

**MACAU UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**School of Computer Science and Engineering**

**Faculty of Innovation Engineering**

**<<Software Project for Course Software Engineering>>**

Homework ID : Task2-User Requirements Specifications

Report Title : Tarturus User Requirements Specifications

Student Name : Yu Siyuan, Liu Benhuang

Student No. : 1220004131, 1220004875

Date : 28/10/2024

**Amendment History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Modified By** | **Reviewed By** | **Amendment Details** |
| 1.0 | 10/27 | Yu Siyuan Liu Benhuang | Yu Siyuan | Initial version.1.1 |
| 1.1 | 10/28 | Yu Siyuan Liu Benhuang | Liu Benhuang | Version.1.2 |

Table of Contents

[1. Introduction 4](#_Toc178688185)

[1.1. Purpose 4](#_Toc178688186)

[1.2. Notation 4](#_Toc178688187)

[1.3. Scope 4](#_Toc178688188)

[1.4. Context Diagram 4](#_Toc178688189)

[1.5. Definitions and Acronyms 4](#_Toc178688190)

[1.6. References 4](#_Toc178688191)

[1.7. Overview 4](#_Toc178688192)

[2. General Description 4](#_Toc178688193)

[2.1. System Functions 4](#_Toc178688194)

[2.2. User Characteristics 5](#_Toc178688195)

[3. General Constraints 5](#_Toc178688196)

[3.1. Software Constraints 5](#_Toc178688197)

[3.2. Hardware Constraints 5](#_Toc178688198)

[4. Assumptions and Dependencies 5](#_Toc178688199)

[5. Functional Requirements Master List 5](#_Toc178688200)

[6. Functional Requirement REQ-1.1 6](#_Toc178688201)

[6.1. Description 6](#_Toc178688202)

[6.2. System Input 6](#_Toc178688203)

[6.3. Display 6](#_Toc178688204)

[6.4. System Processing 6](#_Toc178688205)

[6.5. System Output 6](#_Toc178688206)

[6.6. Other 6](#_Toc178688207)

[6.7. Constraints 6](#_Toc178688208)

[6.8. Data Handling 6](#_Toc178688209)

[6.9. Error Handling 6](#_Toc178688210)

[7. External Interface Requirement 7](#_Toc178688211)

[7.1. Data Interfaces 7](#_Toc178688212)

[7.2. User Interfaces 7](#_Toc178688213)

[7.3. Other Interfaces 7](#_Toc178688214)

[8. Non-Functional Requirements 7](#_Toc178688215)

[8.1. System Performance 7](#_Toc178688216)

[8.2. Information Security 8](#_Toc178688217)

[8.3. Availability 8](#_Toc178688218)

[8.4. Capacity 9](#_Toc178688219)

# Introduction

## Purpose

This document outlines the system requirements for the game "Tarturus," based on "Vampire Survivors". It aims to provide detailed requirements to guide the development process, ensuring alignment with user expectations and enhancing overall gameplay experience.

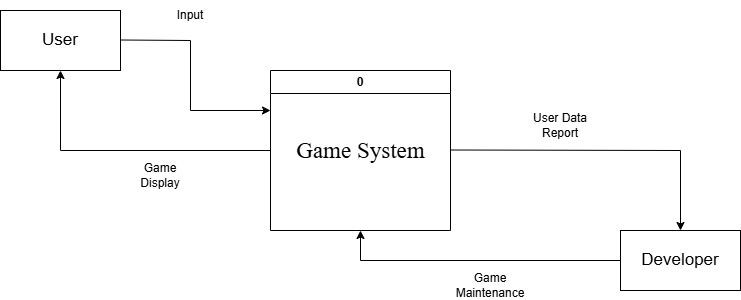
## Notation

This document uses Unified Modeling Language (UML) notation for diagrams and system specifications.

## Scope

The scope of this project includes the design, development, and testing of the game "Tarturus." This includes defining game mechanics, story arcs, character progression, visual and audio enhancements, and implementing the core survival and RPG elements to ensure an engaging player experience.

