Method Selection and Planning

Edited by Team 13:

Darcy Adams

Samin Alborzi Movahhed

William Dunlop

Nam Duong

Jakub Grizmil

Davids Kacs

Team 15:

Joe Wrieden

Benji Garment

Marcin Mleczko

Kingsley Edore

Abir Rizwanullah

Sal Ahmed

Method Selection and Planning

Software Engineering Methods

Scrum is our team's software development method of choice, as this agile methodology is suitable for small teams, and shall allow us to deal with changing requirements in the future, especially in relatively short time constraints. Along with this, it is in our best interests to reconvene in a weekly Scrum meeting, in which we reflect on our progress throughout the week, checking that every requirement in our previous stage maps onto at least one requirement in our current stage, and address issues which may have been initially overlooked before greater complications can arise, as well as have frequent team walkthroughs of our artefact throughout its incremental development stages - from the initial requirements to the solution we will pose.

- Other agile frameworks such as XP were considered, but Scrum was chosen due to it being up to the team to prioritise work according to that which they know is needed, as well as having less of a focus on strictly following particular engineering methods (e.g paired programming) that XP advocates.

- Focusing on the principles of Scrum rather than the practices (e.g. daily Scrum standups, which we are unable to do due to workloads from other modules; user stories are not a viable option as we knew the system requirements from the onset), allows us to be flexible with how much each person can contribute at a time depending on their circumstances, but also ensuring equal overall contribution through the project schedule, which means our bus factor remains high (more on this later in "Team Organisation").

Development and Collaboration Tools

The Java framework we decided on is libGDX.

- We had considered using LWJGL, a Java library that provides access to native APIs for graphics and audio. However, libGDX uses this library in its framework, and since code reuse is an essential part of Software Engineering, we decided we would use the higher level methods that libGDX provides. Another thing that we had considered was using jMonkeyEngine which is a game engine that comes with a software development kit. However, this game engine is used for 3D games and our 2D game didn't fit with our ideas.

- On that note, libGDX is quite seamless in that the majority of the program consists of regular Java, which our team is familiar with. The high-level libGDX methods may simply serve to render graphics to the screen, for example.

Our IDE of choice is IntelliJ.

- IntelliJ IDEA is the recommended IDE by the libGDX team, and so it has the most support for the framework.
- We had considered Visual Studio Code as an alternative, however, we had difficulties with VS Code during our findings, specifically with using the installer from libGDX, which creates projects using gradle.

- We had also considered Eclipse but we fairly quickly decided against Eclipse because as a team we did not feel comfortable enough with Eclipse compared to the other two.
- IntelliJ on the other hand does not have this issue, and thus in learning how to use libGDX, IntelliJ is often referenced, making it the ideal choice. Furthermore, IntelliJ lends itself to a project-style of development. It has features that aren't present in VS Code that make it easier to work on large scale projects with multiple classes such as automatic class refactoring.

For version control, we are using Git with our repository stored on GitHub.

- This allows us to avoid collisions and other inconsistencies when merging different members' works together.
- Furthermore, hosting the repository on GitHub allows us to easily maintain version control history and even gives us access to GitHub Pages which allows for easy access to documentation.
- Git has native integration in IntelliJ thus making it easier to pull and push changes.
- Github is also a great help when it comes to writing up excellent documentation.
- Github Action also helped us to set up continuous integration using workflow templates.

For software testing, we decided to use Junit4 with mockito.

- Mockito is the most popular testing framework for Java which can let users write tests with a clean and simple API. Tests are very readable and they produce clean verification errors.
- We chose Junit4 because we found more material in our research to set up a Junit4 framework and utilising it rather than Junit5 framework. We also found that Junit4 works better with existing tools to test LibGDX.

For online meetings, we use Zoom.

A weekly scheduled Zoom Scrum meeting allows us to communicate via voice and text chat as well as sharing our screens for input from other members for any problems encountered. Zoom also gives us a platform to engage in Team-Customer meetings should we have any questions or queries for the client or vice versa.

For further collaboration, we use Discord.

Separate text channels allow for easy communication on the relevant issues in each channel, along with support for uploading various file types. Voice channels allow us to communicate with each other in a less structured way when we need to collaborate before the Scrum meeting.

For the storage and collaboration of documents, we use Google Drive, making it easy for several members to update the same document and edit in real time through the version control functionality.

We Also considered using Slack for our communication. Slack is a communication platform targeted towards companies, However we felt that all of the above collaboration tools were enough, so we decided not to use Slack.

