SEPR

SpaceKey Projects

Assessment 2

Updated Assessment 1
Deliverables

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

2019/2020

5. Updates on Assessment 1 Deliverables

In all of the updated deliverables the following key is followed, in order to show complete traceability and transparency of updates made:

Black - Unchanged text
Red - New text
Blue - Removed text

a) Updated Requirements

i.

Assessment 1 Requirements

Assessment 2 Requirements

ii.

Several changes were made to the statement of requirements. In order to fit the page limit restrictions, the constraints table as well as two functional requirements (FR_ENGINE_SPECS and FR_ENEMY_HEALTH) have been removed from the document as these were unchanged - they can still be found in the assessment 1 deliverable.

One additional user requirement was added as it was realised that it was forgotten from the original report, this is UR_FORTSHOOT. This UR is crucial requirement for the gameplay and so was added with *Shall* priority.

An *Assumptions* column was added to the FR and NFR tables in order to make any assumptions made by the team clearer for each requirement. Previously, this was merged with the *Risk* column but it was decided that this could be confusing so it is better to distinguish between them.

Several FRs, for example FR_GOALS, have been relocated to NFRs were it has been realised that they are more suitable as they describe the criteria for the game and so are non-functional; not functional.

More detail was added to several FRs so that they are less vague. An example of this is including the rate of health regeneration in FR_REGEN. All requirements should now have an appropriate level of specifics. The detail of the fit criteria has also been improved, such as the specified standards in NFR_MAINTAINABILITY.

b) Methods, Plans update

i.

[Assessment 1 Method and Plan]

[Assessment 2 Method and Plan]

ii.

Software Engineering Methods

No changes were made to this section of the documentation due to its proven adequacy in providing our team with a methodology outline which has resulted in our team's current engineering approach. Due to the lack of change, this section of the report has been omitted in the updated version as to ensure that the page limits were obeyed in this deliverable. This omitted information is still very much applicable to the project but has been simply been removed for clarity's sake – it can still be accessed by viewing the original Assessment 1 deliverable. This page conservation is done as advised by Dimitris.

Tools Overview

Where none were supplied earlier, all tools now have an alternate described and discounted so that the reader can fully understand the choices made.

Two additional tools are now described: LibGDX for our game development framework, and Eclipse as our chosen Integrated Development Environment. It was realised that these crucial tools were mistakenly not mentioned in the earlier deliverable.

Team Organisation

Clarity on the appointment of group roles was added as it was realised that it was not obvious how roles were literally assigned. Previously, information was only given on the criteria used.

We believe that the roles themselves have proved suitable and so do not require any change.

Project Plan

The Gannt chart used for deliverable planning has been updated to include more detail for phases 3 and 4 of the project. This will allow us to monitor our progress in the next assessments and contribute to an even workflow. The same method of showing start/finish dates, tasks, dependencies etc was followed as in the previous version. Details of this are described in the *methods and planning* document.

c) Risk Assessment and Mitigation Update

i.

[**Assessment 1 Risk**]

[**Assessment 2 Risk**]

ii. [tldr: one update made]

The majority of the risk assessment document was left unchanged due to agreement of the Risk Owners that the previous mitigation was, for the most part, satisfactory. This decision was made during the weekly meetings in which risks were monitored in parallel with our development – obviously, these meetings will continue into Phase 3 and 4 of the project. It was also decided that there would be no need to change the current risk owners as these again were satisfactory.

The exception to this being the addition of one new risk, identified as PJ13 and highlighted in the new risk documentation. This risk was appended due to agreement in the risk meetings that it was having a somewhat significant impact on the teams' productivity and individual relationships within the team. As members were repeatedly late to scheduled meetings, the punctual members felt it showed a lack of respect and dedication to the project which impacted our ability to work together. It also meant that planned tasks for meetings either must be hurried when there is a hard finish time, or that meetings finished a lot later than agreed in order to get everything done. Because of this, the risk was assigned a 'Moderate' severity rating. Mitigation methods mean that the likelihood of this risk is saved from being 'High' category.

