Method Selection and Planning

Team 15
Team 14

Joe Wrieden
Benji Garment
Marcin Mleczko
Kingsley Edore
Abir Rizwanullah
Sal Ahmed

Babar Khan
George Rogers
Jacob Turner
James Crump
Chloe Remmer
Shijie Lin

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 that XP advocates.
- Development is carried out in weekly sprints, culminating in the artefact's current iteration for the week being provided as a deliverable to the client so they may be included in the development process should they wish to have access and provide feedback at any point.
- Focusing on the principles of Scrum rather than the practices, 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.

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.
- This also allows us to focus our time during each Scrum sprint on solving issues that relate to the task given to us by the client, rather than spending the time to learn to use LWJGL as it provides low-level access.
- 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.

- 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.
- It allows members the ability to access the code after each sprint to understand how it works in order to reduce the effects of the bus factor, especially since members are working remotely.
- 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.
- Finally, Git has native integration in IntelliJ thus making it easier to pull and push changes.

For design diagrams we have chosen to use PlantUML as we have found it an easy way to create clear and concise diagrams, such as Gantt charts.

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.

Finally, 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.

Team Organisation

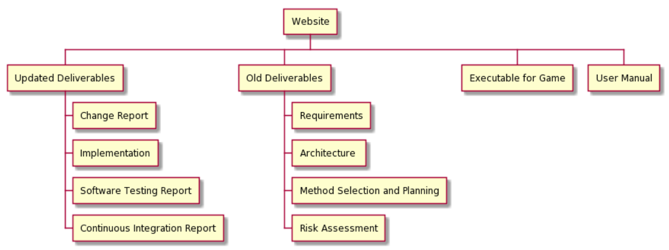
Our tasks, as described below, have initially been distributed in a way that allows two of our members, Joe and Benji, to focus on learning the technical aspects of the development and programming, while the other four members take on refining the artefact's successive design versions and its documentation. This meant that while the programming team familiarise themselves with libGDX, the other members are to define the software architecture from the requirements which had been elicited, carry out planning, as well as identify risks and their respective mitigations. The reason we feel it is appropriate to split the team into two groups is so that people can work with their strengths and productivity can be boosted. The documentation is worth three times more than the code, which is why we have decided to put more people on writing documentation. We decided against putting 5 people on documentation as this would greatly decrease the bus factor, since there would be only one focusing on code. Following their findings, the programming team is to inform the rest of the group on how to lay out the concrete architecture, such as detailing technology-specific methods. They will then continue to implement the product while the rest of the team write-up the other tasks as laid out below. The documentation team has decided to pick up a section a week rather than designate one section to one team member. There are a few reasons for this: first of all, we manage to get four times the amount of ideas on each section, secondly, the layout/style of each document is going to be the same. Thirdly, having four people working on a section each week increases the bus factor. And lastly, people can combine their written-communication and technical abilities.

As we are working with a SCRUM methodology, we also defined the following roles for our team:

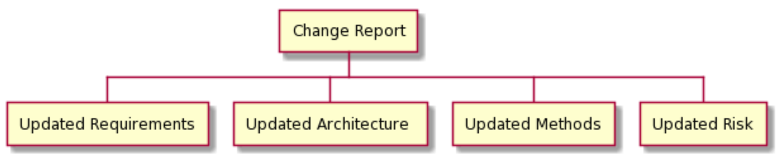
- SCRUM Master: Babar Khan
- SCRUM Product Owner: Chloe Remmer
- Development Team: George Rogers, Jacob Turner, James Crump, Shijie Lin

Apart from the designated SCRUM roles, our team also separated into two, where Babar Khan, Jacob Turner and James Crump focused on the Software Development whereas George Rogers, Chloe Remmer and Shijie Lin focused on the documentation. This enabled us to consistently produce a good amount of work each sprint whilst reducing the bus factor.

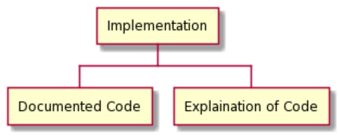
Project Breakdown



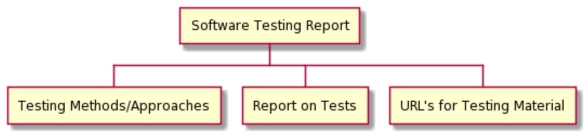
The website is to be the face of our project, an area for which all of the documentation and updates about the project can be viewed by the client. We have chosen to break down the 'Updated Deliverables' section further as it requires several more components



