Software Design Specifications

for

COPING WITH ANXIETY USING VR 1.0

Prepared by:

Geethika Choudhary Yadlapalli
Deekshitha Karvan

Documentation and Research

Table of Contents

1 Introduction

This Software Design Specification (SDD) provides a detailed overview of the software architecture and design for the Coping with Anxiety Using VR (PC Edition) system. It outlines the structure and interaction of various software components, modules, and interfaces that collectively support the system's functionality. The document is intended to guide the development team by describing how the system will be implemented to meet the requirements defined in the Software Requirements Specification (SRS). It also incorporates relevant definitions, acronyms, references, and a comprehensive view of the system's design approach to ensure consistency, clarity, and alignment across all development activities.

1.1 Purpose

The purpose of this Software Design Specification (SDD) is to present a detailed and structured design for the Coping with Anxiety Using VR (PC Edition) system, serving as a crucial link between the Software Requirements Specification (SRS) and the system's actual development. It defines the architectural framework, component interactions, and design decisions necessary for building a functional, reliable, and maintainable application.
This document is intended to guide software developers in implementing the system, assist test engineers in creating relevant test cases, and support project managers and stakeholders in understanding the technical direction of the project. By clearly outlining the software's internal structure, this SDD ensures all team members share a consistent understanding of how the system is intended to function, supporting informed decision-making throughout the development lifecycle.

1.2 Scope

- Applies to the design and development of the Coping with Anxiety Using VR (PC Edition) system.
- Covers all software components, modules, classes, and interfaces involved in the system.
- Includes the architecture and internal design of features such as:
    - VR environment rendering and control
    - User interaction and navigation
    - Session management and activity flow

- Affects the following phases of the software development lifecycle:
    - Implementation (coding)
    - Testing and quality assurance
    - Integration and deployment
    - Maintenance and future enhancements
- Serves as a reference for:
    - **Software developers** – to implement the design accurately.
    - **Test engineers** – to design test plans and validate behavior.
    - **UI/UX designers** – to align interface designs with functional modules.
    - **Project managers and stakeholders** – to track progress and ensure design compliance with requirements.
- Ensures consistency and alignment with the Software Requirements Specification (SRS).

1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

Term	Definition
VR	Virtual Reality – A simulated experience that can be similar to or completely different from the real world, typically experienced through headsets or immersive environments.
PC	Personal Computer – A computer intended for individual use, which will be the target platform for the VR application.
SDD	Software Design Document – This document, which outlines the detailed design of the software system.
SRS	Software Requirements Specification – A document describing the functional and non-functional requirements of the system.
UI	User Interface – The graphical and interactive elements through which a user interacts with the application.

UX	User Experience – The overall experience and satisfaction a user derives from interacting with the system.
3D	Three-Dimensional – Refers to spatial environments rendered with depth (x, y, z axes), commonly used in VR applications.
SDK	Software Development Kit – A collection of software tools and libraries used to develop applications for specific platforms (e.g., Oculus SDK, SteamVR SDK).
FPS	Frames Per Second – A measure of how smoothly graphics are rendered. A higher FPS is critical for reducing motion sickness in VR.
Latency	The delay between a user's action and the system's response. Low latency is essential for an immersive VR experience.
Controller Input	Signals or commands given by the user via hand-held VR controllers, which influence interactions within the virtual environment.
Head Tracking	A VR feature that tracks the user's head movements to adjust the view in the virtual environment accordingly.
Positional Tracking	Technology that tracks a user's position in physical space and translates it into the virtual environment.
FOV	Field of View – The extent of the observable environment at any given time within the VR headset.
Immersion	The sensation of being physically present in a non-physical world, a key goal of VR experiences.
Module	A self-contained unit of software that performs a specific function within the overall system.
Component	A logical piece of the system architecture that may contain one or more modules or functionalities.
State Machine	A computational model used to manage different states and transitions within interactive systems.

Term / Acronym	Definition
Unity	A cross-platform game engine used for developing 2D, 3D, AR, and VR applications.

Scene	A container in Unity where all environment objects, assets, and behaviors for a specific level or part of the application are placed.
GameObject	The base class for all entities in a Unity scene; everything visible or interactive in Unity is a GameObject.
Component	A functional unit that can be attached to GameObjects to give them behavior or properties (e.g., Collider, AudioSource, Script).
Transform	A component that defines an object's position, rotation, and scale in 3D space.
MonoBehaviour	The base class from which every Unity script derives, enabling interaction with Unity's event-driven engine (e.g., Start(), Update()).
Collider	A component that defines the shape of an object for physical collisions or triggers; often used for detecting interactions.
Script	A C# file attached to a GameObject to define custom behavior and logic.
Animator	A system used to control animations using parameters and states defined in an Animator Controller.
Animation Clip	A file containing keyframe-based animation data for objects or characters.
Material	Defines how a surface appears visually, including texture, color, and shader properties.
Shader	A program that runs on the GPU to determine how surfaces are rendered visually.
Lighting	The system in Unity that simulates how light interacts with objects, affecting mood and realism.
NavMesh	A navigation mesh used for AI pathfinding and movement in Unity scenes.
Build Settings	Configuration panel in Unity that allows you to define the target platform (e.g., PC, Android) and scenes to include in the final build.
Play Mode	The mode in the Unity Editor used to test the application in real-time without building.

