Team iPatch

Assessment 1: Method Selection and Planning

Christian Pardillo Laursen, Filip Makosza, Joseph Leigh, Josh Wakefield, Tina Weng & Oliver Relph

Software Engineering Methods

Our group is composed of only six people, so we decided to find a methodology which best fit small groups. We looked to industry to find the most common and praised development methodologies, and found that an agile Scrum-like approach would likely be most suitable for our team. A survey by the Scrum Alliance found that 42% of surveyed development teams use Scrum exclusively, and that the average Scrum team size is seven people.[1] The prevalence of Scrum in teams this size suggests that it is a worthwhile methodology to consider first.

One of the stated advantages of Scrum is its flexibility, rapidly iterating on working prototypes to get customer feedback. We were given only 3 months to finish the software project, a relatively strict deadline which would benefit from incremental progress to ensure that we know that we have a functional product as early as possible. As stated in the Scrum Alliance report, one of the top reasons for using the methodology was the focus on meeting budget, time, and scope constraints.[1] The iterative and modular development style encouraged by Scrum is optimal for identifying the general amount of time it will take to complete components of a project, as well as avoiding feature creep by segmenting the project up into specific tasks of varying priorities. If the customer changes their mind and requests a feature change mid-development, the agile Scrum approach should minimise the work that needs to be done to integrate the changes.

We looked to the other agile methodologies to see how they compare to Scrum, and found that the Kanban approach to requirements fit our team a bit better. Keeping the to-do list fixed for the duration of a sprint is probably good for full-time development groups, but it would be too restrictive to properly adhere to for our group. Since the time we can dedicate to programming can vary from week to week, we decided that the relaxed Kanban approach [2] to artifact management would be better suited for us. We also found the less rigid Kanban team structure more appealing as it allows team members to fluidly assume different roles depending on circumstances, giving the team greater flexibility if any members are unavailable.

We also considered more traditional waterfall-like methodologies like IBM RUP [3], but ultimately decided that the rigid and bureaucratic structure of such a solution would be too cumbersome for a relatively small and constrained project like ours. The focus on comprehensive documentation and preplanning of all aspects of the project would make it difficult to adapt to requirement changes and would require a lot of prior knowledge of the project technologies to minimise mid-development changes to the project. We considered other agile methodologies as well, but ultimately the differences between solutions like Scrum, Kanban, and Extreme Programming were not significant enough for us to stick to one methodology rigidly. We settled on a mix of different agile methods that best fit our team, leaning mostly towards Kanban.

Tools

In order to carry out development of the project effectively despite not spending the majority of development time in the same physical location, we had to employ several tools to facilitate easy communication, collaboration, and sharing of both code and ideas with each other.

Git

We picked Git as our version control system of choice. The decentralised nature of Git means that team members can spend long amounts of time working on a particular feature without having to commit half-working code to the main repository, also being able to use Git's features locally if they

have no internet access. Git is one of the most popular version control systems in the world, so there is plenty of online help and documentation as well as other support for the software, including IDE integration and graphical clients. We believe this is the best choice for us as we will often be making changes simultaneously and so the decentralised nature is very helpful when this is the case. In addition to the decentralisation, Git supports branching which will be very useful during development when multiple features will be being developed without needing to affect the master branch.

GitHub
GitHub will not only allow us to keep our repositories for the project on a remote server, as it also provides access to project boards (which allow for more flexibility in planning our Scrum sprints) and web hosting for our project website. It is also free and easily accessible online so it helps in allowing the whole team to access the repositories easily and quickly.

Google Docs
We decided to use Google Docs to create and store non-code files and documents related to the project. Google Docs makes it easy to collaborate with other team members, allowing multiple people to edit the same file simultaneously through a web browser. Google Drive makes it easy to share and store files in a place which is easy to access for all team members, implementing its own rudimentary version control system to easily revert significant mistakes or damage to any project files. In addition, compatibility issues can be avoided, which could result from different word processing applications being used across different systems, as Google Docs is accessible across all systems.

Discord
We picked Discord as our instant communication method. Discord makes it easy to talk to other team members both through text and through multi-user voice meetings, and allows team members to set up topic-specific channels and share various files directly. It is free to use and available on all major desktop and mobile operating systems, as well as directly through a browser. This allows our team members to easily see new and important messages; helping in our aim to minimise time lost which is important for a small development window.

