task deadline.
- The board is split up into “To-do”, “Doing”, and “Done”, which allows us to asynchronously self-organise and determine which [sub-]tasks need to be completed.

We chose Trello since a few of our members had experience with it already, and it is generally well-regarded as a project planning tool due to its intuitive UX.

A possible alternative to Trello would have been Jira by Atlassian, but the free plan of 2GB of storage we felt was insufficient as many attachments will be required when collaborating on, and updating the team about, documentation.

In the requirement interviews, the client made it clear they would prefer custom made graphics. When deciding on software to use for the graphics of the game we considered two main programs, PiskelApp and Photoshop. Photoshop is the more professional of the two however, we found it was more difficult for new users to pick up, which would have slowed development, and the limited time we could use for free made it unreliable. So we have chosen to use PiskelApp for producing graphics.

For producing fonts, we had two choices: Gdx Freetype and Hiero. The former had trouble with font scaling and required more time for its implementation in testing, so we decided on Hiero to generate the font files required for the game. The font files needed to be in a “.fnt” format and this was one of the only free up to date tools we were able to find.

To develop the game we used the engine LibGDX. This has strong ties to the Java community and lots of detailed documentation and information online on use. It’s also free to use commercially, and has a project generator tool which includes all dependencies.

Team Organisation

We have the following deliverables. Assigned to each are various members of the team. Larger deliverables are assigned more members as needed.

- Continuous Integration -- Dougie (peer reviewed / supported by Justin)
- Change Management Report -- all members will document changes to assessment 1
    - Architecture -- Justin, Jack (peer reviewed / supported by Andorea)
    - Method Selection and Planning -- Dougie, Justin (peer reviewed / supported by Julia)
    - Requirements -- Julia, Andorea, (peer reviewed / supported by Justin)
    - Risk Assessment and Mitigation -- Luke, Justin, Jack (peer reviewed / supported by Dougie)
- Testing -- all members of the team contributing at various points to write and perform test
- Implementation - DragonBoat Race game, programmed in Java -- all members of the team contributing at various points.

Jack was assigned secretary and takes down minutes / a summary of all meetings and discussions held, to ensure that key resolutions reached are recorded persistently.
Luke leads the team on the code base, and acts as Scrum leader. Pull requests must be reviewed by at least one other member of the team (this is enforced by GitHub).
Dougie leads the organisation of the project in general, as he was assigned to the Planning deliverable and as a result was required to identify key tasks & their priorities, thus was well-placed to assign deliverables to available team members.

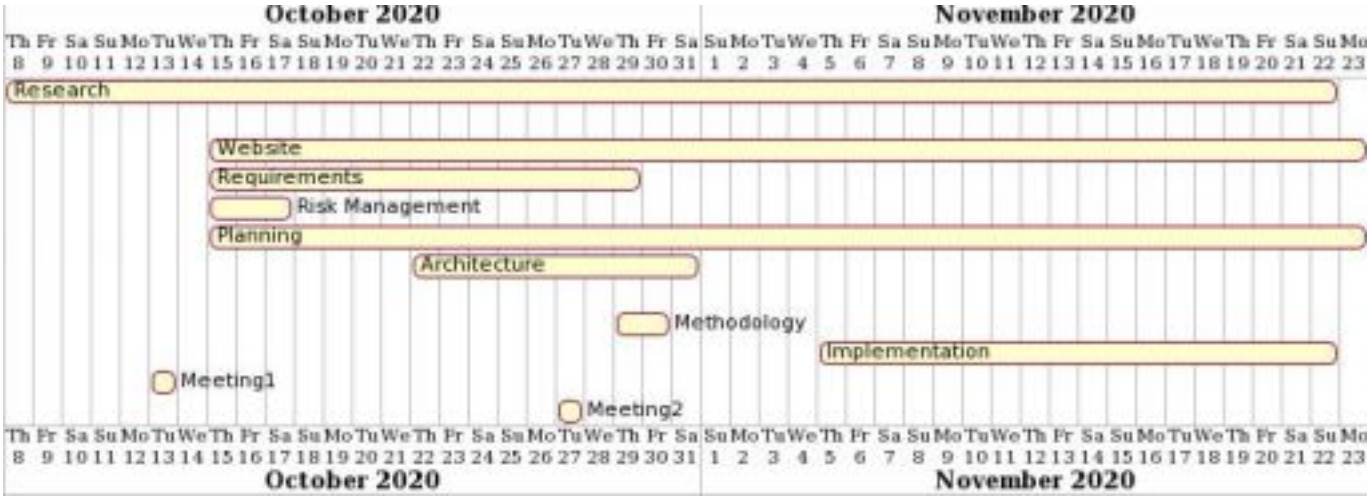
Stakeholders include the University of York communications office (who will use our game for promotional activities), and Dr. Javier Cámara, who represents the customers and final users of the game. Our agreed method of communication with the customer is voice call, arranged via email 24 in advance. We had weekly slots where we could schedule a meeting, and took advantage of this, posing any questions / clarifications we desired about the requirements (and product in general) to the stakeholders.
Hardware and software resources: the customer has specified that the program should be written in Java to facilitate cross-platform play, but with a primary target platform of standard Desktop PCs with a display ranging between 15-27 inches, and standard peripherals (keyboard and mouse).

