Method Selection and Planning

Team 14 - Taken On By Team 15

Joe Wrieden

Benji Garment

Marcin Mleczko

Kingsley Edore

Abir Rizwanullah

Sal Ahmed

1 Software Engineering Methods - Part A

1.1 Methodology

Our goal in this project is to deliver a product that is of good quality and is in accordance to the expectations of the stakeholders. To supplement this, we researched the practices associated with software development.

One practice we decided to look into was Agile methodologies. Advantages of using this approach consist of its being suitable for small teams and is useful in dealing with perhaps unforseen changing requirements. We decided to follow an Agile methodology, as we believed it would allow us to pick up any bugs or errors quickly in the process as well as incrementally improve and add to the game code. An Agile methodology would also guarantee that there would be predictable and guaranteed delivery of features every week which we decided was important due to the limited time available for the project.

Following this, we decided to use the SCRUM framework which follows the core principle of continuous improvement whilst also being a good framework for dealing with any changes in the user requirements. We also believed that having a Product Backlog (a 'to-do' list for SCRUM) and a Sprint Backlog (the goals of the current sprint) would help keep us on track during the sprints.

Alternatively, we had considered a Waterfall methodology, however the rigid processes and the difficulty in adjusting the product to customer feedback after completion meant that we didn't believe it would be suitable for our project, especially considering client feedback and communication was important to us. Also, if any bugs or errors were encountered towards the end of the process it would mean having to restart a lot of the other processes to fix them, which wouldn't have been suitable for our deadline. Development would be carried out in weekly sprints, with the goal of the sprint being the artefact's current iteration being able to be provided as a deliverable to the client so they too may be included in the development process should they wish to have access. Thus, they could provide feedback at any point that a concern of theirs may be raised.

1.2 Collaboration/Development Tools

In order to facilitate team cooperation and planning we utilised Zoom, Discord, Github and Google Drive.

Following the SCRUM Agile Methodology, we had weekly meetings using Zoom calls in order to discuss ideas, allocate new tasks and update the entire group on the progress of previous tasks that were allocated. Alternatively, we had considered using Google Teams but agreed that, as Zoom was more commonly used, it would be a better fit as there would be no adjustment period to using a new platform.

During the rest of the week, we agreed to use Discord to communicate as we could split the server into multiple channels, such as: Java-Progress, Design-Progress and Tasks. We felt that this way we could better separate out what needs to be done and, when you look in a specific channel, all the information would be relevant. We had considered using Slack, but, once again, found that Discord was more commonly used and the group were more likely to utilise a platform they had used regularly than one they hadn't.

A Google Drive was shared between all members of the team and all work, such as designs and deliverables, were to be stored there for easy collaboration. This way the Software-Development team could access the designs the Design team created without needing to constantly ask. It also reduces the risk that work could be lost as there will always be a copy in the Google Drive.

Github was used to manage the code for the project, it allows us to work remotely and merge any new snippets of code into the project. This way we can all work on different parts of the game at the same time. An alternative could have been Bitbucket, however, unlike GitHub, Bitbucket does not make use

of a desktop client, and it may have been a bit difficult to get the group members unfamiliar with version controlling up to speed with using command line interfaces within time constraints, so we used GitHub.

The UML diagram creation tool we used was Draw.io. We experimented with PlantUML but we found Draw.io to be much more seamless and intuitive to use, as well as having room for both customisation and real time collaboration which the alternative lacks. As well as being light and easy to use, without use of vector type software.

We used Team Gantt to manage our project through the Gantt chart we created. It allows for task level communication, which was found to be very useful. Additionally, it is user friendly, so we didn't have to spend much time planning the project as we would have had to, had we used a less approachable tool. Before settling on this, we tried nTask, however this was limiting in the number of team members we could collaborate with without payment, and so due to financial constraints we settled on Team Gantt. However, as we realised when we later added task dependencies to our Gantt chart, it was a bit unclear as to which tasks were task dependent on another, and the number of lines was a bit confusing and messy. Nevertheless, it was better than alternative tools and it had done the job, so we stuck with using Team Gantt.

LibGDX is the library we settled on, as it has lots of resources (tutorials, open source code and an active community) to learn and get help from, which has helped us in getting grips with it in limited time constraints. We were also restricted by the language specified in the product brief, which was Java, so could only use Java frameworks. An alternative was LWJGL, but this is less easy to pick up quickly due to its only allowing lower level access, so we decided against it.

Our choice of IDE was initially Visual Studio Code, however after learning of the numerous plugins, one of which we used to construct the concrete architecture (), that IntelliJ boasts over VSC, we settled on IntelliJ for its greater functionality.

2 Team Organisation - Part B

2.1 Team Roles

During the first meeting of the group, we outlined a Team structure according to each individual's strengths and what they wish to accomplish during the project. We split the group into two divisions: a Software team and a Design team. The Software team was responsible for the implementation of the project, including the architecture, whilst the Design team would handle any artwork, requirements and other documentation work. This structure meant that there would always be one other person capable of working on someone else's task if the initial person was unavailable, as long as they were in the same team. For example, at least two people were capable of understanding sections of the game code, which helps provide a high-bus factor. Within the team, any tasks that need to be completed are broken down into smaller chunks and then distributed equally, which keeps the workload fair and manageable.

The sub-teams created were interchangeable, so that the flexibility in bus factor was still kept, and for people to have freedom in what they wanted to work on next. The role of Scrum Master was allocated to Kingsley, as he was the most prominent force for timetabling meetings as well ensuring the team followed the processes and practices that were agreed. As well as this, he maintained regular contact with the client. The Product Owner Role was jointly held by Abir and Marcin as they were involved in relaying the customer needs to the development team acting as the intermediary between both the stakeholders and the Scrum team. The Development team role was held by Benji, Joe and Sal, as they had the greatest experience with programming in Java and were heavily involved in the development of the project. The role of team coordinator was taken on by Sal, as he was adept at forming the bridge between the theoretical and practical aspects of the Engineering cycle.

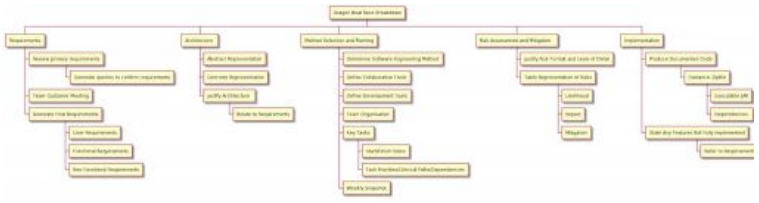
Even though we had assigned dedicated roles, our team members also opted for a more democratic

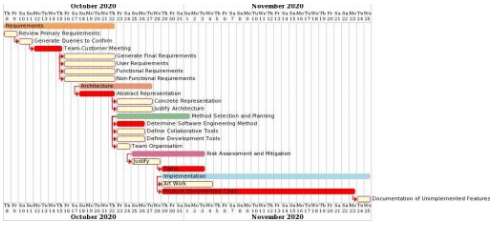
structure, in that when a person saw something was off in accordance to the tasks or risks they had been allocated to complete or monitor, they would raise their concerns when the team reconvened in the weekly meetings.

3 Project Plan - Part C

3.1 Key Tasks

We broke down each section of the project into key tasks using a Work Breakdown Diagram with PlantUML:





In the first Sprint we created an initial Gantt Chart to lay out what needed to be done within a certain time frame and help us better see a critical path and any task dependencies. We felt that a Gantt chart would help support our Scrum methodology by giving us a clear view of what was required for the entire project as well as helping us to break down the tasks for our sprints.

3.2 Task Priorities and Dependencies

Priority tasks are coloured in red in the Gantt chart. We considered each stage of the project to determine which tasks needed to be completed first during a sprint and decided upon the following tasks:

● Team-Customer Meeting - as we would need to know any further information or requirements before we could start proper work on the Architecture as the requirements would define what the software needs to be accomplish

- ● Abstract Representation - as the Development team need a groundwork of what needs to be included in the code and the Design team will also need to consider any artwork or documentation
- ● Determine Software Engineering Method - as a methodology will define how we lay out our project and our time, in our case we chose the Scrum methodology
- ● Table - we believed a risk assessment table was crucial so we can potentially act upon any risks before we got too far into the project or implementation stage

- ● Produce Documented Code - as this was what the client requires and needs to be completed before the deadline, it was set as a high priority

Using this information we also highlighted the task dependencies:

- ● ‘Generate Final Requirements’ is dependent on the ‘Team-Customer Meeting’ finishing, so we can get any extra requirements that weren’t initially mentioned
- ● ‘Abstract Representation’ is dependent on finishing the ‘Generate Final Requirements’ task as the Development team need to know exactly what is required by the code to create their blueprint ● ‘Concrete Representation’ and ‘Justify Architecture’ both rely on finishing the ‘Abstract Representation’ as they are extensions of the ‘Abstract Representation’
- ● ‘Art Work’ depends on finishing the ‘Generate Final Requirements’ task as any additional drawings or the clients design choices need to be considered
- ● ‘Produce Documented Code’ relies on finishing ‘Generate Final Requirements’, ‘Concrete Representation’ and ‘Art Work’ tasks

3.3 Critical Path

A critical path is a sequence of activities that, if they were delayed, then the completion of the entire project would also be delayed. Considering we have a total of 48 days to complete this project, we identified the critical path as:

1. Review Primary Requirements

2. Generate Queries
3. Team-Customer Meeting
4. Abstract Representation
5. Art Work
6. Produce Documented Code
7. Documentation of Unimplemented Features

The ‘Produce Documented Code’ task is the longest task we have to complete, however, it relies heavily on the tasks beforehand (listed above). Therefore, if any of the previous tasks are delayed then so would the ‘Produce Documented Code’ task and this could delay the entire project.

3.4 Updated Plan for Assessment 2

We used Team Gantt to manage tasks and set initial time frames for them so that we stayed on track. This allowed us to communicate frequently even out of scheduled times utilising time that people could communicate at their own convenience, even if asynchronously.

A sprint culminates in the sprint review held every Thursday over Zoom. The separate teams come together to exchange information and review the sprint, stating whether they have been successful or not in completing all the tasks allocated to them in that duration according to the project plan. If not, necessary adjustments are made to the project plan to accommodate the unforeseen circumstances, and the next sprint is planned. Tasks are in one of five columns: Deliverables, To do, In progress, To be reviewed, and Complete. This helped us make sure everyone and everything was on track. Separate Implementation or Documentation meetings take place on a more regular basis, approximately every two days, or individuals of a specific team are at least in regular contact with each other over Discord daily. Having daily standups was not a viable option due to individuals having workloads from other modules as well.

3.5 Discussion on Plan Evolution

Adjustments on task allocation in the project management were made swiftly, depending on the availability of team members and the rate of the progress made on them, which was checked in the weekly meetings. If a task was going too slowly, more team members were allocated to it, or the individual(s) assigned to it spent more time on it if they were comfortable. These stalls in progress were fortunately found quickly due to the regular reconvening of Agile development. As time went on, new tasks that we had initially overlooked and which needed to be complete by the end of the project came to our attention. These were added to the Gantt chart as sprints went by. We also broke down more higher level tasks into more lower level tasks. Initially the higher level tasks had a number of team members assigned, but as these were broken down, individuals were assigned more lower level tasks. As we got more acquainted with the project and realised the tasks which would be dependent upon other tasks, we reflected this knowledge of task dependencies in the Gantt chart. The various iterations of our project plans in the forms of Gantt charts can be found in our Weekly Snapshots here.

Our systematic plan is in the form of a Gantt chart (see last page). It lays out key tasks, as well as task priorities (high priority tasks indicated by red bars, medium priorities by yellow bars, and low priorities by blue bars) and dependencies. The task allocation uses the initials of our teammates, as opposed to using their full names, to make the Gantt chart easy to understand and not overwhelm the onlooker.

Team 15's Project Schedule