ArgoUML
To produce UML diagrams we chose ArgoUML, free and open source cross-platform UML diagram creation software. ArgoUML is a stable and mature piece of software which is available on Windows, OSX, and Linux, and produces graphs of a standard UML format which can be saved and opened in other standards-compliant UML editors.

jMonkeyEngine
We picked jMonkeyEngine as the game engine to develop the project in. Our initial preference was for Unity, but we found out that we are restricted to Java libraries. jMonkeyEngine is a free and open source cross-platform game engine, and has the most comprehensive documentation of any Java game development framework we could find. It is focused around 3D development and comes with a custom SDK which makes it easier to manage larger projects than with libraries like libGDX. It also comes with various tools like Git and Blender integration, and a 3D scene editor built into the SDK which can be used to graphically compose game scenes and generate simple terrain maps.
jMonkeyEngine is one of the most comprehensive and mature game development frameworks available to Java programmers, which is why we chose to use it as our game engine.

Team Organisation

In order to function coherently and efficiently as a team, we needed to understand the strengths of each team member so that everyone can contribute effectively. Initial roles were assigned to team members: meeting chair, secretary, librarian, report editor. The position of meeting chair was given to Christian Pardillo Laursen, Filip Makosza was appointed secretary, Mingxuan Weng was given librarian, and the position of report editor was assigned to Josh Wakefield. The remaining team members were able to assist freely if necessary. These roles were allocated in regards to documentation, meeting memos, and paperwork, we chose a far more freeform approach for our development structure.

Scrum suggests a more organised approach, where a Scrum Master is responsible for organising and managing sprints and development. This approach can be tricky to get working in a team of students where not every team member can be available at set times of the week, so we settled for a more freeform structure as allowed by Kanban. Chris, the meeting chair, assumes the more relaxed role of manager within our group, with the goal of making sure the project has focus and team members are working on relevant code.

Because of the varied nature of game development, we chose to have informal team roles to ensure members can stick with writing code they're comfortable working with. Due to the flexible nature of the approach we wanted to take, team members are allowed to change roles throughout the development process. To achieve maximum productivity at different stages and given the strengths of the individual, it may be necessary to shuffle roles and discuss strengths and weaknesses with the rest of the team to avoid gaps in development.

Because of the modular nature of the project, the relaxed team structure paired with a Kanban approach to development would be optimal for effectively allocating time between team members, without the disturbances absences at regular meetings would cause. The separation of project objectives into achievable and self-contained artifacts will make it easier to visualise progress and see which parts of the project still need work, as well as which artifacts are high-priority and serve as a dependency for other artifacts. The team can organise around the Kanban board and discuss any organisation issues through our communication tools, which should make it easier to coordinate the team and ensure important elements are finished.

Plan

Due to taking the Kanban approach, we will have a continuous workflow. Therefore, we will not have a regular schedule so it is acceptable if tasks are completed earlier or later than others as long as they are completed before the deadline.

During one of the first meetings regarding Assessment 2, we will divide the software project into artifacts which can then be put onto the Kanban board. By the end of the meeting, we will have a comprehensive list of artifacts required to be completed in order to finish the program, with an appropriate level of abstraction for the earliest stages of development. If at any point an artifact is discovered to be too broad or too specific to be implemented comfortably, the team will have an informal discussion regarding the artifact and possible ways to resolve the issue.