Requirements

a) **Elicitation & Negotiation Justification**

We used a combination of “creativity” and “survey” [1] based techniques in order to elicit our requirements. We began by brainstorming based on the information on the assessment documentation/brief. The brainstorming was documented by a moderator figure, and no comments were made initially. Afterwards, the requirements generated were analysed and refined. Once we had a base idea of what was required from the system, we organised an interview with the customer and asked a series of unleading and unbiased questions in order to clear up any areas which we were uncertain about. The results of this survey were used to generate the remaining requirements and adapt others to better fit the customer’s desire. We felt that by using a union of elicitation techniques, we would derive requirements to a high standard of completeness [1].

Below are our elicited requirements. The tabular presentation makes each requirement easily distinguishable and readable. By using unique requirement identifiers, a high level of traceability is maintained so that relationships between requirements can be comfortably understood. The unique IDs follow the format REQUIREMENTTYPE_NAME, for example FR_MAP. Traceability between requirements is integral to identifying cause and effect of failure (maintenance), accountability and impact analysis during change management [1].

Our requirements take the following structure:

- **User Requirements**:

Table consists of the ID, the requirement itself, and the priority. Priority is selected from *may, should, shall* and is used to influence task precedence later in the system development, as well as for testing. The requirements are written in nontechnical language, and map out the high-level tasks that the user may be able to carry out with the finished system.It is important to remember that our primary user-base is *open day visitors* - this is vital to the forming of requirements, for example when considering the game session length.

- **System Requirements:**

Derived from the URs and describes the operations that the system must carry out in order to achieve the URs, split into:

  - **Functional Requirements:**

For traceability’s sake, each FR has at least one corresponding UR which it works towards. Also described are any assumptions or risks to be kept in mind when creating tasks, as well as a requirement type from *transformation, invariant, failure*. Including FR types helps to ensure thorough eliciting.

  - **Non-Functional Requirements:**

These describe the qualities that the system must have upon completion. We considered a range of NFR topics. This range of topics helps to ensure that we are rigorous when reviewing all aspects of the prospective system.

- **Constraints:**

Describe the predefined factors which limit the global system, these are to be kept in mind during requirement elicitation and throughout development.

b) Statement of Requirements

SSON: "The game shall allow a player to save York from intruding, water-fearing aliens, by tactically positioning defensive fire trucks at key locations within the city."

USER REQUIREMENTS		
ID	**Requirement**	**Priority**
UR_ENJOY	The system shall offer an enjoyable user experience	Shall
UR_AESTHETIC	The system GUI shall be clean and easy to look at	Shall
UR_INFO	The system shall present all necessary information to the player through the GUI	Shall
UR_ TUTORIAL	The system shall be intuitive to understand	Shall
UR_SHOOT	The system shall allow the user to attack the ET fortresses and patrols with water from the fire engines	Shall
UR_FORTSHOOT	The user's fire engines should be attacked by the fortresses	Shall
UR_HEALTH	The system shall restore the health of any damaged fire engines that the user has returned to the fire station	Shall
UR_REFILL	The system shall restore the water level of any fire engines that the user has returned to the fire station	Shall
UR_DRIVE	The system shall allow the user to move the fire engines around the map	Shall
UR_DIFFICULTY	The difficulty of flooding the ET fortresses shall increase with time	Should
UR_END	The system shall award the user a victory or lose state when they have flooded all of the ET fortresses, or been defeated themselves, respectively.	Shall
UR_MINIGAME	The system shall offer a minigame unrelated to the main game but in the same theme *[see Minigame URs in website]*	Shall
UR_UPDATES	The system should be able to be updated in the future with new functionality	Should

	FUNCTIONAL REQUIREMENTS				
ID	**Corresponding UR Req. ID**	**Description**	**Assumptions**	**Risks**	**Req.Type**
FR_GOALS	UR_ENJOY	To make the game fun, it will have the goal of destroying fortresses	If a game is confusing it is not fun	There might be too many goals for the user to focus on and the game stops being fun	Invariant