Team Organisation

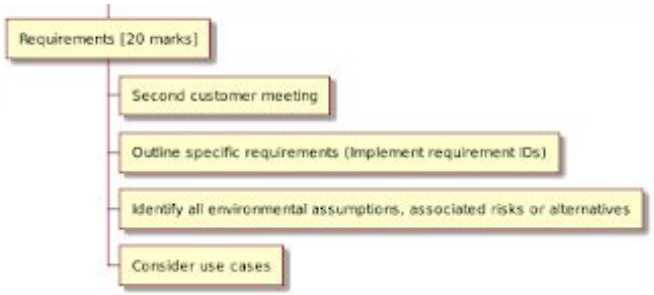
In terms of team organisation, we did not assign anyone a set role, however our meetings do have structures in terms of everyone does something during the meeting. Our Team leader

is William Dunlop who is usually more comfortable with doing most of the talking and assigning tasks to each member of the team. Usually, during our team meetings, someone has the role of scribe and taking the key points/notes of the meeting which can help us remember the work we have done each week. Also, sometimes the tasks are assigned to team members using Github projects.
At the beginning of the project, During the first couple of meetings, we discuss the project brief, making sure everyone understands it and requirements are usually discussed within the beginning week as the whole group works together.
Once the requirements are set, everyone is assigned to a different task with a deadline (which is usually the next group meeting). Davids Kacs is the member who is the most comfortable with the code implementations of the project but usually every group member is assigned a coding task, in groups of two, taking part in the coding of the project so we have a good bus factor. During the meetings, we check everyone is up to date with their tasks and we keep track of the progress of the project, usually done by the team leader.
Similar scenario happens when assigning group members to do certain parts of the code documentation. Apart from risk assessment which for the most part, everyone has a say in them, other parts of documentation is assigned to individual members or members in groups of two (depending on the amount of workload and our time left). All works done by individuals are eventually proofread by one other or more members of the team.
We believe that this form of team organisation is appropriate because although we may have one or two members of the team who might be more comfortable/skilled in some areas of the project, we believe that nobody has developed a specific field of expertise yet and we are all quite flexible with what we can do. Also, this approach not only increases our bus factor but can have a great impact on the individuals' learning. Giving everyone a chance to perform a different task each week, means we can get the task done as well as letting the individuals have the freedom to do the tasks they are more comfortable doing.

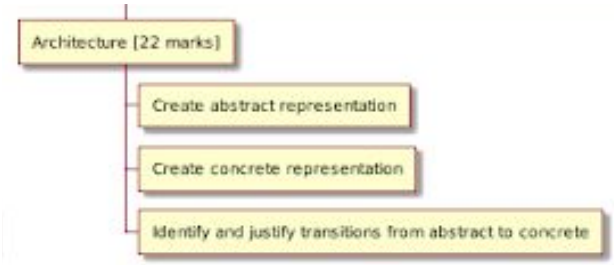
Project Breakdown

(For assessment 1)

The website is to be the face of our project, an area for which all of the documentation and updates about the project can be viewed by the client.

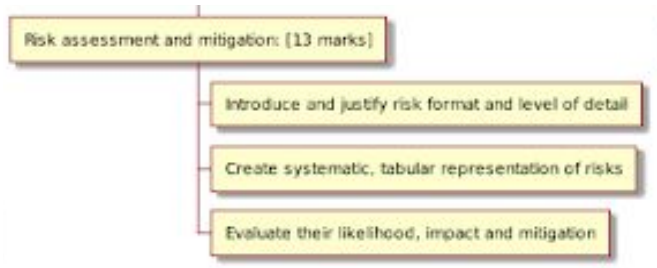


Requirements are elicited from a Team-Customer meeting, then each User, Functional and Non-Functional Requirement is defined.

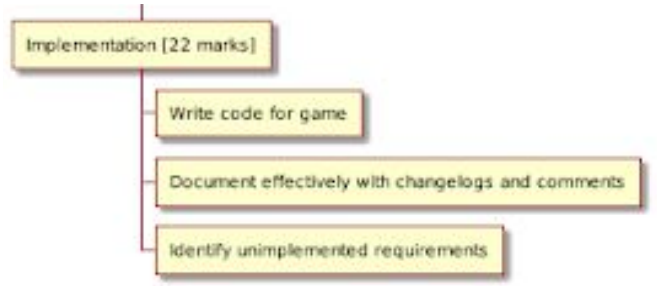


The Abstract Architecture is created as a diagram allowing the team and the client to understand how the implementation is laid out, with a Concrete representation to be made from it.

Software Engineering methods are selected and the project is planned using a Gantt chart.

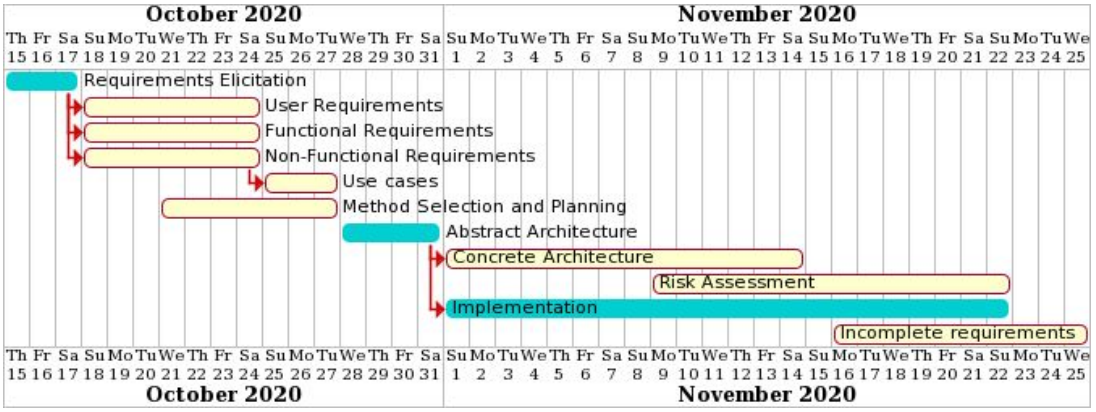


Risks are identified, represented, and evaluated based on their likelihood of occurrence and their impact should they occur. Methods to mitigate these risks are also discussed.



