Method selection and planning

Group 28

Piazza Panic

By OuseWorks

Ben Harris
Joshua Gill
Niamh Hanratty
Amy Raymond
Matthew Czyzewski
Matt Rohatynskyj

Part A:

Software Engineering Methods:

One of our main priorities has been collaboration, however, because we only have meetings once or twice a week, we need to do lots of work outside of our meetings. Therefore one of our main software engineering methods has been **agile development**. This means that during our meetings, we can focus mainly on discussing what each member has worked on over the past sprint (this could range from a number of days to a number of weeks) and prioritise what needs to be improved or finished and then assign tasks for the next sprint.

We also came up with our own **plan-driven method** which is similar to a small-scale Rational Unified Process (RUD). The method we came up with incorporates lots of documentation. We felt like communication was a main priority within our team so we have tried to document every decision we have made so that we make sure everyone knows what we are doing each week. We have done this by making meeting notes for every meeting and this also means that it is easy to catch up if someone is absent. Each week, we have updated our UML gantt chart and made a list of our priorities and dependencies. This will improve our progress as we are continuously reviewing our plan and making changes where needed. We feel like keeping all this documentation won't distract us from the actual project itself, but instead aid us in writing our deliverables.

Development and Collaboration Tools:

Google Drive

The main tool we have used to collaborate on our deliverable documents for this project is Google Drive [1]. It has been useful for us as it allows us to synchronously work together online whether we are together or working on it at separate times. There is a comment section where members can comment on work done by other members, allowing us to both praise and challenge ideas raised by others during asynchronous work as communication works really well to aid our plan driven method. We used folders within this software to organise our work. This software is good because there are lots of extensions which you can add to chrome, including PlantUML Gizmo.

The alternative to Google Docs could have been to store parts of the deliverables locally and then send the parts to the master copy which could have been on someone's personal device, however we wouldn't have been able to see live updates of what people are doing, making synchronous and asynchronous collaborative work hard to complete. This would have also led to an increased risk of losing all the work, if we didn't have it backed up.

Integrated Development Environment (IDE)

The IDE we used is Visual Studio Code (VS Code) [2]. The alternatives that we could have used are Eclipse [3] and IntelliJ [4] but as we knew we had all worked with VS Code and not the others, we decided to go with the familiar option. This means we didn't have to download new software and saved time learning how to use the development environment. VS Code has documentation online for connecting to Git repositories so we knew we would be able to find libraries easily to help us with the implementation of our game. VS Code allows us to each work on and test our code locally during each sprint before we discuss it during the next meeting.

PlantUML

PlantUML [5] is software that lets you write code for diagrams such as class, package, sequence, state, and component diagrams. You can either use the software directly on the website or download the PlantUML Gizmo google doc extension. This allows you to add the diagram and edit it within google docs itself. However, this doesn't update the diagram in actual time as you type, unlike using the website. For the class diagrams, we used the actual website rather than the extension, as we did these together and projected it to a large screen so everyone could see the diagram update live.

With the gantt chart, we used the google doc extension as this was easier to do on a smaller scale (i.e. one team member can do it). It was easier to duplicate the gantt charts on google docs, to show the evolution of the plan every week, which we regularly looked over to see what to improve and do next during our project. This worked well for the agile development method as everyone had access to the gantt chart so they knew what they would need to complete during the sprint before the next delivery.

An alternative to PlantUML was Lucidchart [6], however this requires a subscription to be able to collaborate on it and to use some of the features. After testing PlantUML after our lecturer recommended it, we decided that it was easier to use, and there isn't a limit for free UML diagrams, unlike with Lucidchart.

Git
Git [7] is one of the most popular platforms for collaborative software development, allowing a team to create projects and work on them on their individual systems before amending the master source code. The software can be interacted with either online or through the Git extensions available on their website and through some IDE's. Git enables the team to have an agile approach, as it requires code repositories to be held in branches. Each team member can be assigned a task to work on in their own branch, allowing everyone to work without disrupting others and breaking the master repository. Along with this, before changes can be made to the main repository, each team member's work needs to be reviewed by other team members. Having taken an agile approach this mitigates the risk of code being introduced which can break the game system and for quicker development to occur.