FR_VARIATION	UR_ENJOY	To make the game fun, the fortresses will have different attacks	If a game is confusing it is not fun	The attacks might be too complicated and confuse the player	Invariant
FR_MAP	UR_AESTHETIC	The map will be a simplified version of York city centre	If a game does not provide enough challenge then it is not fun	The map could be too simple to be fun, impacting UR_ENJOY	Invariant
FR_GUI	UR_AESTHETIC UR_INFO	The GUI will be simple and not take up more than 25% of the screen	A game is not easy to look at if it is too busy or if information is too small	The GUI could be too small and cluttered, or too big and take up too much screen space	Invariant
FR_SHOOT	UR_SHOOT	The fire engines will shoot water in the direction of driving and the water will continue to travel for 2 seconds		The range of the fire engines might be too small if the distance is realistic, making it difficult to get into range If the water travels for too long then the map may become cluttered and impact UR_ENJOY	Transformation
FR_FORTSHOOT	UR_FORTSHOOT	The fortresses will shoot projectiles when a fire engine is in a range of 100 screen units and will continue to travel for 2 seconds		If the projectile travels for too long then the map may become cluttered and impact UR_ENJOY	Transformation
FR_REGEN	UR_HEALTH	The fire engine's health is refilled at a specified rate of 60/second when it reaches the fire station	If the game is too difficult it is not fun	The fire engines' health might regenerate too slowly, making the game too difficult and impacting UR_ENJOY	Transformation
FR_REFILL	UR_REFILL	The fire engine's water is refilled at a specified rate of 60/second when it reaches the fire station	If the game is too difficult it is not fun	The fire engines' water might refill too slowly, making the game too difficult and impacting UR_ENJOY	Transformation
FR_CONTROLS	UR_DRIVE	The fire engine can be moved using the arrow	Keyboards are an	If the computer does not have a keyboard,	Transformation

		keys on the keyboard	exceedingly common peripheral	the game cannot be played	
FR_SPEED	UR_DRIVE	The fire engine will move at a specified speed across the map Varied speed across the map from 0 to 100 pixels/second depending on the map beneath the truck. Slower on green and faster on roads		The fire engine might move too slowly for the game to be fun	Transformation
FR_COLLISION	UR_DRIVE	The fire engines will not be able to drive through the fortresses or the city walls		The user may become frustrated if they cannot drive where they like	Transformation
FR_FORTRESSHEALTH	UR_ DIFFICULTY	As the game progresses, the fortresses will increase in strength linearly 0.2 units/second		If the fortresses are too strong, the game cannot be won	Transformation
FR_WIN	UR_END	The system will end the game and present a win screen if all the fortresses have reached 0 health		The system may also have other reasons to end	Invariant
FR_LOSE	UR_END	The system will end the game and present a lose screen if all the fire engines have 0 health		The system may also have other reasons to end	Invariant

		NON-FUNCTIONAL REQUIREMENTS		
ID	**Corresponding UR Req. ID**	**Description**	**Assumptions**	**Fit Criteria**
NFR_REPEATABLE	UR_END	The game should play similarly each time	It is confusing if different playthroughs of the game had different rules	The rules must not change each time the game is played
NFR_TIME	UR_DIFFICULTY	The game should be playable within a reasonable amount of time	The target demographic (open day visitors) will have 10 minutes to	The game must take no longer than 10 minutes to complete

