ENG1 Team Project Assessment 2

Deliverable 2: Change Report

Team 13 – Team Unlucky

LILLY BROWN
ROSCOE GILLATT
ADAM JOHNSON
YUMIS ZYUTYU
BRANDON WEST
AYMAN ZAHIR

a)
Change Management is to ensure that the system's progress is managed and priority is given to the most urgent changes. [1]

We decided to split the project into two halves: implementation and documentation, using GitHub's Projects kanban board. This allows us to automate the change management process, keep a track of all changes and be able to assign priority to those that require urgency.
Documentation Project refers to any changes that are required for the deliverables.
Implementation Project refers to any changes that need to be made to the code as a result of bugs or changes to the requirements.

To manage the changes to both the documentation and implementation, we will be using GitHub Issues, this tool allows us to create an issue that relates to a specific change and assign it to a team member. Within the issue, we will be using markdown to communicate within the team and easily identify what changes have been made and their justification.

For the documentation, we have chosen not to use GitHub's priority columns in the kanban board to prioritise the changes, this is because it doesn't make sense in the context of deliverables; the completion of one deliverable does not have any higher priority over another. Further we have created an automated system where all of the issues created feed into a to-do list, from there an issue may be placed into any of the following categories: Research, In Progress, Review in Progress, Review Complete or Done. Once the issue has been moved into the Done section, the issue can be closed. This process is agile, meaning an issue can be moved into the Review Complete section, only to be moved back to the Research or In Progress section. [2]

For the implementation, we are assigning priorities, this is because within code it may be necessary for a bug to be fixed urgently, opposed to a small feature that does not necessarily need to be implemented immediately. Once an issue has been created it will move into the Needs Triage section, then it can be assigned to either High or Low Priority. Once the code for the issue has been completed, the issue should be assigned to the Tests section, here the testers can move the issue into the Test in Progress section. Finally, when the test has passed for the changed code, the issue can be moved to the Done section and closed, this process can also be automated with Git, so that when a test is passed, the issue is automatically closed. In addition to this, we documented our changes within the actual code by using a specific commenting syntax, this allowed us to record who made the change and on what feature. The example format for one of these comments is as follows: //NAME - CHANGE.

GitHub Issues provides a higher level representation of the changes made that is useful for us as a team during development. For a lower-level legacy view, we will be using the version history feature of Google Docs; this is the tool we are using for collaboration and it contains a feature that allows us to view which member has added what to each document and access any previous versions of each document.

Inspections and reviews will be conducted after reaching a predefined development milestone outlined in the plan and referenced in the change report at the end of the project.

b)
i) Requirements
When it came to making the changes to the requirements, we decided to look first at what requirements needed to be removed. Firstly, within our team meeting with the client, we asked for some clarification regarding the differences between the XP and points system. It was concluded that only one of these was necessary because they operated in the same manner and were essentially the same system but named differently. We decided that we would remove XP as a feature and only implement points, this meant that a variety of requirements would need to be removed. The first of these was the user requirement UR_EARN_XP, which dictated that the user should be able to earn XP, along with its two relevant functional requirements. FR_XP_TRACKING, stated that the game should be able to keep track of the user's XP and FR_XP_UPDATE stated that when a user completes an encounter they receive XP. Obviously, these two requirements were not necessary to keep if the XP system was being removed. Another requirement we chose to remove was FR_MENU_KB_INPUT, stating that keyboard input should be accepted for menu navigation, because the requirement was not implemented by the previous team and not mentioned by the client for the new requirements, it could be deleted.

Next, we looked at which requirements needed their priorities to be changed, all the requirements moved from a 'may' to 'shall' priority. The reason the priorities needed to be changed is that these requirements were not implemented by the original team, but are required by the client for the second half of the assessment. The following requirements had their priorities changed: UR_SHIP_COMBAT, UR_OBSTACLE_ENCOUNTER, UR_WEATHER_ENCOUNTER, UR_SPEND_MONEY.

After this, we looked at the requirements table and realised that not every user requirement had a related requirement attached to it. In most cases, these were user requirements the previous team added for the second half of the project but only had 'may' priority. We decided to update the user requirements table to account for this, highlighting where we needed to create functional requirements to relate to a user requirement. The user requirements and their added functional requirements are as follows:

- UR_FRIENDLY_INTERACT: FR_FRIENDLY_COLLEGE
- UR_HOSTILE_BUILDING_COMBAT: FR_COMBAT
- UR_HOSTILE_COLLEGE_CAPTURE: FR_COLLEGE_CAPTURE
- UR_SHIP_COMBAT: FR_COMBAT
- UR_OBSTACLE_ENCOUNTER: FR_OBSTACLES
- UR_WEATHER_ENCOUNTER: FR_WEATHER_ENCOUNTER
- UR_SPEND_MONEY: FR_SHOP

The various descriptions of these requirements can be found on our website under the requirements table. All new functional requirements had their priority set to 'shall'.

As a result of the new brief, extra user requirements would need to be added, the first of these was UR_POWER_UPS, which states that a user should be able to collect five power-ups, including temporary immunity and repairing damage. This requirement is in the earnables category and has its priority set to 'shall'.
As per the brief we needed to implement a variety of game difficulties, therefore we added the requirement UR_GAME_DIFFICULTY under the 'game setting' category. This relates to the user requirement UR_GAME_INIT and will have its priority set to 'shall', allowing the user to select either easy, normal or hard difficulty.
Another new requirement that was added in the brief was that the user should be able to save the game state at any point, therefore we added the new requirement UR_SAVE_POINT under the 'game setting' category. The priority of this requirement will be

set to ‘shall’ as it is imperative we implement it. After we created these user requirements, we needed to create new functional requirements for them to relate to, they are as follows:

- UR_POWER_UPS: FR_POWERUPS
- UR_GAME_DIFFICULTY: FR_GAME_DIFFICULTY
- UR_SAVE_POINT: FR_SAVE_POINT

The descriptions for these functional and user requirements can be found on the website in the requirements table.

We only added one new non-functional requirement, this was NFR_NO_FLASHING, which states that the weather effects or any animation should not flash within the game. This relates to the risk R22 and the user requirement UR_PLATFORM. At first, we were going to create a new user requirement, but then we realised that NFR_COLOURBLINDNESS, which has a similar context was related to UR_PLATFORM. Therefore, we justified relating it to the same user requirement, because it means that a user wouldn’t need to use a specific anti-epilepsy machine to play the game, in the same way, a user wouldn’t need to use a colourblind screen.

URL to the changed document:
https://apj520.github.io/ENG1Team13.github.io/pdfs/Req2.pdf

ii) Abstract and Concrete Architecture
We began making the changes to the abstract architecture first, this would allow us to better understand the changes that needed to be made for the implementation. The previous team did not have their abstract architecture in class diagram format and this is something that was mentioned in their feedback, therefore we decided to change the format to accommodate this. We extracted the core classes from the previous teams' class diagram and converted it into class diagram format, which required us to also add generic/example variables and functions. We also added the relations between each class and the interface. The new abstract architecture can be viewed on the website.

As we decided to use the same tools as the previous team to represent the architecture, the changes to the concrete architecture were far more simple; we kept the original diagrams and simply appended any new classes or variables/functions that were necessary to implement due to the new requirements. The changed parts of the diagram are obvious because their diagram is a different colour to the originals. The new concrete architecture can be viewed on the website with the following link: **insert link here**. We also trimmed the diagrams for classes that were not directly referenced in the requirements table (non-key classes), if they were not mentioned then they were redundant.

In the feedback the previous team received, they were told they had not fully implemented a table that broke down the relationship between the elements of the architecture and the functional/user requirements. We made sure to fully complete the table to ensure this criticism was satisfied. Finally, we resized some of the images of the UML diagrams to ensure that the page limit was not exceeded.

We used different tools to implement certain aspects of the architecture. The previous team used draw.io to produce their abstract architecture, we used PlantUML to implement the new abstract architecture. Upon deciding that we were going to implement the abstract architecture with class diagrams, we knew we would have to switch tools, as we were familiar with using PlantUML to implement class diagrams from the previous part of the assessment, we chose to continue to use that. For the concrete architecture, the previous team used PlantUML and Adobe Photoshop; we will continue to use PlantUML and we will also continue to use Photoshop for editing the screenshots of the diagrams. As previously mentioned PlantUML is a tool we were familiar with using to implement class diagrams and Photoshop is one of the best image editing tools and is one that team members were also familiar with so it made sense to continue using it.
URL to the changed document:
https://apj520.github.io/ENG1Team13.github.io/pdfs/Arch2.pdf

iii) Methods and plans
The following changes were done followed by their justification:

- Part A
    - We changed the software engineering method from plan-based to RAD, due to the team being already familiar with that method rather than having to learn a new one. The method seemed to work well for the team in the first part of the assessment and therefore we were very confident when deciding to keep it.
    - Regarding what was discussed in the meetings, game design was discussed but it wasn't the main part, therefore we added an additional clarification to make it more accurate.

    - Tools Used - we used similar software to Team 3's, however, we used a few additional ones to support the documentation and testing.
        - Communication and Collaboration
            - Zoom was removed from the list of tools for communication as our team meetings were in person. This was chosen as it was convenient for every team member to meet in person and allowed us to have more open discussions and clearer communication than we would have online. It made it easier for the implementation team to work together on the code and prevented any delays in communication.
            - The team used Discord as a communication platform too however, additionally, we created Discord channels such as "Documents", "Research", "Module-Resources", "To Do" and "Weekly snapshots" which made it easier for the team to navigate through the server and enable the team to communicate simultaneously on different topics.
            - We used Github projects and issues to organise and keep track of tasks rather than a Trello board as the team was already familiar with using GitHub from the first part of the assessment.
            - Google drive - used for documentation keeping, version control and quality control. Team 3 did not have any tools listed for such matters, therefore it was essential to include it in the documentation.
        - Planning - a new tools category.
            - For planning purposes, the team used OfficeTimeline to create a Gantt chart. OfficeTimeline was used rather than Excel as it provided a specialised user interface and extra tools such as task allocation, % completeness, urgency and status.
        - Architecture
            - Draw.io was removed from the tools list due to not being used by the team to create the abstract architecture, instead, we choose to be consistent with the software and used PlantUML for both diagrams.
        - Implementation
            - We added GitHub Actions as a continuous integration tool, which wasn't needed for the first part of the assessment and was therefore missing from the tools list.
            - We added a new tool category for testing as that wasn't part of the first assessment.

