

SEPR

SPACEKEY PROJECTS

Assessment 1

Methods

JORDAN CHARLES
SAMUEL HUTCHINGS
CHLOE HODGSON
GOLNAR KAVIANI
TAMOUR ATLAF
JACK THOO-TINSLEY

2019/2020

Method Selection and Planning

a) Software Engineering Methods

The importance of selecting the correct software engineering methodology for this project was recognised early on by SpaceKey Projects. By looking at the general nature of the project, and analysing the forthcoming assessment details in the brief, the following needs were derived that the methodology must satisfy, or contribute to:

- Team culture of collaboration and comfortable communication, due to the small size and overlapping skillset of our team.
- Reduce the cost of change and allow a flexible approach to development. We want to not only be able to adapt to a client's altering ideas but welcome them, in order to produce a higher quality product.
- Clear view of how an individual's work and actions impact the progress of the whole team.
- A focus on readable, working and tested code over initial heavy amounts of documentation. This will allow us to learn from failure more quickly, as well as ensuring that our system is easy for other teams to adopt in later assessments.

To begin, we compared plan-driven and agile methods. The plan-driven models that we investigated the most were 'Waterfall' and 'V-Model'. Waterfall is an example of an incremental model, where "the project is broken down into a series of builds with their own lifecycle"[1] and "each phase of development proceeds in order without any overlapping" [2]. It was decided that the overheads created during the intensive early planning stages of this model would impact our level of adaptability too much, and so it was rejected in favour of a more flexible approach. The V-Model is an adaptation of the Waterfall model, the difference being that "the stages turn back upwards after the coding phase is done"[2]. We appreciated the emphasis on thorough testing that operating under this model would bring, however we felt that the lack of task overlapping would hinder our progress too much due to the close deadline - it also inherited many of the adaptability issues from the Waterfall model.

We found more success when researching agile methodologies. A couple of members of our team have experienced using agile methods during internships, and so we predicted that this would be the outcome, however we still felt it necessary to research plan-driven methods as seen above. Agile methods fit the criteria which we had outlined; "huge emphasis on collaboration with the customer", "thinking in terms of self-empowered and self-motivated teams", "Working product is the primary measure of progress"[4].

In order to implement an Agile methodology, we will use a SCRUM framework - "embracing uncertainty and creativity" [3]. SCRUM revolves around a central product backlog of high-level tasks which "evolves with the project". Tasks are sorted by priority and an "estimation of effort" is assigned to each task. It is important that these estimations are decided as a team. Work will be completed in two-week long 'sprints', in which tasks are selected from the backlog to be worked on. Work is visible - each task's progress can be tracked by each team member on the 'SCRUM board'. In the board, tasks are sorted into 'to do', 'in progress' and 'done'. Our team has also chosen to add a 'review needed' column, which is used when an individual has completed a task but feels that validation is needed

from the whole group before it can be considered to be 'done'. Each sprint is reviewed upon finish to create a retrospective. This will inform future sprints and help us to learn from our weaknesses, both individually and as a team. We will be consistent in our approach to the sprint planning and sprint review meetings, operating on a bi-weekly basis as we believe this will be enough time to make significant progress on most tasks. Due to us having to fit the project around other studies, we will not complete daily "stand-ups" as many SCRUM teams do, but instead operate these on a frequency dependant on the current weight of work.

Tools Overview

Jira - Project Management & Task Organisation:

Jira makes the use of the SCRUM framework easily manageable and operational. For example, it allows us to easily create a collaborative backlog, 'SCRUM board' for each sprint, assign effort estimations and assign tasks. Our workflow will revolve around our Jira project, and it will play a large part in our team's organisation. Before beginning the system development, each team member was given a tutorial/introduction to the software, in order to ensure that everyone sufficiently understands its functionality and role in our workflow.

GitHub - Version Control:

Initially, we were going to use BitBucket for this purpose, as it is easily interfaceable with Jira as they are both Atlassian products. However, we have chosen to use GitHub as we feel that the advantage of being able to easily link and update our website, outweighs BitBucket's Jira integration. We also feared that excessively linking out task management with our version control may over-complicate our approach and create a "spaghetti"-like situation if not fully devoted to. GitHub, like alternatives, allows multiple team members to easily contribute to the project development by branching, cloning and merging work.