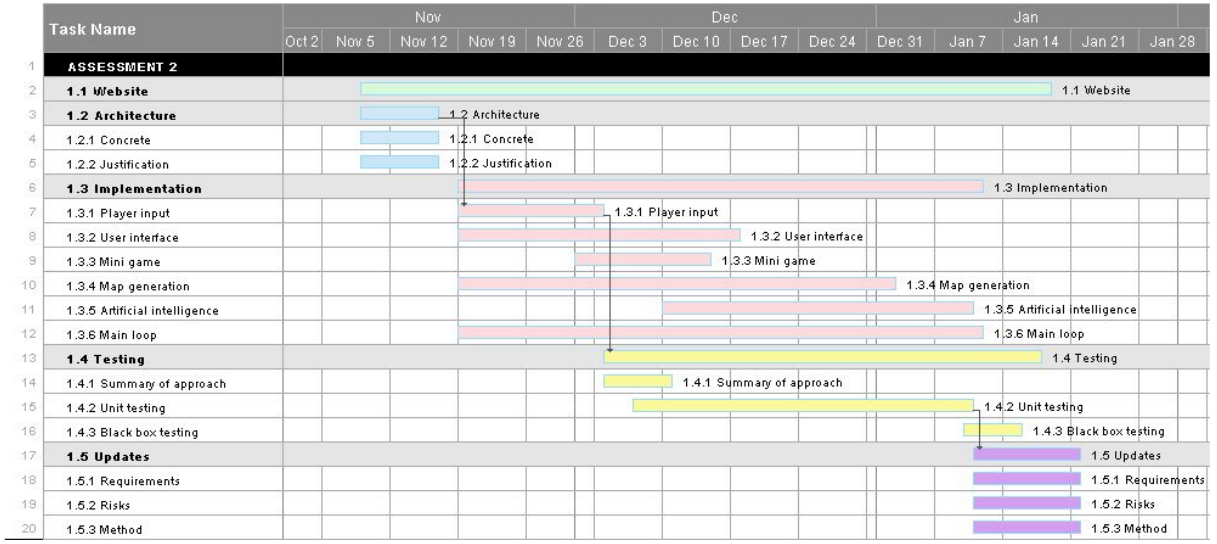
Kanban does not specify anything like a Scrum Sprint and it does not require regular meetings, but the group will organise semi-formal meetings at least once every two weeks, where group members will discuss their progress, any issues they've encountered, and priorities to focus on over the

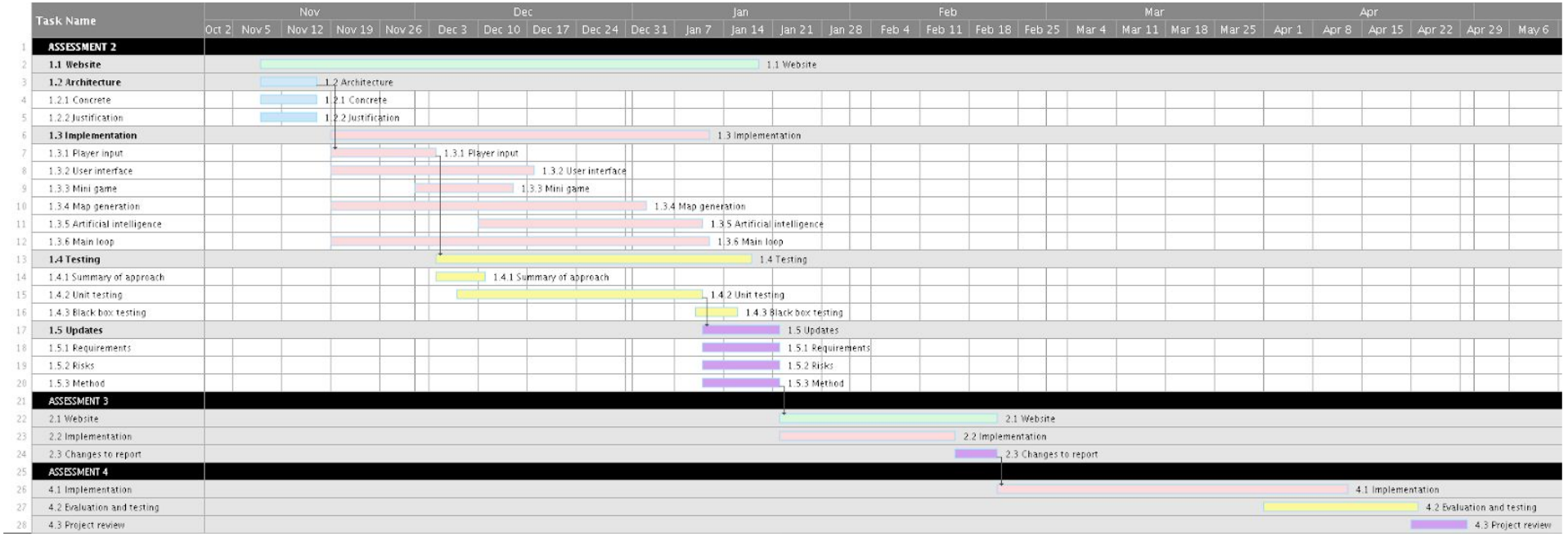
coming weeks. This meeting will not formally set out objectives for the week, but rather exist as a way for the group to be kept up to speed regarding the work of the other members, as well as a convenient opportunity to refocus development on any high-priority issues and resolve particularly difficult problems together, or reorganise development in case of changes in requirements. Development will otherwise focus around the Kanban board, where members can suggest new artifacts or change the priorities of existing artifacts if the need arises.