An alternative to Git which could have been used is SVN [8], but due to a lack of knowledge and understanding of version control systems we chose not to use this system. It is much more suited to power users. Some of the team members had previous experience of using Git and it appeared to be the most widely documented.

LibGDX
Our choice of framework for this project was libGDX [9]. It is a cross-platform based on OpenGL which is an API supported by virtually all consumer technologies i.e. tablets, iPads. As our brief required the game to be cross-platform, using libGDX would mean from the start all code written will work across a range of devices. libGDX has lots of predefined functions to support 2D game development, saving us time having to create these ourselves. Without these predefined functions, we would have struggled to create working iterations of the project due to a lack of in-depth knowledge of the Java language. As the framework is open-source, all team members can download a copy of the software onto their own machines and work on their tasks when they are able to. For the agile development style, this was important as it allows team members to work on the project out of team hours, checking the robustness of their code, without system incompatibility..

An alternative we considered was jMonkeyEngine [10] as it has a more code first approach. Due to a lack of in-depth Java knowledge we chose not to use this as it would create problems for us further down in development. libGDX is also designed to natively support the development of 2D games. This made it the perfect framework for the team.

Tiled
Tiled [11] is the free, open source software we used to create the game map. It is user friendly, meaning it was easy to become familiar with and supported the use of free tilesets we downloaded or created, saving lots of time during the implementation of the game.

<u>Part B:</u>

Within our team, we decided to give everyone roles to stay organised. We found this was an important step to do as we knew the shadow roles in particular would play a significant role in our risk assessment document and would increase the bus factor of our project. The roles we came up with are as follows:

Meeting Chair:	Ben Harris/Joshua Gill
Secretary:	Niamh Hanratty / Amy Raymond (shadow role)
Librarian:	Matthew Czyzewski / Ben Harris
Report editor:	Amy Raymond / Niamh Hanratty (shadow role)
Dynamic Shadow Role:	Matt Rohatynskyj and Joshua Gill

Having a meeting chair is appropriate for this project as they will help keep the meetings organised by making sure we don't get off-topic and stay on track with timings for that meeting. The secretary's role is to document what goes on in the meetings and make notes so that if people are absent, which is likely to happen during this project, they will know what they missed. The librarian will provide and document most of the resources we will use throughout the project. The report editor's job is to keep the documentation in the shared drive organised and to keep the deliverables cohesive and consistent. They can also oversee the formatting of the deliverables, which is a requirement for this project. We then have a shadow role for each of these roles, which will oversee the role's tasks in case the assigned team member is absent or falling behind. We also have two dynamic shadow roles who oversee everyone's tasks and pick up missing or behind work. This approach is appropriate for our team as it is likely we will have absences during this project as peoples' schedules clash, so this is a way to stay organised despite having these setbacks.

In terms of organisation during our original planning stage, we set out to split the work into two-week chunks, having flexible plans at the beginning of each chunk and then reviewing what we've done at the end of each sprint of this agile delivery method. This means we would be able to spread out the workload and wouldn't be rushed to finish it all at the end. Within these two week chunks, we will assign different tasks to different members of the team, so that we can make progress in different areas a lot faster than if we were to tackle each step as a whole team. I think this is appropriate for this project as there are the deliverables which can be our main 'chunks' and then there are smaller tasks within these that we can assign to members.

We have chosen to organise all our documents in folders in a shared google drive [1]. This means it is always easy for us to access the meeting notes and deliverables.

Part C

Evolution of Plan

During our first recorded meeting, 14-11-2022, we assigned everyone roles and set up our initial plan (Week1 on website). Our main priority was to start thinking of questions for our customer meeting which we had scheduled. We planned to start the planning document the next week, after our customer meeting, and then when we finished that we would start our risk assessment and mitigation document.

The next week, we improved by starting to look at the project as a whole, instead of just week by week, so that we could improve on the timings. We ended up starting the planning document and risk assessment the week before so we updated this on our gantt chart (Week2 on website). We gave ourselves the deadline of 02-12-2022 to complete the requirements tables and risk management document. We started the CRC cards and explored game engines as a group.

During the third week we decided that the website wasn’t a priority anymore as it isn’t worth many marks. After we initially set it up, we can put a pause on developing it until closer to the deadline. The UML diagrams for the architecture document became our new top priority as we needed first drafts of our diagrams before we started our implementation. We updated our gantt chart to show our new plan (Week3 on website).

The next iteration of our plan (Week4 on website) was made after the summer term ended. We decided to give ourselves longer to do tasks as it was harder for us to do synchronous collaborative work whilst not being able to do meetings in person. We extended the time period for us to complete the architecture diagram as we realised this would evolve during the process of implementation.

Our plan then evolved again during week 5 as we realised that the Method Selection and Planning document could be split into two sections. Parts A and B could be finished as soon as we knew what software and methods were using, whereas part C is to be added to over the entire project. Therefore we could set a deadline for parts A and B to be finished, and assign this to two members of the team to do. We added this to our gantt chart (Week 5 on website).

During the week of 19-12-2022, our plan didn’t change much as this was Christmas. We pushed back the start date of implementation, until more UML diagrams had been made. Instead, we allowed members of the team to just add to documents when they had time. We updated this on the gantt chart (Week6 on website).

Up until this point, most of our sprints had been a week long, however our next sprint covered the next three weeks (Weeks7,8,9 on website) up until the start of the winter term. Not much of the large-scale plan had changed. We started the implementation of our game and got the basics coded, evolving the architecture as we went. Many aspects of our plan remained the same as we still continued to update the Method Selection and Planning Document and did final edits of the User Requirements document.

During week 10, we split into two groups of three. One covered implementation and one covered architecture, instead of everyone trying to work on both tasks like we had done for the previous sprint. We realised this worked better and we made progress at a faster rate. We also originally planned to find assets online, however after searching for them we found that to keep consistency between our ‘objects’ and background, we would need to make some of the ‘objects’ ourselves. We updated our lists of dependencies and priorities and added this to the plan (Week10 on website). This included one member of our team doing more work on the website.

We have combined two weeks for this sprint as the second week only has 3 days before the deadline. During week 11 we worked on finishing the last parts of our implementation and architecture, which were our main priorities (Week11,12 on website). We are focusing on testing our game and making changes and improvements where we find errors. Although the architecture document has been started weeks ago using first versions of our architecture diagrams, a lot of it depends on the final version being made. Once the implementation and testing is finished, we will finish finalising all of our documents and putting them onto the website.

As our project progressed, we moved from working more independently and asynchronously to working collaboratively, in order to maximise productivity. This arose when meeting deadlines set by the group became more difficult, approaching the deadline.

References:

[1] *Personal Cloud Storage & File sharing platform* (no date) *Google*. Google. Available at: https://www.google.com/drive/.

[2] Microsoft (2021) *Visual studio code - code editing. redefined*, *RSS*. Microsoft. Available at: https://code.visualstudio.com/.

[3] Guindon, C. (no date) *The community for Open Innovation and Collaboration: The eclipse foundation*, *Eclipse Foundation*. Available at: https://www.eclipse.org/.

[4] *IntelliJ IDEA – the leading Java and Kotlin Ide* (no date) *JetBrains*. Available at: https://www.jetbrains.com/idea/.

[5] *Open-source tool that uses simple textual descriptions to draw beautiful UML diagrams.* (no date) *PlantUML.com*. Available at: https://plantuml.com/.

[6] *Intelligent diagramming* (no date) *Lucidchart*. Available at: https://www.lucidchart.com/pages/.

[7] (no date) *Git*. Available at: https://git-scm.com/.

[8] *Home · tortoisesvn* (no date) *Home ·*. Available at: https://tortoisesvn.net/.

[9] *LibGDX* (2022) *libGDX*. Available at: https://libgdx.com/.

[10] Authorsstephengold, Authors and Authorsriccardobl (no date) *JMonkeyEngine*, *jMonkeyEngine*. Available at: https://jmonkeyengine.org/.

[11] (no date) *Tiled*. Available at: https://www.mapeditor.org/.