Slack - Communication:

In order to avoid mixing personal use and project planning, we will use Slack for messaging outside of meetings. We feel this is a better option than using a casual application, such as WhatsApp or Facebook Messenger. Within Slack, different channels can be used to separate topics of conversation, which will help to ensure that no messages are lost from having one main thread. Slack is also easy to use on both desktop and mobile.

LucidChart - Flowchart & UML:

LucidChart is an industry recognised web app for creating graphic designs. The software supports the creation of both flowcharts and UML diagrams thanks to its many available templates as well as a wide variety of shapes and symbols for use. Because of this, it is more than suitable for our design requirements.

Google Docs / Google Drive - Documentation:

Our written reports, product briefs and other text-based documentation will be created/stored on our shared Google drive. The real-time nature of Docs is very useful for collaboration; allowing multiple team members to work together from different devices. It is useful to have everything stored in one place using a drive. The 'review changes' feature is good for version control, as well as being able to leave comments on each other's writing.

b) Team Organisation

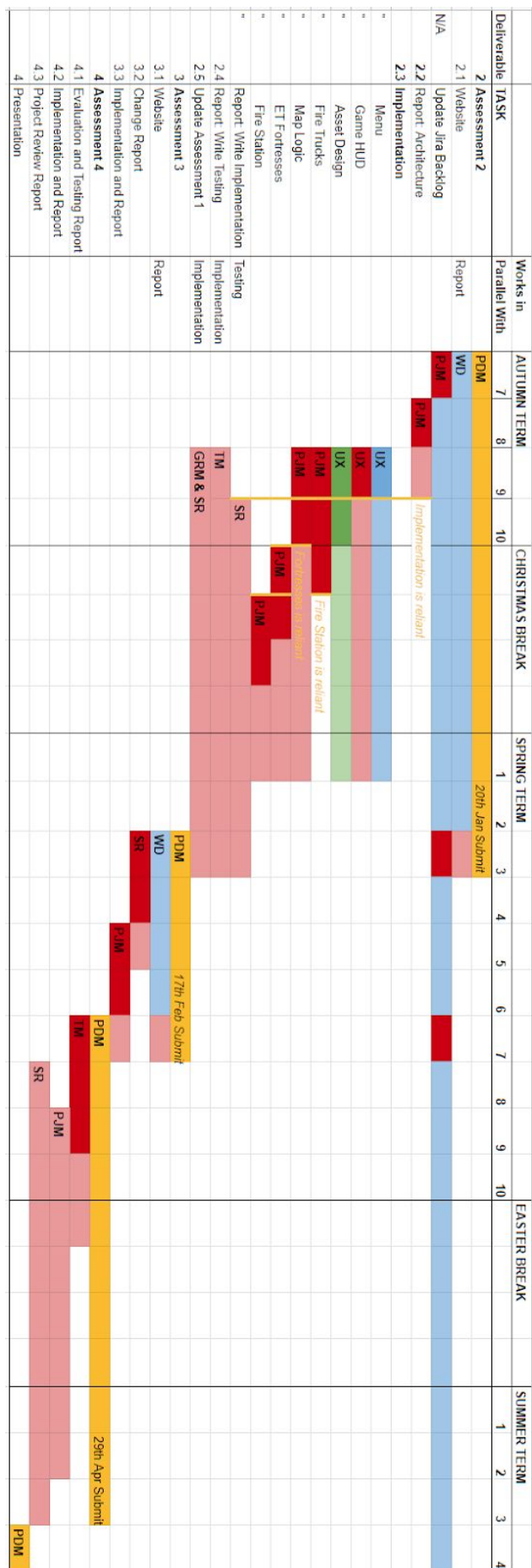
We approached our team organisation rather casually at first - we felt that an informal conversation about our preferred working styles would be more beneficial than taking personality tests or similar. We each made references to the roles that we had individually taken on in previous group projects, such as first year modules and sports teams. This also served as a good ice breaker. Proof of this process' effectiveness was our success in agreeing on, and bonding over, the creation of our team name: SpaceKey Projects. Ian Sommerville's book 'Software Engineering' proved as a guide for us, and stresses the importance of choosing a team name inclusively; taking the first steps towards creating a team identity and forming a "cohesive group"[5].