        - Alternatives considered
            - We considered Eclipse as an IDE, however, our reasoning as to why we didn't use it differed from team 3's and was therefore changed.
            - All the alternatives considered for testing and CI were added as they were not part of the first assessment.
- Part B
    - Team Roles
        - We did not have the same role allocations as Team 3 due to having a different software engineering method than them. Therefore, their team roles were all removed from the documentation and ours were added.
    - Task Assignments
        - Task Assignment Method was different for our team as we had a better idea of each member's skill set as well as some members were already familiar with the reoccurring parts of the assessment process from the previous assignment. We had been given an allocation of marks, which made the task assignment easier, knowing already the skill set of every team member, we managed to allocate the tasks at an earlier stage than Team 3. Therefore, their task assignment method was changed.
    - Quality Control
        - Team 3 did not have any quality control measures set in place, so we decided it was essential to add these in order to ensure the high quality of the final product. This was done to ensure every document was checked at least twice by different team members.
        - For the implementation, we used GitHub version control as a quality control tool, which was added to the list of tools as Team 3 did not use it.
- Part C
    - Plan - Team 3's plan overview was kept as per instructions from the assessment brief.
    - Gantt Chart - we added our Initial Gantt chart Plan as well as the final look of the Gantt chart and a paragraph explaining the colour coding used for the prioritisation of tasks. The Gantt chart includes the team members assigned to each section as well as the dependencies of the deliverables. An individual image of the Gantt Chart as well as the weekly snapshots can be found here: https://apj520.github.io/ENG1Team13.github.io/index2.html

    - URL to the changed document: https://apj520.github.io/ENG1Team13.github.io/pdfs/Plan2.pdf

iv) Risk Assessment and Mitigation
Our approach towards the change management of the risk assessment was first to identify and remove any inappropriately worded content or content that the team could not trace back to either the customer requirements or the assessment brief. Furthermore, we met to discuss and add new risks that the previous team overlooked or did not consider.

The presentation was kept the same - a table. A new column relating the risks to the relevant requirements from the requirements table was added to allow the reader to easily and efficiently navigate between the two pieces of documentation if a risk occurs.
A standardised colour-coding system for different severity levels was added to increase the team's efficiency and reduce the time to navigate the table. Making higher priority risks easier to spot. Three distinct colours were used - red(H), yellow(M) and green(L). To guarantee that high-level risks are reaching their goal and completion. Additionally, the team went over the already existing risks and changed any severity levels that did not seem correct.
The name and descriptions of the table's columns were kept the same as they are appropriate and all relevant for the risk table. The removal of the owner column was considered due to the majority of the risks belonging to the same team member; however, after a discussion was decided to keep it as it is and make sure that all risks have been linked to the correct team members as it will contribute positively if any risks have to be traced back. The description of the "Owner" column on the first page was changed to give a better understanding of what is meant by it.
The types created by the previous team were kept as they were relevant, however, the following new types and the new risks that fit in those categories were added, as we thought they were of importance to the project and the lack of them could delay the search process within the risk assessment table:

- Tools - R16, R20, R26, R27
- Requirements - R19, R24
- Estimation - R14, R17

The description of what the new and all other types mean was added to the introduction of the document, as it would be useful for the reader to understand what was meant by each type as some of them are not as obvious.

More risks were added to the "People" risk type, those were risks related to the occurrence of any team conflict (R18), any human errors from other team members (R23), miscommunication throughout the project (R25), loss of work (R28) and lack of any research material being distributed by the team member responsible for it (R29). As well, we thought would be appropriate to add any risks that could have occurred due to negligence from the previous team's methods (R31). All of these were added as the previous team did not consider enough risks within their team communication.
The risks directly related to the game - types "Technology" and "Product" were kept as they were due to that part of the game not changing in the new part of the assessment. New risks in those types were added to reflect the new requirements for the game. R21 was added in case of any major bugs, R30 and R22 related to the new features.

The following correction on the already existing risks was done:

- R7 and R8 mitigations were changed to suit the risks better as their previous description wasn't appropriately worded.
- R15 mitigation to fit the team's methodology and platforms of communication.

All of these were carefully analysed, discussed and approved by all team members, corrections were done where necessary after the first draft. The team kept reviewing and adding new risks throughout the whole continuation of the project.
URL to the changed document:
https://apj520.github.io/ENG1Team13.github.io/pdfs/Risk2.pdf

Bibliography:

[1] Ian Sommerville, Software Engineering, Pearson, 2015. Chapter 25 p.746
[2] Ian Sommerville, Software Engineering, Pearson, 2015. Chapter 3 p.76