## Context Diagram



## Definitions and Acronyms

1. SRS (System Requirements Specification): A document describing the detailed requirements of a software system, including functional and non-functional requirements, to guide the development process.
2. BRS (Business Requirements Specification): A document describing the business needs and expectations of users and customers for the system, serving as the foundation for system design.
3. HLD (High-Level Design): Describes the overall architecture and design of the system's major components, providing guidance for detailed design and development.
4. LLD (Low-Level Design): Describes the detailed implementation of each system component, including logic and methods, serving as the basis for developers to write code.
5. UML (Unified Modelling Language): A standardized modeling language used to describe and visualize system design and structure through diagrams such as class diagrams, sequence diagrams, etc.
6. AI (Artificial Intelligence): Refers to the use of machines and computers to simulate human intelligence, such as optimizing monster behavior in a game to make interactions more engaging and challenging.
7. Context Diagram: A diagram used to show the boundaries of the system and its interactions with external entities, illustrating how the system interfaces with other components or users.
8. Unity: A popular game development engine that supports cross-platform game development and is widely used for creating 2D, 3D, and virtual reality experiences.
9. C#: An object-oriented programming language commonly used for implementing game logic and features in Unity.
10. GitHub Repository: A storage and version control system for project code; the development team uses it for collaboration, code management, and sharing.
11. Item Pickup Interaction: Refers to player interactions with items in the game, such as picking up, discarding, or viewing item information.
12. Character Selection Interface: An interface that allows players to select different characters before starting the game, usually displaying character attributes and skills.
13. Scene Diversity: Refers to providing a variety of styles and themes in game scenes to enrich gameplay and increase its appeal.
14. Monster AI Behaviour: Refers to the behaviour of monsters in the game based on AI algorithms, designed to increase the challenge and interactivity of the game.

## References

Game Project requirement analysis questionnaire. (2024). *GAME Project requirement analysis questionaire*.

## Overview

This SRS document provides a comprehensive overview of both functional and non-functional requirements for the game optimization project. It covers various aspects of the system, including expected behaviours, user interface specifications, performance criteria, and external dependencies that may impact the system. The document aims to ensure that all requirements are clearly defined to guide the development process and align with user expectations.

# General Description

## System Functions

The Tarturus game system is designed to provide a challenging and immersive gameplay experience inspired by "Vampire Survivors". The core functions of the system include:

* **Survival Mechanics**: Players must manage resources, fend off waves of enemies, and navigate challenging environments.
* **RPG Elements**: Character leveling, skill upgrades, and a branching character build that responds to player choices, providing a dynamic and personalized gaming experience.
* **Narrative Integration**: The system features predefined story arcs and procedurally generated environments to create a unique and engaging narrative each time the game is played.
* **Reward System**: Recognizes player achievements and skill progression through unlockable abilities and character enhancements, encouraging continued engagement.

## User Characteristics

The system supports various user types, each with different characteristics:

* **Casual Players**: Players who are new to the survival or RPG genre and seek an entertaining experience without much learning curve. The user interface will be simple and self-explanatory.
* **Advanced Players**: Experienced players who are familiar with survival mechanics and RPG elements. The system will provide detailed customization options for advanced gameplay strategies.
* **Community Participants**: Users who contribute to the community, providing feedback, suggestions, and interacting through forums or in-game messaging.

The user interface is designed to support different interaction types, such as data entry, self-service features, and role-specific interactions, such as game customization for advanced users. The game will be compatible with multiple platforms, including:

* **Devices**: Desktop PCs and consoles.
* **Operating Systems**: Windows, macOS, and potentially consoles like PlayStation and Xbox.
* **Browsers**: For web-based components, modern browsers like Chrome, Firefox, and Edge will be supported.

This user classification will inform the Low-Level Design (LLD) phase, which includes creating tailored user experiences and platform-specific optimizations to cater to different user needs.

# General Constraints

## Software Constraints

**Licensing Requirements**: The game will use third-party libraries and engines such as Unity, which requires appropriate licenses for commercial use. All software used in development must comply with open-source or proprietary licensing agreements.

**Technical Standards**: The software must adhere to established industry standards, including compliance with gaming frameworks like Unity and best practices for game development. The code must be optimized to meet modern gaming performance standards.

**Preferred Products**: Development tools such as Unity for game development and Cubase for audio integration are preferred. Additionally, version control will be managed using GitHub to ensure efficient collaboration.

**Impact of Decisions**: Choosing Unity as the primary game engine influences compatibility, meaning the game will initially be available on platforms supported by Unity (PC and major consoles). These decisions impact the portability and extensibility of the game to other platforms in the future.