			spare	
NFR_MAINTAINABILITY	UR_UPDATES	The code should be easy to maintain in the future	The code is going to be maintained	There must be extensive documentation and the code kept to the specified standards should follow the Java naming conventions/coding style
NFR_DOCUMENTATION	UR_UPDATES	The code should be documented	Any changes made are justifiable	Extensive documentation, specifying the class structure and code standards, and explaining the code functionality
NFR_RESILIENCE	UR_INFO UR_ENJOY	The system should not crash if an error occurs	We are able to catch bugs in testing	Whenever an error occurs, it will be caught and reported in an error log
NFR_OPERABILITY	UR_TUTORIAL	The system shall be playable by a new player	Players will spend time to understand the start screen	A one page max tutorial page will be shown explaining how to play the game
NFR_USABILITY	UR_INFO UR_TUTORIAL	The game should be simple and easy to understand	A limited colour palette makes for an easier to look at game	Simple colour-scheme and gameplay which can be understood from the NFR_OPERABILITY specified tutorial
NFR_GUI	UR_AESTHETIC UR_INFO	A large GUI may result in the game looking overcomplicated and a cluttered screen	Cluttered screens reduce enjoyability	The GUI will be simple and not take up more than 25% of the screen
NFR_GOALS	UR_ENJOY	To make the game fun, it will have the goal of destroying fortresses	Goals and achievement make games more enjoyable	The user is able to destroy fortresses
NFR_VARIATION	UR_ENJOY	To make the game fun, the fortresses will have different attacks	Variety is gameplay makes games more enjoyable	The fortresses all have unique attack styles
NFR_RUNSPEED	UR_ENJOY	The game should run at a decent frame rate	Frame rates lower than 60fps are irritating to the player	The game does not visibly lag as a result of processing

REFERENCES

[1] K. Pohl, *Requirements engineering*, 1st ed. Berlin: Springer, 2010.

[2] M. Prensky, *Digital game-based learning*, 1st ed. St. Paul: Paragon House, 2001.

[3] J. Dick, E. Hull and K. Jackson, *Requirements Engineering*, Cham, Switzerland: Springer, 2017,

Method Selection and Planning

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 issues was considered as an alternative, but decided against due to multiple members past experience with Jira, as well as its superior *story point* assignment ability to help ensure work is evenly divided; and clear layout.

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. Draw.io was considered as an alternative, but decided against due to LucidChart’s capability for more complex diagrams and collaborative tools.

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. This is compared to, for example, Microsoft Word where documentation versions are not real-time updated and shared.

LibGDX – Game-Development Application Framework:
It was dictated by the product brief that the game must be written in the programming language Java. LibGDX provides a library of useful code base and is cross-platform which is good for our team as members work on a mix of Mac OS and Windows. It allows for the same code to be used to run desktop PC and Android applications, which is appropriate due to our customer's desire (confirmed in one-on-one customer meeting) to run the final game on mobile. jMonkeyEngine was considered as an alternative, but decided against due to its weighting on the use of 3D graphics, which the team decided against in early concept meetings.

Eclipse – Integrated Development Environment:
Although alternatives, such as Visual Studio Code, provide much of the same features, Eclipse was chosen as our IDE as it was found to be the most familiar to the majority of the team. It is easily compatible with our Git usage due to helpful plugins, and the autocomplete and code tidying features contribute to development efficiency.

a) <u>Team Organisation</u>

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].