To a certain extent, we aim to maintain this informal approach throughout the organisation of our project, our decision is supported by Sommerville, "Agile development teams are always informal groups". However, we do recognise the need to loosely appoint a number of roles and responsibilities to members, in order to avoid a lack of strategy. The team will remain cross-functional as "the best agile teams" should be [3]. The roles were decided as a group and appointed based on a number of factors, including each individual's self-assessed technical experience, organisation level and enjoyment. We thought it to be important that we appoint roles not on skill level alone, and that each team member has a particular interest in their role as to promote cohesiveness and performance.

At the current stage of assessment, the roles include, but are not limited to:

- **Project Manager (PJM)**
Encourage group cohesiveness, keep track of critical decisions, act as "SCRUM master", manage project/code development
- **Product Manager (PDM)**
Monitor product success throughout the entire lifecycle regarding customer needs
- **General Risk Manager (GRM)**
Monitor risk mitigation throughout implementation [see *Risk Section*]
- **Secretary and Reviewer (SR)**
Ensures records are kept, that the report sections are consistent with each other and that there is good communication between roles
- **UI and Graphics Designer (x2) (UX)**
Design and create assets and GUI for the system
- **Web Designer (WD)**
Ensure website is up to date and functional
- **Test Manager (TM)**
Ensures system testing is thorough and completed to a high standard

The roles themselves were derived from the assessment brief, and predicted game-development requirements. The roles are fluid, meaning that one person may be responsible for multiple roles at one time; retire from/switch a role during the course of the project and new roles may be introduced as needed (reviewed case by case by the whole team). Everyone will contribute to code development. Our sprint-planning and retrospective creation forms a large part of our organisation - see "Software Engineering Methods" above.



c) Plan for the rest of the project

The Gantt Chart to the left shows our systematic plan for the rest of the project.

A tasks earliest starting and finishing dates are shown by the lighter block colour corresponding to each task. Within this, the darker colour shows an estimate for the length of time that this task should take - but this can be done anytime within the lighter colour. The exception to this are the yellow assessment bars, which are included simply to make the assessments easier to overview as a whole.

The task priorities are shown by the following colours:

- **Red** - This must be done well / it is a deliverable
- **Blue** - This should be done to a high standard
- **Green** - This may be done but expense can be spared for other tasks' sake if needed

A critical path can be followed by the darker coloured blocks.

Task dependencies are shown by thin yellow bars which link the end and beginning of the dependant tasks - an explanation is also given in yellow text for each case. Tasks with no darker colour means that they are ongoing, or revolve around another task. In this case, an estimate is not given as they work in parallel with multiple other subtasks. An explanation for each case is given in the "works in parallel with" column.

The group member/role responsible for each task is shown at the beginning of each task's colour block. A key to these can be found on the page above.

[This chart is also included on our website as Figure 1, under Assessment 1 so that it can be viewed in more detail if required]

REFERENCES

- [1]"Plan-driven software development - Wikiversity", En.wikiversity.org, 2019. [Online]. Available: https://en.wikiversity.org/wiki/Plan-driven_software_development. [Accessed: 08-Nov- 2019].
- [2]S. Balaji and M. Sundararajan Murugaiyan, "WATERFALL Vs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC", International Journal of Information Technology and Business Management, vol. 2, no. 1, 2012. [Accessed 8 November 2019].
- [3]J. Sutherland, Scrum the art of doing twice the work in half the time, 1st ed. New York: Crown Business, 2014.
- [4]M. Moreira, M. Lester and S. Holzner, Agile For Dummies, CA Technologies Edition, 1st ed. Hoboken: Wiley Publishing, Inc., 2010.
- [5]I. Sommerville, Software engineering. Boston [i pozostałe]: Pearson, 2018.