The product is implemented using Java and libGDX, completed with documentation. Any requirements that will not be implemented should be discussed and justified.

Project Plan (For assessment 1)



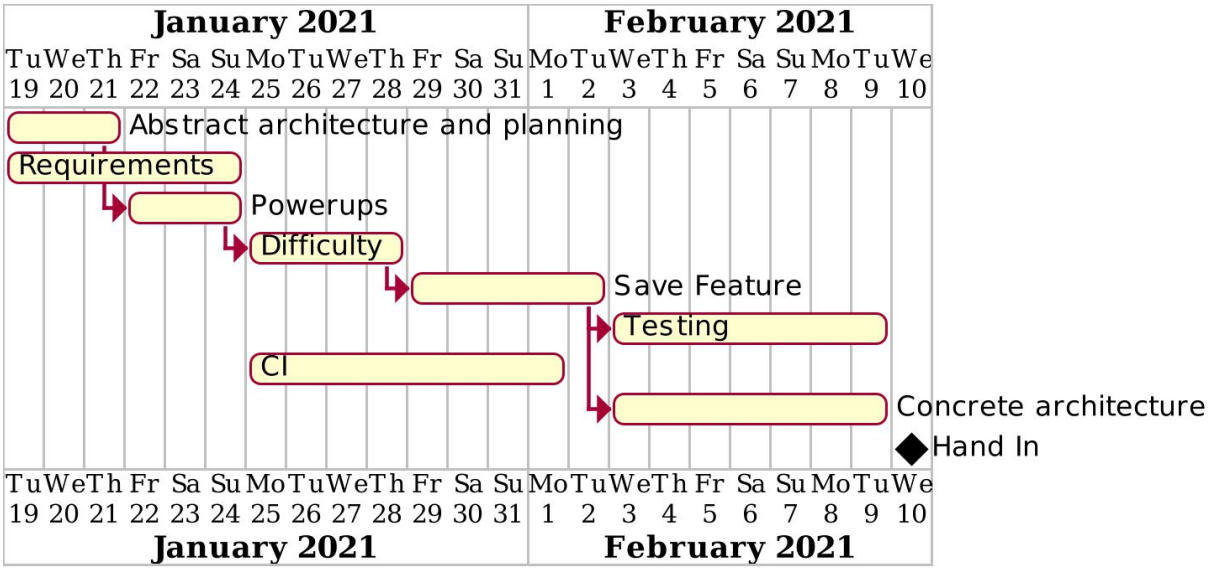
Tasks that are **dependent** on others are denoted by a leading red arrow:

- Requirements elicitation dictates what features we must implement and thus what our Architecture and Method Selection will be. Defining each kind of requirement cannot be completed without first eliciting the requirements with the client in a Team-Customer meeting. There may be several of these over the course of the project but this initial elicitation is vital.
- Describing specific use cases for the product depends on knowing what the product should do, as denoted by the requirements.
- Abstract Architecture is a high priority as without it, the implementation can't begin.
- The Implementation and the Concrete Architecture can be completed simultaneously as the implementation of the product will help define what the architecture looks like and vice versa. However, neither can be completed without first outlining an Abstract Architecture to give a plan for which entities should be programmed.

- Discussing the unimplemented requirements can be started during Implementation as the development team should have a good understanding of how much they can get done.

The **critical path** consists of Requirements → Abstract Architecture → Method Selection → Implementation. The plan above assumes that each task takes the longest possible amount of time to allow room for any risks that may arise. Changes to how long each task takes and when they begin/end is discussed on the website.

Project Plan **(For assessment 2)**



19/01 - 25/01

- Requirements are already existing within the documentations for assessment 1 (team15) so we checked them and added a few more in order to include the new features requested as a part of the Assessment 2 criteria.
- Made an abstract architecture
- Implementation started with working on powerups and getting them to work

26/01 - 01/02

- Strategy of change management finalized
- Then we moved to add difficulty to our implementations (with a separate screen to select difficulty).Then we started to implement a pause screen and savings. Alongside these implementations we started researching CI and some testings.

23/01 - 09/02

- Putting CI in place and writing tests , as well as finishing off the implementation.
- The writing of documentation is now divided between group members according to the group work and implementations done.
- Produce a concrete architecture

09/02 - 10/02

- Final checks to make sure everything is in place, all requirements are met, risks are identified, the documentations are proof read
- The same website which we made in assessment 1 is used in this part of the project and with doing a few tweeks so the website will be updated and making sure the documentations are in their right place on the website.

The **critical path** consists of Requirements → Abstract Architecture → Implementation (powerups, difficulty and savings)→Concrete Architecture.