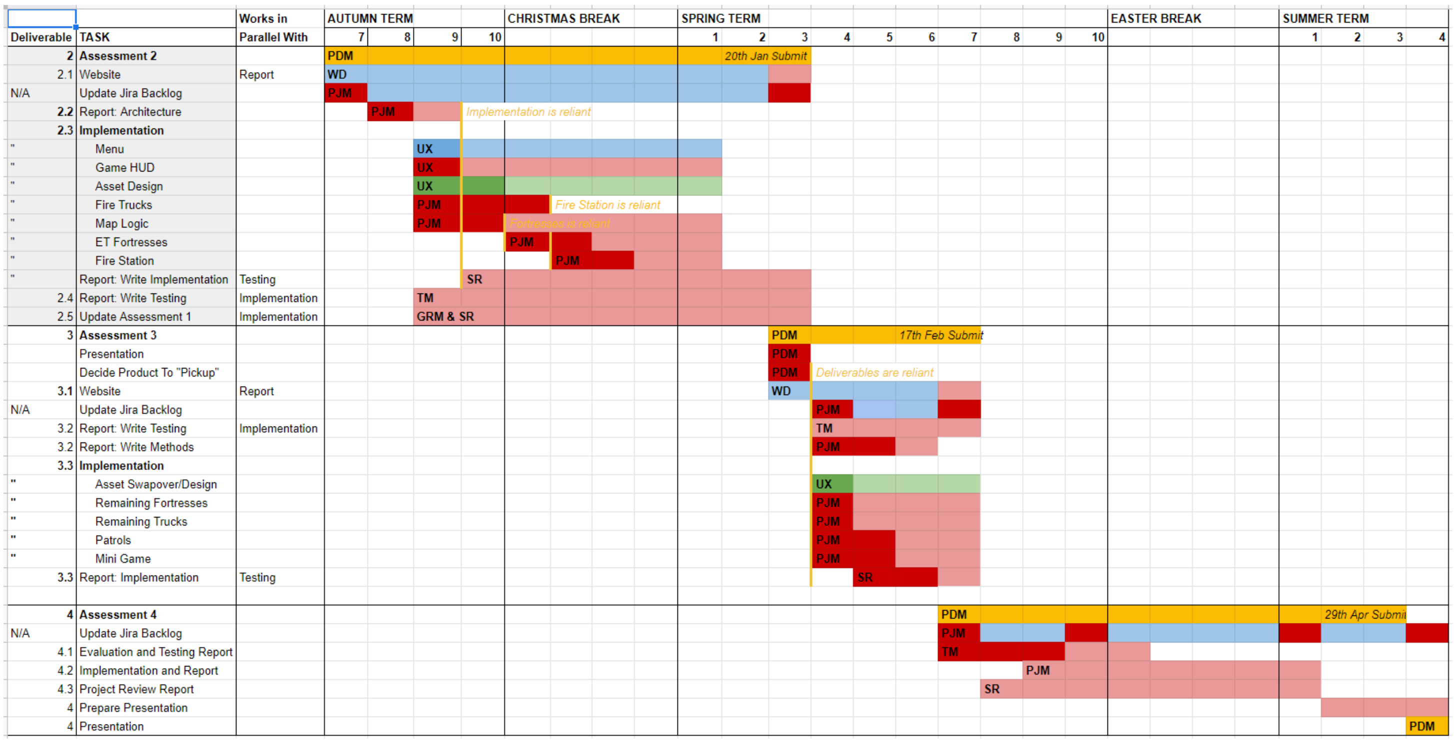
To a certain extent, we aim to maintain this informal approach throughout the organisation of our project, our decision is supported by Sommerville, "Agile development teams are always informal groups". However, we do recognise the need to loosely appoint a number of roles and responsibilities to members, 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. Roles were appointed as a result of open, group discussion considering the above factors. Skill level was determined by verbal self-assessment.

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** (x1) **(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