We feel that this arrangement is suitable because we have distributed work and responsibilities fairly between the team, assigning team members to roles that played towards each member's key strengths. In Assessment 1 this structure was shown to work well as we finished well before the deadline.

Systematic plan for Assessment 1

The key tasks are listed below, this outlines everything that we will be doing in order to cover all parts of the project. It breaks down the major tasks into smaller ones. This allowed the team to easily keep track of our progress as we colour coded the key tasks once we started/completed it.

1. Website
    a. Link all assessment documents to the website
    b. Ensure website is well structured
2. Requirements
    a. Evidence research into requirement specification and presentation
    b. Table of requirements with environmental assumptions, possible risks and alternatives
3. Architecture
    a. Decide on UML
    b. Represent relationships between entities
    c. Justifications of the structures – how abstract -> concrete
4. Methodology
    a. Outline methods(agile), identify development/collab tools used and alternatives tools considered. All with justifications
    b. Team's approach to team organisation – why is it appropriate for the team and the project.
    c. Systematic plan – gantt chart, critical path and task dependencies.
    d. Weekly screenshot of the plan on the website and discussion of how the plan has evolved.
5. Risk management
    a. Introduce and justify risk format and level of detail
    b. Table of risks, likelihood, impact and mitigation
6. Implementation
    a. Zip file / link to github of the documented code with an executable JAR file. State features that are not fully implemented.



The critical path is: requirements, architecture, implementation then website. The architecture is dependent on the requirements. Then the implementation is dependent on the requirements, risk management and methodology. Finally the completion of the website depends on all the other tasks being completed as everything has to be uploaded into the website.

Systematic plan for Assessment 2

We highlight that initial plans will most likely be wrong, our project planning must be iterative and change as our understanding of the requirements evolve (and the technical work progresses). Thus we aim for our plans to drive activity, and for the activity to subsequently inform planning. Through good communication and division of tasks, we can adapt as we move through the project.

The critical path is: CI, change management, requirements, plan, risks, architecture, implementation, testing then website and presentation. Change management encompassed all other tasks (apart from CI) so needs to be done for the start. The architecture is dependent on the requirements. The implementation is dependent on the architecture, requirements, risks, CI and plan. The website and creating a presentation depends on all the other tasks being completed and being uploaded.

1. Website and Presentation
    a. Link all assessment documents, Java Doc and manual to the website
    b. Create promotional material for the product
2. Continuous Integration
    a. Identify workflow to use
    b. Implement CI Server
3. Change Management - document all changes to original documents
    a. Requirements
        i. Identify new requirements for new implementation and introduce any missed
    b. Architecture
        i. Implement our new classes and changes to classes in the architecture
    c. Methodology
        i. Identify any changes we are making to Team 21's plan and justify why and if we are, create systematic plan – team organisation, gantt chart and critical path
    d. Risk management
        i. Add any missed risks to the table and fix any issues with the table
4. Testing
    a. Identify testing approach
    b. Implement tests for existing codebase and for additional features
    c. Create test report and upload proof of tests to website
5. Implementation
        i. Complete assessment 1 product and add assessment 2 new features
        ii. Add changes to original codebase to changelog and comment changes
        iii. Document code with Java Doc and comments and upload JAR file to Github.
        iv. State any requirements not implemented