Requirements

Mozzarella Bytes

Requirements

Requirements engineering process: Requirements engineering (RE) is the process of developing a software specification that communicates the systems' needs from the customers to the developers [1]. Requirements should abide by the IEEE standard of being [2]: traceable, feasible, singular, verifiable, consistent, unambiguous, complete, correct, and affordable (see reference for definitions). The elicitation, analysis, documentation and validation phases of the requirements engineering process [3] are needed to ensure that the product will best fulfil the client's needs.

How requirements were elicited: To elicit the client's "real" requirements (which often differ from their "stated" requirements [4]) we used a combination of techniques. Firstly, we used the traditional technique of analyzing the initial brief for "stated" requirements that were ambiguous or imprecise. From this we designed open and closed-ended questions aimed at clarifying and eliciting new requirements [5]. These questions were then discussed with the main client to help discover the "real" requirements. After analysing the requirements we clarified those that did not meet the IEEE standard in an interview with the customer.

How requirements were validated: To validate the requirements, they were systematically peer reviewed by members of the group to ensure the team was confident that they could implement them and there were no conflicting requirements. The documented requirements were then walked through with the stakeholder. Necessary changes were discussed, which often restarted the RE process; this continued until both parties agreed on the requirements. This phase is necessary to ensure that the requirements are feasible, and that they define the system that the client really wants [1].

Why requirements are presented as they are: In accordance with the IEEE standard [2], requirements should have a: unique identifier, priority, source, type, assumed difficulty and risk. The requirements are split into three tables dependent on their type. The user requirements table states what services the user expects the system to provide, and is written in natural language to allow customers to understand it [1]. The functional requirements table describes the system's functions in more detail, while the non-functional requirements table include system property constraints under how the system must operate [4]. To be able to verify a non-functional requirement, there is a fit criteria column which provides a means to measure whether that requirement has been met.

To abide by the IEEE standard, each requirement has a relevant unique identifier (ID). This allows the non-functional and functional requirements to be traced back to the user requirements. Each table also has a source column to state whether that requirement was elicited from the product brief or a customer interview, and a priority column where the requirements are ranked according to whether it shall, should or may be met. To indicate the perceived difficulty of implementing a particular requirement, a colour code system is used where green is easy, amber is nominal and red is difficult.

Requirements change management:

Single statement of need: Build a single-player game that involves moving fire engines between the Fire Station and the ET fortresses, avoiding ET patrols on the way, and attacking ET fortresses when the fire engines' water cannons are within shooting range.

Use case:

- Name: "Win"
- Context: The user destroys all fortresses before all of their fire trucks have been destroyed
- Primary Actor: The user
- Stakeholders: Us - with our interests being the user enjoying the game
- Precondition: The system is working as intended
- Minimal Postcondition: The user destroys all the fortresses
- Trigger: The user destroys the last fortress
- Main Success Scenario: 1.The user destroys the last fortress
    - 2.The win state is displayed
    - 3.The game ends
    - 4.Goes back to the game menu
- Secondary scenarios:
    - 2.1 The game continues to run as if the end isn't reached, e.g the
    - fire station timer continues to run so the fire station is destroyed
    - 4.1 The game starts again
- Success Postcondition: The user enjoyed the game and did not find it too easy

User requirements

User ID	**Description**	**Source**	**Priority**
UR_WIN	The player wins if they flood the ET fortresses before the ET fortresses and ET patrols destroy all of the player's fire trucks	Product brief	Shall
UR_LOSE	The game is lost if all the player's fire trucks have been destroyed before the player has flooded all of the ET fortresses	Product brief	Shall
UR_MINI_GAME	The game should include a minigame	Product brief	Should
UR_MINI_ GAME_THEME	The minigame should be different in style, but aligned to the theme of the main game	Product brief	Should
UR_REPAIR	Fire trucks can be repaired and refilled at the fire station	Product brief	Should
UR_FIRE_ TRUCKS	There must be at least four fire trucks	Product brief	Shall
UR_FORTRESS	There must be at least six fortresses	Product brief	Should
UR_ SCALABILITY	The game should be able to be played on other platforms	Product brief	May
UR_PATROLS	There should be ET patrols that the user aims to avoid	Product brief	Should