As shown in the Gantt chart, development of assessment 2 will take place from 9/11/2018 to 18/01/2019. The architecture will be completed first in order to prepare and plan for the implementation. The largest period of development for assessment 2 will be the implementation, consisting of coding specific sections of the game. As soon as an artifact is finished, it will be flagged for review. It will then be checked for code quality and readability. We will also be testing in parallel to implementing to ensure that the code we are building upon is correct and functioning correctly. Finally, towards the end of development once all previous tasks have been completed, we will update the deliverables from assessment 1 so that they are consistent with recent changes.

In regards to assessments 3 and 4, the exact details will have to be planned out once we are aware of the project we will be adopting. As shown in the Gantt chart, however, we have planned out a rough time frame for various tasks we expect to have to carry out. Because of the brief time frame given for assessment 3, we estimated approximately three weeks would be dedicated to implementing the desired extended requirements, and a week to documenting the implementation, how it satisfies the requirements, the technical details, and an explanation as to what the new requirements were.
Assessment 4 has a longer deadline, so we committed about a month and a half to implementing the new requirements. At least three weeks, some in parallel to the implementation phase, are going to be used for evaluation and testing, and writing the corresponding report. The last week before the deadline will be used to write the review report.

The Assessment 2 Gantt chart accompanies this file as gantt1.png. The Assessment 2+3+4 Gantt chart accompanies this file as gantt2.jpeg.


Task Name
Nov
Dec
Jan
Oct 2
Nov 5
Nov 12
Nov 19
Nov 26
Dec 3
Dec 10
Dec 17
Dec 24
Dec 31
Jan 7
Jan 14
Jan 21
Jan 28
1 ASSESSMENT 2
2 1.1 Website
3 1.2 Architecture
4 1.2.1 Concrete
5 1.2.2 Justification
6 1.3 Implementation
7 1.3.1 Player input
8 1.3.2 User interface
9 1.3.3 Mini game
10 1.3.4 Map generation
11 1.3.5 Artificial intelligence
12 1.3.6 Main loop
13 1.4 Testing
14 1.4.1 Summary of approach
15 1.4.2 Unit testing
16 1.4.3 Black box testing
17 1.5 Updates
18 1.5.1 Requirements
19 1.5.2 Risks
20 1.5.3 Method


Task Name
Nov
Dec
Jan
Feb
Mar
Apr
Oct 2
Nov 5
Nov 12
Nov 19
Nov 26
Dec 3
Dec 10
Dec 17
Dec 24
Dec 31
Jan 7
Jan 14
Jan 21
Jan 28
Feb 4
Feb 11
Feb 18
Feb 25
Mar 4
Mar 11
Mar 18
Mar 25
Apr 1
Apr 8
Apr 15
Apr 22
Apr 29
May 6
1 ASSESSMENT 2
2 1.1 Website
3 1.2 Architecture
4 1.2.1 Concrete
5 1.2.2 Justification
6 1.3 Implementation
7 1.3.1 Player input
8 1.3.2 User interface
9 1.3.3 Mini game
10 1.3.4 Map generation
11 1.3.5 Artificial intelligence
12 1.3.6 Main loop
13 1.4 Testing
14 1.4.1 Summary of approach
15 1.4.2 Unit testing
16 1.4.3 Black box testing
17 1.5 Updates
18 1.5.1 Requirements
19 1.5.2 Risks
20 1.5.3 Method
21 ASSESSMENT 3
22 2.1 Website
23 2.2 Implementation
24 2.3 Changes to report
25 ASSESSMENT 4
26 4.1 Implementation
27 4.2 Evaluation and testing
28 4.3 Project review

References

[1] “The 2015 State of Scrum Report,” Scrum Alliance, Jul. 2015 [Online]. Available: https://www.scrumalliance.org/ScrumRedesignDEVSite/media/scrumalliancemedia/files%20and%20pdfs/state%20of%20scrum/scrum-alliance-state-of-scrum-2015.pdf

[2] M. Rehkopf, “Kanban vs. Scrum,” *Attlassian*. [Online]. Available: https://www.atlassian.com/agile/kanban/kanban-vs-scrum. [Accessed: 26-Oct-2018]

[3] “Rational Unified Process - Best Practices for Software Development Teams,” Rational Software, 1998 [Online]. Available: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf