Assessment: 2

Deliverable: Change report

Team Name: Team 8

Team members: Charlie Hayes, Matilda Garcia, Joshua Stafford, David Kayode, Ionut Manasia, Matthew Tomlinson

Formal Approach

Before undertaking the second assessment, research took place to understand how our team could best manage the resulting changes. Initially, research on how to manage the changes to requirements was the focus [1], before switching to change management as a whole [2].

Following the information gained from research on requirements change management, we began the process of specifying the changes. This involved meeting with the customer to further clarify the implementation expected for each of the new requirements, creating entries for each change in the requirements documentation and 'costing' them to assign priority to each one. Then kanban entries were formed from the requirements and priority ready for assignment.

This approach was followed for the deliverable changes themselves as well (not including the costing). Each deliverable was analysed to find necessary changes (in addition to those resulting from the new requirements and team changeover) which were used to create 'change requests' on our documentation kanban board. Once changes were approved by the team, deliverable changes were assigned to team members who already had experience in the related deliverable. Modifications and additions to each deliverable are highlighted to indicate a change from the original along with justification in the change report below (also in the deliverable itself where page limits allow or as Google docs comments). Where notable aspects of the deliverables were removed (e.g. irrelevant risks) or modified beyond recognition (abstract architecture diagram) they have been added to an appendix at the end of each deliverable.

From the general change management research [2] it was understood that in larger companies a formal change request form would be heavily used. However, it is noted that for smaller organisations (like ourselves), a less formal process is appropriate, although change requests should still be formally recorded, costed and approved by team members. Due to this, we felt the management process, as described above, was suited to our situation.

For code, a very similar change request approach was used. Proposed changes to the code were added to another kanban board (using GitHub issues and projects) marked as either a bug or enhancement along with a priority. This way, team members could effectively determine which changes to focus on. For tracking changes made to the code, different branches allow separation of changes/features with the changes themselves accompanied by a derivation history for each changed component (as discovered during research [2]). For the derivation history, a standardised comment is typical. As we are using java for the project, the @version, @since and @author javadoc tags will be used (alongside relevant comments) as they clearly convey who made the changes and @since 2 will mark that a given method/class is new for the given version or @since 1 to show an extension and @version 2 will mark if it was modified for this new version (for this section of the project, version 2 seemed appropriate). The benefit of this approach is that generating the javadocs will pick up these tags and allow easy identification and tracking of changes from the code documentation (rather than just in the comments of the code).

For the documentation of the code itself and individual changes, this is indicated with 'MODIFIED' comment followed by justification and the author can be tracked through git and git blame.

Requirements [3]

In terms of the changes made to the original requirements, we decided to add a constraint requirements table, changing the original number of tables from 3 to a total of 4 requirement tables. These requirements were put in place as a way to ensure the specific needs of the customer are met and our team stays in line with the customer's vision. We also made a few alterations to the already existing user requirement table. For example, a minor alteration we made was to ensure the test for 'UR_MENU 1.2.1' included the ability to change difficulty using buttons to navigate through the game. Another update added was 'UR_BOAT_CHOICE 1.2.3' which enables the player to pick their desired boat in the menu in accordance with the new requirements set in assessment 2,'UR_LEGS1.3.0' which ensures that there are exactly 4 legs to the race, this requirement was added because it was an essential requirement in assessment 1 but was not included in the original.

Other changes in user requirements include 'UR_WIN1.3.2' which makes sure the player must be able to win and be explicitly referenced as the winner when the game is completed, 'UR_LOSE1.3.3' which enables the player to lose and be clearly referenced if they come out last, 'UR_POWERUPS1.3.4' this new requirement allows users to collect up to 5 power-ups during the race. These power-ups give them a special advantage such as improved stamina or health. The final addition we made to the user requirements was 'UR_SAVE1.3.5' which was essential to assessment 2, this allows the user to save their progress in the game, data such as health, stamina, position and previous leg results will be saved. We included 'FR_DIFFICULTY2.0.10' to the functional requirements in order to alter the number of obstacles encountered depending on the difficulty the player initially selected at the start menu. This requirement is important because without it the game would not include any form of difficulty, lack of difficulty during gameplay would lead to customer and user dissatisfaction.

The constraint requirements consist of 4 new requirements. Firstly, 'CR_2D 4.0.0' was added because the customer would rather the game be in 2D. Secondly, 'CR_GRAPHICS_CONSISTENCY 4.0.1' was added in accordance with the customer's desire for consistent graphics throughout the game. 'CR_LANGUAGE 4.0.2' tackles the constraint of the game being programmed in Java. Finally, 'CR_VIOLENCE 4.0.3' restricts the game from having any form of violence or gore displayed in it. Ultimately, the constraint requirements were included because the original document didn't include any solid constraint requirement, and the addition of four constraint requirements is suitable in this instance as too many constraints could limit potential solutions and stifle the project.

We also removed a few requirements that didn't necessarily correlate with the customer's visions for the game. One was 'UR_CHANGE_RESOLUTION 1.2.2' which enabled users to be able to play in different resolutions, this feature isn't of high priority and importance due to the customer's requirement of consistent graphics throughout the game, the option of changing resolution deviates the main focus of graphic consistency. Another requirement that was removed was 'UR_CHANGE_SETTINGS 1.2.4' this allowed the user to change settings such as audio control and graphic settings, this feature isn't high on the priority list nor mandatory and is therefore inconsequential, hence why it was removed. Lastly, 'FR_MINIMAP2.0.8' which enabled the player to see a minimap in the corner of the screen was also removed as it wasn't a necessity. In conclusion, all three requirements were removed in order to maintain a straight to the point game and avoid any excessive complexities that aren't needed.