UR_DESTROY_STATION	At a point in the game the fire station should be destroyed	Product brief	Should
UR_ENJOYABILITY	The game should be enjoyable to play	Product brief	May
UR_PLAYABLE	The game must be playable Dependent on environmental assumptions (see bottom of document)	Product brief	Shall
UR_PLAYER	The game must be a single-player game	Product brief	Shall
UR_CODE	The game must be coded in Java	Interview	Shall
UR_PC	It must be a PC game	Interview	Shall

Functional requirements

ID	Description	Source	Priority	User ID
FR_FIRE_TRUCKS	Each fire truck must have a unique spec in terms of its speed, amount of damage it can take before being destroyed, the volume of water it can carry, the range and delivery rate of its water cannon	Product brief	Shall	UR_FIRE_TRUCKS
FR_FORTRESS	Each ET fortress must have a unique spec in terms of the range of its defensive weapons, the amount of damage these weapons can deal to Fire trucks over a period of time, and the volume of water it takes to flood	Product brief	Should	UR_FORTRESS
FR_WATER	Over time the amount of water needed to flood a fortress should increase	Interview	Should	UR_WIN
FR_MOBILITY	The user can move the fire trucks. Patrols and fire engines should be mobile; fortresses should be immobile	Product brief	Shall	UR_ENJOYABILITY
FR_AI	The ET patrols and ET fortresses are controlled by the computer AI	Product brief	Shall	UR_PLAYER
FR_TRUCK_ATTACK	Fire trucks can flood ET fortresses	Product brief	Shall	UR_FIRE_TRUCKS
FR_PATROL_ATTACK	ET patrols can attack trucks	Product brief	Shall	UR_PATROLS
FR_FORTRESS_ATTACK	ET fortresses attack trucks	Product brief	Shall	UR_FORTRESS

FR_VIEW _TIMER	The player must see the amount of time until the fire station is destroyed	Interview	Should	UR_ DESTROY _STATION
FR_PATROL _INCREASE	The number of patrols should increase throughout the game	Interview	Should	UR_ PATROLS
FR_PATROL_ DAMAGE	Patrols should damage fire trucks in close proximity to them but to a lesser extent then the fortresses	Interview	Should	UR_ PATROLS
FR_PATROL _SIGHT	Patrols should chase fire trucks that are within their range of sight	Interview	May	UR_ PATROLS
FR_ACCESS_ MINIGAME	The mini game should be accessed from within the main game	Interview	Should	UR_MINI _GAME
FR_ CONTROLS	There should be a screen that explains the controls	Interview	May	UR_ PLAYABLE
FR_ STATION _DESTROY	Fire trucks cannot be repaired or refilled after the fire station has been destroyed	Product brief	Should	UR_ DESTROY _STATION

Non-functional requirements

ID	Description	Rational	Fit criteria	User ID
NF_PC	The game must be playable on engines/things that can be played on PC	Product brief	The game must use libraries /function that could be used on other platforms	UR_ PLAYABL E
NF_ RESPONS E	The game must respond quickly to user input	Improved user experience	Average response time >1 second, maximum response time >2 second	UR_ ENJOYABI LITY
NF_ CONTROL S	The controls should be easy to learn	Prospective students should be able to play the game	The player should be able to grasp the controls in under 2 minutes	UR_ ENJOYABI LITY

Environmental assumptions: 1) The player is assumed to be playing on a modern computer that is of reasonable specifications. 2) The user will have standard hardware such as a keyboard and mouse. 3) The user will have java installed to run the program.

Risks: To ensure these requirements are met we will take precautionary steps to mitigate potential risks. The main risks relevant to the requirements are R7 (see Risk Assessment) which could mean that the game is not enjoyable (UR_ENJOYABILITY) if the game isn't working as envisioned by the client, or potentially meaning the game isn't playable (UR_PLAYABLE). Another main risk is R4 which would mean some requirements could become obsolete or that new requirements will be needed.