## Hardware Constraints

**Target Platforms**: The game is primarily designed for PC systems, with potential expansion to gaming consoles like PlayStation and Xbox. The hardware requirements will include a mid-range GPU and CPU to ensure smooth gameplay.

**Technical Standards**: The game must be optimized to perform well on modern gaming hardware. This includes ensuring compatibility with GPUs that support DirectX 11 or higher and maintaining acceptable performance metrics (e.g., 60 FPS on recommended hardware).

**Preferred Products**: The game should be tested on popular hardware configurations, such as NVIDIA and AMD GPUs, as well as Intel and AMD CPUs, to ensure broad compatibility and performance optimization.

**Impact of Decisions**: The decision to optimize the game for mid-range gaming PCs will affect the quality of graphics and gameplay features available on lower-end hardware. This decision aims to balance performance and accessibility for a wider audience, while ensuring a high-quality experience for players with capable systems.

# Assumptions and Dependencies

## Assumption

**Development Tools Availability**: The necessary tools for game development, such as Unity, Cubase, and GitHub, will be available throughout the project lifecycle. It is also assumed that team members have adequate experience with these tools.

**Hardware Specifications**: It is assumed that players will have access to mid-range or better hardware capable of running the game smoothly, which allows for the inclusion of enhanced graphics and performance-intensive features.

**Team Collaboration**: The development team will be able to collaborate effectively, utilizing version control tools such as GitHub, and will have consistent communication channels throughout the project duration.

## Dependencies

**Third-Party Libraries**: The project depends on several third-party libraries and APIs for functionalities such as artificial intelligence, audio processing, and user authentication. Changes in the availability or licensing of these libraries could impact the development timeline.

**Platform Support**: The game development relies on the Unity engine, which determines the platforms that can be supported (e.g., PC, consoles). Any major changes in Unity’s platform support or licensing could affect the release strategy.

**Community Feedback**: The game design and narrative elements are dependent on community feedback collected during the early research phase. This feedback will play a critical role in refining gameplay mechanics and ensuring user satisfaction.

# Functional Requirements Master List

This section shows the basic Functional Requirements and Advanced Game System.

|  |  |  |
| --- | --- | --- |
| **Req. ID** | **Requirement Name** | **Requirement Description** |
| REQ-1.1 | Character Movement and Control | Players can control character movement across the game environment using keyboard or controller inputs. |
| REQ-1.2 | Map generation and optimization | All the Maps and Props and generate Randomly and Continuously, and in order to reduce the System overhead, using the function to remove the items when invisible. |
| REQ-1.3 | Implementation of character basic attacks | Implement a system allowing players to perform basic attacks against enemies. |
| REQ-1.4 | Spawning the Monsters | Implement an enemy spawn system to introduce waves of monsters that pursue and attack the player. |
| REQ-1.5 | Game achievement system | Implement an achievement tracking system to recognize and reward player accomplishments. |
| REQ-1.6 | Game Menu and Setting | Implement a game menu with options for starting the game, loading saved games, accessing settings, and exiting. |
|  |  |  |

# Functional Requirement

# REQ-1.1

## Description

Players can control character movement across the game environment using keyboard or controller inputs. This includes directional navigation, sprinting (if applicable), and any character-specific movement abilities.

## System Input

**Keyboard Controls:** W, A, S, D keys or arrow keys for movement.

**Controller Support:** Left joystick or D-pad for directional movement.

**Additional Inputs (optional):** Shift for sprinting, spacebar for jumping (if applicable).

## Display

**In-game HUD:** Display visual indicators such as stamina (if movement includes sprinting) or a minimap for directional guidance.

**Character Feedback:** Show animations corresponding to each movement (e.g., walking, running) for visual feedback.

**Control Prompts:** Option to display control instructions for keyboard/controller in the settings or tutorial screens.

## System Processing

**Movement Physics:** Calculate the character's position and velocity based on input and apply movement smoothly within the game environment.

**Collision Detection:** Ensure characters don’t pass through walls or other objects. Handle interactions with obstacles and terrain (e.g., slopes).

**Sprint and Stamina Management:** If sprinting is enabled, deplete stamina during sprints and regenerate stamina over time when not sprinting.

**Animation Triggers:** Based on movement speed and direction, trigger the appropriate animations (e.g., walk, run, idle).

## System Output

**Character Position Update:** Continuously update the character’s location within the game world.

**Display Feedback:** Output character movement animations and adjust the camera view to follow the character.

**Status Indicators**: Update and display any movement-related status (e.g., stamina bar if sprinting)..

## Other

**Camera Control**: Optionally, the camera follows or adjusts based on the character’s movement direction for a better player experience.

## Constraints

**Platform Compatibility:** Ensure movement controls work smoothly across all intended platforms (PC, consoles).

**Hardware Performance:** Movement mechanics must perform consistently across different hardware configurations, maintaining at least 30 FPS.

## Data Handling

**Input Validation:** Only registered inputs (e.g., W, A, S, D, joystick directions) are processed for movement to prevent unintended actions.

**Data Format:** Input values are processed in standard ASCII for keyboard controls or controller signal encoding.

## Error Handling

**Input Loss:** If input is lost (e.g., disconnected controller), pause the character’s movement and notify the player.

**Invalid Input:** Ignore any unintended or unassigned keys.

**Collision Errors:** If a collision error occurs, prevent the character from moving through solid objects and reset position if necessary.

# REQ-1.2

## Description

The game map should be generated randomly and continuously to create a dynamic and varied environment. To optimize system performance, map items and props (e.g., trees, rocks) should be removed from memory once they move out of the player’s view System Input.

## System Input

**Seed/Randomization Input: Seed value or random function call to initialize map generation.**

**Player Position**: Continuously track the player’s position to determine the visible map area.

## Display

**Map Visibility**: Display map elements (terrain, props) only within the visible range around the player

**Seamless Transitions**: Ensure map tiles load and unload seamlessly as the player moves, with no visual lags or stuttering.

## System Processing

**Random Map Generation:** Generate map tiles on-the-fly based on a random seed or procedural algorithm. This generation should cover terrain, props, and any interactable items.

**Culling for Optimization:** Implement a culling function to continuously detect which props and map elements are outside of the player’s field of view.

## System Output

**Generated Map Layout:** The map layout and all generated elements are displayed based on player location and visibility.

**Active Elements**: Only elements within the player’s visible area are rendered and kept in memory, reducing overall system load.

## Constraints

**Platform Compatibility:** Ensure movement controls work smoothly across all intended platforms (PC, consoles).

**Hardware Performance:** Movement mechanics must perform consistently across different hardware configurations, maintaining at least 30 FPS.

## Data Handling

**Reusability:** When tiles go out of view, retain their procedural data for fast reloading if they become visible again.

## Error Handling

**Map Loading Failures:** If map generation fails, the system should retry with a default or backup tile layout.

**Unloading Issues:** If props or tiles do not unload properly, log these items for debugging and continue to render only visible elements.

# REQ-1.3

## Description

This feature allows the character to perform basic attacks on enemies using player input. The attacks should include animations, hit detection, and damage calculations, enabling players to interact meaningfully with enemy units.

## System Input

**Attack Input:** Key/button press by the player.

**Target Data**: Enemy positions and health status to determine if they are in the attack range.

## Display

**Map Visibility**: Display map elements (terrain, props) only within the visible range around the player

**Enemy Health Bar**: Display any change in enemy health upon successful attack hits.

**Cooldown Indicator**: Optional – display attack cooldown (if applicable) on the HUD.

## System Processing

**Attack Range Detection**: Check if enemies are within the character’s attack range when an attack is initiated.

**Damage Calculation**: Compute the damage dealt based on character stats and apply it to the enemy’s health.

**Animation Triggers**: Trigger appropriate attack animations based on the attack type and ensure smooth transitions between actions.

## System Output

**Health Reduction**: Apply damage to enemies in range and update their health.

**Attack Feedback**: Display the attack animation and sound effects.

## Constraints

**Attack Cooldown**: Implement a cooldown between attacks to prevent continuous spamming.

**Animation Timing**: Ensure attack animations align with the hit detection timing to avoid visual mismatches.

## Data Handling

**Hit Registration**: Track hit information to prevent multiple hits on a single enemy for one attack.

**Damage Scaling**: Apply damage scaling based on character or weapon level if applicable.

## Error Handling

**Cooldown Management**: Prevent attacks during cooldown periods and notify the player if necessary.

# REQ-1.4

## Description

Implement an enemy spawn system that introduces waves of monsters based on player progression. This system should allow for configuring spawn rates, enemy types, and scaling difficulty as the game advances.