		Works in	AUTUMN TERM				CHRISTMAS BREAK				SPRING TERM										EASTER BREAK				SUMMER TERM			
Deliverable	TASK	Parallel With	7	8	9	10					1	2	3	4	5	6	7	8	9	10					1	2	3	4
2	**Assessment 2**		PDM									*20th Jan Submit*																
2.1	Website	Report	WD																									
N/A	Update Jira Backlog		PJM																									
2.2	Report: Architecture			PJM		*Implementation is reliant*																						
2.3	**Implementation**																											
"	Menu				UX																							
"	Game HUD				UX																							
"	Asset Design				UX																							
"	Fire Trucks				PJM			*Fire Station is reliant*																				
"	Map Logic				PJM		*Fortresses is reliant*																					
"	ET Fortresses						PJM																					
"	Fire Station							PJM																				
"	Report: Write Implementation	Testing				SR																						
2.4	Report: Write Testing	Implementation			TM																							
2.5	Update Assessment 1	Implementation			GRM & SR																							
3	**Assessment 3**												PDM			*17th Feb Submit*												
	Presentation												PDM															
	Decide Product To "Pickup"												PDM	*Deliverables are reliant*														
3.1	Website	Report											WD															
N/A	Update Jira Backlog													PJM														
3.2	Report: Write Testing	Implementation												TM														
3.2	Report: Write Methods													PJM														
3.3	**Implementation**																											
"	Asset Swapover/Design													UX														
"	Remaining Fortresses													PJM														
"	Remaining Trucks													PJM														
"	Patrols													PJM														
"	Mini Game													PJM														
3.3	Report: Implementation	Testing													SR													
4	**Assessment 4**																PDM									*29th Apr Submi*		
N/A	Update Jira Backlog																PJM											
4.1	Evaluation and Testing Report																TM											
4.2	Implementation and Report																		PJM									
4.3	Project Review Report																	SR										
4	Prepare Presentation																											
4	Presentation																											PDM

The Gantt Chart above 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.

Risk Assessment & Mitigation

Every software engineering project comes with risks which can damage the project. A risk is a future activity or event that may threaten the success of a software engineering project. However, a team can minimise the damage from risks with a robust Risk Management system. For our Risk Management to work well, we will largely be using qualitative analysis. The qualitative analysis can be broken down into the following sections: the identification, categorisation, a description, the likelihood of it occurring, its impact severity and any mitigation or contingency plans. [1]

To maximise the effectiveness of our Risk Management, each risk in our project has been identified by researching online and brainstorming in a group the threats to our project. We looked at websites for common threats to a software engineering project and discussed as a group what some specific threats could be to our project. Each risk was assigned a unique ID, type, likelihood rating, impact severity rating and an owner who is responsible for handling that risk.[2] Our General Risk Manager will be responsible for updating any additional risks discovered during our project. Risks that were clearly insignificant will not be included. Each risk is given an ID based on what risk type it is categorised under. If it can be categorised under multiple types, the type that we believe is most appropriate will take priority.

Risk type, the types of risk and impact of that risk is defined as:

- **Project** *(PJ)* - Risks that affect the project schedule and project resources.
- **Product** *(PD)* - Risks that affect both the functional and non-functional requirements thus the quality of the developed product.
- **Business** *(B)* - Risks that affect the organisation of developing and procuring the software.
- **Technology** *(TN)* - Risks relating to the hardware or software used in the project.
- **People** *(PP)* - Risks relating to team members within the project.
- **Tools** *(T)* - Risks relating to the software and tools used in the project.
- **Requirements** *(R)* - Risks relating to changes in customer requirements while the project is ongoing.
- **Estimation** *(E)* - Risks relating to the estimation of the system characteristics and system resources

The likelihood and severity rating, the probability of a risk occurring and the impact of the risk if it were to occur, will both have three levels:

- Low - Unlikely to occur/Little to no impact.
- Moderate - A fair chance of occuring/A fair amount of impact.
- High - An extremely high chance of occurring/A large impact.

We decided to distribute the risk ownership between three team members to ensure each risk could be constantly monitored. Risks that can be detrimental to the project are assigned more than one manager. As we chose Agile Development Method, the risk monitoring will be done every week in a meeting, each risk manager will discuss any concerns regarding risks and any problems caused.

The following roles have been assigned to distribute risk ownership:

- Project Manager (PJM) - Chloe Hodgson
- Product Manager (PDM) - Jack Thoo-Tinsley
- General Risk Manager (GRM) - Tamour Altaf

As Product and Project were common risks, they have their own managers to tackle those risks and the General Risk Manager will assist in tackling more severe and likely risks.