Architecture [4]

The main change that needed to be made to the original architecture document in section (a) was that the UML state diagram originally produced in the first document needed to be replaced by an abstract architecture document. This decision was made as the new diagram helped to better visualise the structure of the system and key relationships as opposed to the previous structure with the UML state diagram which did not read as clear. Another change that needed to be made was that the concrete architecture diagram was large and complex, and therefore difficult to read and digest, so a description of the concrete architecture and how it has been extended from the abstract diagram was also added to the updated version of the document. Details were added explaining key classes that were extended from their abstract counterparts so that the purpose of each element and how each links to build the complete system. These changes were made in order to make the structure of the system clear and to allow for easy implementation throughout project development.

Further changes needed to be made to the architecture diagrams in order to allow for changes in requirements and implementing these into the system. Additional classes and methods were added for new requirements such as implementing power ups, with the class Powerup inheriting from the obstacle class. Descriptions of each new class and methods being added were also explained here in order to understand which was being added to fulfil certain requirements and where in the structure these would need to be added for completeness.

The changes that needed to be made in part (b) to the original architecture document were needed to compensate for the large change made in replacing the UML state diagram with the abstract class diagram. These changes included added justification for the new abstract class diagram and explanations of how the concrete architecture diagram was extended from the abstract architecture, and how this aids development. Additional changes made in this section were justifications added for the new classes which were added when the new requirements were introduced, i.e Powerup and SaveLoad classes, with explanations of the attributes and methods, including those inherited from other classes.

Method Selection and Planning [5]

After analysing the feedback from assessment one, and the documentation itself, no major changes needed to be made to the Software engineering methods section in part (a) as this seemed to meet the required specification. We looked at the teams chosen methods and their tools used for development and researched these to decide how appropriate they were for progress of the project. We decided their use of a SCRUM methodology was the best way to continue with the project as we chose a similar approach to assessment one, and we continued to work in sprints and delegate tasks to team members. SCRUM was the most appropriate for this part of the project as there were many changes to the requirements which meant a lot of documentation had to be updated and major code elements needed to be reworked or added to ensure the game was working as expected. Further details were added of the software tool 'TeamGantt' used to produce the Gantt chart that was reworked and extended for assessment two.

Most of the changes made were in terms of the team organisation section in part (b) of method selection and planning. Details were required of the key roles each member of our team had taken on throughout the project and how this aided delegation of tasks during development, so this section was added to the updated document. This change was needed because it was important to discuss the role each member of the team played during our meetings and it also helps to show how we made the most effective decisions in terms of assigning tasks to people who had the most suitable role, and ensuring there was a leader who took control in ensuring everyone had a suitable task to complete. Further information was also added about our SCRUM method approach and which team members were initially assigned the SCRUM main roles, as well as a discussion of how these aided the development of the project.

A further change that needed to be made was the expansion of the plan for assessment 2, as the original method selection and planning document only had links to the Gantt chart for assessment 1, and discussed only the weeks of development earlier in the project. Further detailed versions of the Gantt charts were produced as well as Gantt charts detailing the plan for meetings and assigned tasks throughout the further weeks of development. A discussion of these meetings, assigned tasks and approximated deadlines was also added to the updated document following the previous plan discussed by the original team in order for the full plan to be discussed in detail and to completion.

Risk assessment and mitigation [6]

The original risk assessment and mitigation document states there are 6 defined columns and states the second column represents risk types but doesn't explicitly define what each type means, hence why it was the initial change that was made. This change was made in order for the reader to clearly understand what each type of classification means. For example - A project type risk is one that affects project schedule or resources, a product risk is a risk that affects product completeness/quality, a business risk affects the organisation procuring/developing software. Another update we made was adding the definition of mitigation as the sixth column in the table. Additionally, the register of risks has been split into 5 categories; Technology/Tools, People, Requirements, Estimation and Brownfield. This was done because a good risk register should be split into categories for easy referencing and analysis of risks, this categorization format allows easy addition of new risks as soon as they are identified.

Apart from the categorization of the risk register, a few alterations have been made such as risks being removed due to irrelevancy, for example, the 'difficulty in finding a fitting soundtrack'(ID 7). This risk was removed due to the fact that the soundtrack was not a requirement for the game, hence inability to find a suitable soundtrack does not pose any risks at all. Another risk was 'difficulty acquiring the right software to develop the game' (ID 18), which was removed because the only software required for the game were text editor, java and a game framework. Which are all relatively popular and easy to access therefore making the risk (ID 18) invalid.

Another change that was made was the merging of recurring and duplicated risks. Such as IDs 10-17, which are all risks associated with code quality. They were all merged into 'Poor code quality or missing features due to tight time constraints' (E2). This was done to eliminate redundant risks, meaning the risks could be omitted without loss of meaning. In addition, more risks were identified and new owners for each risk have been assigned due to the previous risks being addressed to the previous group members. This was done in order to allow our current team members the ability to tackle any potential risks that arise.

References

[1] I Sommerville, “4.6.2 Requirements change management” in Software Engineering, Tenth Edition. Pearson, 2016, p133.

[2] I Sommerville, “25.3 Change Management” in Software Engineering, Tenth Edition. Pearson, 2016, p745-750.

[3] Hard G for GIFS & The 8-Team, “Requirements” in Req2

[4] Hard G for GIFS & The 8-Team, “Architecture” in Arch2

[5] Hard G for GIFS & The 8-Team, “Method Selection and Planning” in Plan2

[6] Hard G for GIFS & The 8-Team, “Risk assessment and mitigation” in Risk2