## System Input

**Spawn Triggers**: Events that trigger spawning, which changes according to the elapsed time and player progress.

**Spawn Settings**: Data on enemy types, spawn rate, and locations.

## Display

**Spawn Effects**: Optionally, add visual effects (e.g., animations or smoke) when enemies spawn.

## System Processing

**Spawn Logic**: Determine spawn rates and locations dynamically based on game difficulty.

**Enemy Variety**: Randomize enemy types and quantities within each wave based on game state.

## System Output

**Active Enemies**: Display new enemies in the game world randomly.

## Other

**Spawn Zones**: Define areas where enemies can appear, avoiding spawning too close to the player.

## Constraints

**Spawn Cap**: Set a maximum number of enemies per wave to manage difficulty and performance.

## Data Handling

**Enemy Tracking**: Keep a list of active enemies to manage health, AI, and interactions.

**Wave Data**: Log wave details for tracking progression and difficulty.

## Error Handling

**Spawn Failures**: If spawning fails due to lack of space, retry after a brief delay or adjust spawn location.

# REQ-1.5

## Description

The game will track player accomplishments and unlock achievements based on specific milestones or actions. This system aims to motivate players and provide rewards for their progress and performance.

## System Input

**Player Actions and Stats**: Monitor player actions, such as defeating enemies, completing levels, or accumulating resources.

**Achievement Criteria**: Specific conditions or milestones that must be met to unlock each achievement (e.g., defeating 100 enemies, surviving for 10 minutes).

## Display

**Achievement Notification**: Show a pop-up or notification when a new achievement is unlocked.

**Achievement Menu**: A dedicated menu or screen where players can view unlocked achievements and progress toward others.

## System Processing

**Progress Tracking**: Continuously track player stats and actions relevant to achievements.

**Milestone Detection**: Check if the player’s actions meet any achievement criteria, and unlock achievements accordingly.

**Reward System**: Assign rewards (e.g., points, bonuses) upon unlocking achievements.

## System Output

**Unlocked Achievements**: Display a notification and update the achievement menu with the new achievement.

**Rewards**: Grant rewards to the player immediately after unlocking an achievement, if applicable.

## Other

**Achievement Categories**: Optionally categorize achievements (e.g., combat achievements, exploration achievements) to provide a sense of progression.

## Constraints

**Persistent Tracking**: Ensure that achievements remain unlocked across game sessions by saving data to a local file or database.

**Performance**: Track achievements without affecting game performance, especially for complex criteria.

## Data Handling

**Achievement Records**: Store data on unlocked achievements and player progress toward incomplete ones.

**Session Persistence**: Save achievement data at regular intervals and upon session end.

## Error Handling

**Progress Reset**: If progress tracking fails, prompt the player to restart the game to avoid losing data.

**Duplicate Unlocks**: Prevent achievements from being awarded multiple times for the same criteria.

# REQ-1.6

## Description

The game menu provides options for players to start a new game, load saved games, adjust settings, and exit. The settings menu allows players to configure audio, video, and control preferences.

## System Input

**Menu Navigation**: Player input (e.g., arrow keys, controller input) to navigate through menu options.

**Settings Adjustments**: Input for adjusting settings, such as sliders for volume and buttons for control mappings.

## Display

**Main Menu**: Show options for New Game, Load Game, Settings, and Exit.

**Settings Menu**: Display settings categories (Audio, Video, Controls) with sliders, toggle buttons, and dropdowns.

**Save Confirmation**: Optionally, display a confirmation dialog when saving settings changes.

## System Processing

**Menu Navigation Logic**: Process input to navigate through and select menu options.

**Settings Application**: Apply and save settings changes (e.g., update audio volume or display resolution).

**Session Management**: If applicable, manage loading of saved game data and starting new sessions.

## System Output

**Selected Menu Options**: Show the player’s selected option on the screen.

**Settings Feedback**: Reflect changes in settings (e.g., audio volume adjustment) immediately in the game.

## Other

**Settings Persistence**: Save settings so they persist between sessions.

## Constraints

**User Experience**: Ensure the menu is easy to navigate and accessible across all supported platforms.

**Performance Impact**: Minimize performance impact, especially when adjusting video settings, to avoid frame rate drops.