Risk Planning

ID	Risk Type	Description	Likelihood	Severity	Mitigation/ Contingency	Owner
PJ1	PJ E	Misestimation on development and scheduling times.	Low	Moderate	Consistent use of Jira and its sprint system will be used to stay on top of deadlines.	PJM
PJ2	PJ PP	Members of the team are unavailable to contribute to the project for a prolonged period of time.	Low	High	Avoidance of a low “bus factor”. Each task will be shared between more than one person such that there is no complete dependency on one person.	PJM GRM
PJ3	PJ PP	A member of the team is being uncooperative or not contributing their fair share of the workload.	Low	High	Bi-weekly meetings to check on workload, progress and wellbeing of all team members.	PJM GRM
PJ4	PJ PP	Disagreements between team members	Low	High	Constant communication to make sure all team members are happy. Resolve conflict as soon as possible or contact module supervisor if conflict is serious.	PJM GRM
PJ5	PJ PP	The workload on an individual team member is too great resulting in a decrease in productivity.	Low	Moderate	Bi-weekly checkups. If a team member is struggling with the workload it can be redistributed accordingly.	PJM
PJ6	PJ PP	Overcomplication of simple tasks.	Moderate	Low	Constant peer review on the latest project version.	PJM
PJ7	PJ PP TNT	Learning process of new software or programming	Low	Moderate	Our project manager will be responsible for introducing team members to any new software used. Meetings will	PJM

		languages takes more time than anticipated			be called if team members are still having trouble learning any new software or programming language.	
PJ8	PJ TN T	Files are lost, misplaced or deleted.	Low	High	Backing up documents on different computer systems or online software. Making use of GitHub as it's reliable and having version control throughout the project.	GRM PJM
PJ9	PJ R	A change in requirements midway through the project.	Moderate	Moderate	Clear and concise documentation with all iterations available to everyone through Google Docs and Github. This allows easy changes.	PJM
PJ11	PJ PP	A team member is lacking in motivation or is unsure of their current task.	Moderate	Moderate	If a team member is not enjoying their current task and it is affecting the development schedule then tasks can be re-assigned to better accommodate this.	PJM
PJ12	PJ PP	Other priorities such as university work and social activities may reduce the amount of time going into the project.	High	Moderate	Setting aside bi-weekly meetings and individually an allotment of time ensures that work is done on the project.	GRM PJM
PD1	PD PP	Final product does not meet the requirements and specifications.	Low	High	Regular checkups and peer reviews on work done always keeping in mind the requirements.	PDM PJM
PD2	PD TN T	Relevant technology goes down resulting in the loss or inaccess of our work. E.g. Jira	Low	High	Where possible any documentation is backed up on more than one software.	GRM PDM

PD3	PD T PP	Risk assessment fails to identify all relevant risks.	Low	Moderate	Within the risk managers, review any new risks that have a slight chance of occurrence are noted and added to the assessment.	PDM
PD4	PD PJ PP	Inconsistent documentation	Moderate	Moderate	Documentation will be reviewed weekly by the team to avoid any inconsistencies.	PDM
B1	B PD	Final product contains code bugs.	Low	High	Group testing every week to solve small and large bugs. Regular team meetings and monitoring of the product. Contact module leader if a bug cannot be solved by the team.	PDM GRM
B2	B R PD	Shareholders do not like the final product.	Low	High	Regular communication and always checking that the direction our project is heading is in line with the vision of the shareholders.	PDM GRM
PJ13	PJ PP	Team members are consistently late to scheduled meetings increasing group frustration and effecting productivity	Moderate	Moderate	Guilty team members are reminded of their impact on the other individuals and punctuality expectations are reviewed as a team. Meetings are also scheduled during times close to our degree contact time, so that we should already all be in the local area	PJM

References

[1] P. Simon, D. Hillson and K. Newland, Project Risk Analysis and Management guide. Norwich: APM Group Limited, 1997.

[2] "Risk Management in Software Development and Software Engineering Projects", Castsoftware.com, 2019. [Online]. Available: https://www.castsoftware.com/research-labs/risk-management-in-software-development-and-software-engineering-projects. [Accessed: 01- Nov- 2019].