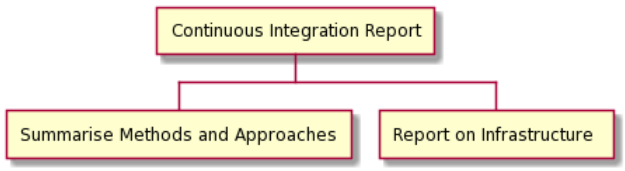
The Change Report consists of updating the old documentation to meet the new Requirements and features of the game



The Implementation requires extended sections of documentation code and a further explanation of any new features or changes made to previous code

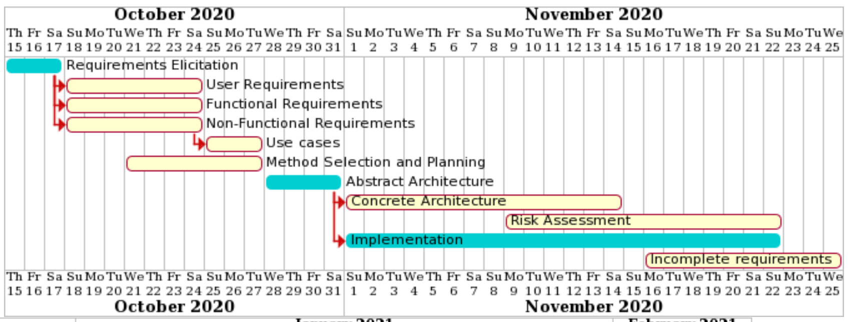


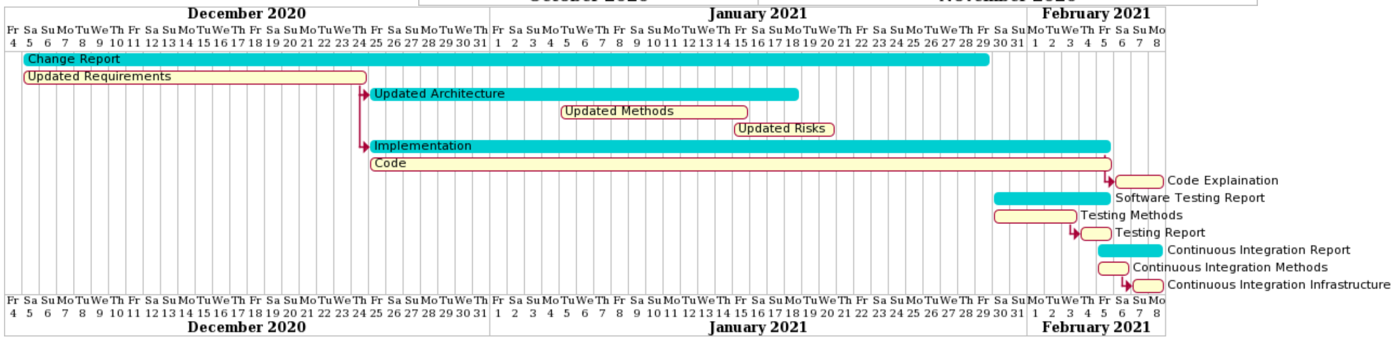
The Software Testing Report requires the summarisation and findings of our testing methods


Continuous Integration Report
Summarise Methods and Approaches
Report on Infrastructure

The Continuous Integration Report summarises our continuous integration methods and approaches, relating to our project. We also have to provide a report on the infrastructure used

Project Plan





The plan above describes each individual task that must be completed and in what order.
High priority tasks are coloured in **blue:**

- Change Report dictates what features we need to change within the code so it decides what our architecture should be
- Updated Architecture tells the software team what they need to include in the code and how to lay it out to keep it consistent
- Implementation is the finished product and, as we are nearing the deadline, is important
- Software Testing Methods dictate how stable the game is and if it performs as it should
- Continuous Integration Report tells the software team how to merge the code and, since 3 people are working on it, they need to know the same information

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

- 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.
- 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 Updated Architecture can't be decided until all requirements have been updated and elicited
- Implementation cannot begin until the architecture has been updated so that the software development team have a clear understanding of what code is necessary and can organise their tasks
- Code explanation cannot be started until the code has been completed as some areas may change as the code is updated and built upon
- The testing report can't start until the software testing methods have been decided upon and implemented
- The continuous integration infrastructure can't be commented upon until it has been decided and implemented

The **critical path** consists of [each of the User, Functional and Non-Functional] Requirements → Abstract Architecture → Method Selection → Implementation → Change Report → Implementation → Software Testing Report. 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.