## Data Handling

**Settings File**: Store settings in a configuration file or database to retain adjustments across session.

## Error Handling

**Settings Reset**: Provide a "reset to default" option if custom settings cause display or performance issues.

**Menu Malfunctions**: Log any errors during navigation or selection, and allow for safe exit to the main menu.

# External Interface Requirement

Based on our previous group of customer feedback, there is a clear demand for an achievement system and high-quality music. And survey data from real potential customers and users shows that 80% of respondents either "liked" or "very much liked" having achievements or collection tasks in the game, underscoring the importance of an achievement system to enhance player engagement. Additionally, 68% of participants indicated that high-quality sound effects and background music are "very important," highlighting the need for a strong focus on audio quality. These insights have guided the refinement of functional requirements, ensuring that the game’s design aligns closely with player expectations for both gameplay mechanics and immersive audio experiences.

|  |  |  |
| --- | --- | --- |
| **Req. ID** | **Requirement Name** | **Requirement Description** |
| REQ-1.1 | Implementation of the game music. | Use software to upload and download music, create database to store music file; user could also change the different music any time in the game. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# REQ-1.1

## Data Interfaces

**Purpose**: Store all game music files, including background tracks, action cues, and victory/defeat sounds.

**Interaction**: The game retrieves specific music tracks based on in-game events.

**Data Format**: Music files store in a compressed format.

## User Interfaces

**Purpose**: Enable players to adjust game music volume or mute/unmute.

**Interaction**: Available in the settings menu, allowing players to configure their preferred audio experience.

**Display**: Show volume controls as sliders and options to toggle music on or off.

## Other Interfaces

**Interface Type**: Software interface to an audio engine, we use the Unity.

**Requirements**: Support for looping, cross-fading, and event-based triggers like starting battle music upon enemy encounter

**Error Handling**: If an audio file is missing or corrupted, the system should skip playback for that file and log an Error.

# Non-Functional Requirements

## System Performance

This section describes the system performance requirements. Example response times are suggested though different systems requirements may apply.

|  |  |  |
| --- | --- | --- |
| **Req. Id** | **Description** | **Response Time** |
| PERF-1.1 | In between levels, to maintain a seamless user experience. | 2 second |
| PERF-1.2 | Time for opening a new screen in the window. | 1 seconds |
| PERF-1.3 | Time for saving the current game data. | 0.5 seconds |
| PERF-1.4 | Maintain a minimum frame rate of 30 FPS to ensure smooth gameplay, with an ideal target of 60 FPS | / |
| PERF-1.5 |  |  |

## Information Security

This section describes the information security requirements.

|  |  |
| --- | --- |
| **Req. Id** | **Description** |
| SEC-1.1 | Player data (such as progress and settings) stored securely and encrypted. |
| SEC-1.2 | Data covered by the Data Protection Act shall not be held in environments that have not passed full system testing. |
| SEC-1.3 | Any personal data collected was complied with privacy standards, ensuring no unauthorized access. |
| SEC-1.4 |  |

## Availability

This section describes the availability requirements in terms of days and permissible planned and unplanned unavailability.

|  |  |
| --- | --- |
| **Req. Id** | **Description** |
| AVA-1.1 | The game’s services (emultiplayer servers, if applicable) should be available 99% of the time. |
| AVA-1.2 | Scheduled maintenance windows should be communicated to users in advance. |
| AVA-1.3 |  |
| AVA-1.4 |  |

## Capacity

|  |  |
| --- | --- |
| **Req. Id** | **Description** |
| CAP-1.1 | The system must be able to manage the following data volumes in year ONE.   * 5000 |
| CAP-1.2 | The system must be able to manage the following data volumes in year TWO.   * 25000 |
| CAP-1.3 | The system must be able to manage the following data volumes in year THREE.   * 100000 |
| CAP-2.1 | The system must be able to manage the following concurrent user operation volumes in year ONE.   * 800 |
| CAP-2.2 | The system must be able to manage the following concurrent user operation volumes in year TWO.   * 2500 |
| CAP-2.3 | The system must be able to manage the following concurrent user operation volumes in year THREE.   * 10000 |
| CAP-3.1 |  |