Inspector	The Unity panel where you can view and modify the properties of selected GameObjects or components.
Hierarchy	The panel that shows the structure and parent-child relationships of all GameObjects in the current scene.
Project Panel	Displays all assets, scripts, models, audio files, and resources in the Unity project.
Asset	Any file used in the Unity project (e.g., texture, audio, model, script).
Canvas	The root component for rendering UI elements in Unity's UI system.
Event System	Manages input and interaction for UI elements, including clicks, drags, and other user actions.
Coroutine	A Unity feature that allows execution of code over time without blocking the main thread—commonly used in breathing exercise timing.

1.4 References

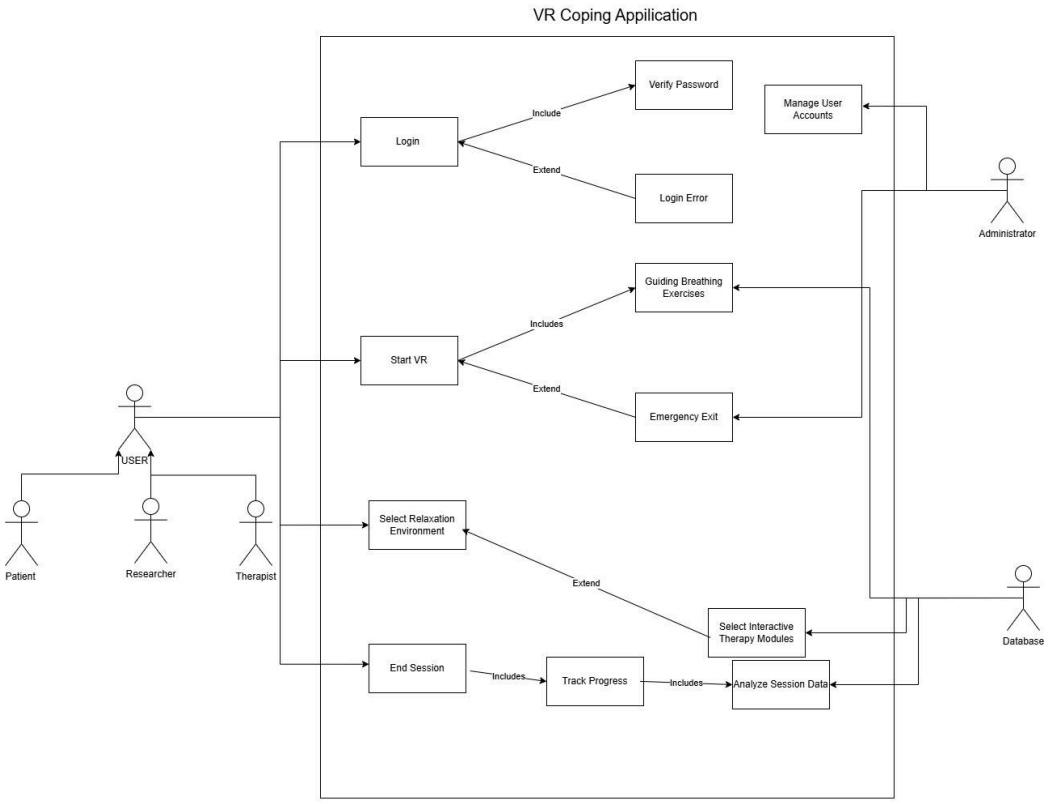
Unity Documentation: https://docs.unity3d.com/
SteamVR Developer Docs: https://steamvr.studio/
UML 2.5 Specification: OMG (Object Management Group)
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2924288/
https://mental.jmir.org/2023/1/e44998/
https://arxiv.org/abs/2105.10756
https://docs.unity3d.com/Manual/com.unity.xr.oculus.html

2 Use Case View

The use case diagram for the **VR Coping Application (PC Edition)** outlines key interactions between the system and its primary users—**patients**, **therapists**, and **researchers**. Users begin by **logging in**, which includes password verification and error handling. An **administrator** manages user accounts.

After login, users can **start a VR session**, which includes **guided breathing exercises** and allows for an **emergency exit** if needed. They can then **select a relaxation environment**, which may lead to **interactive therapy modules**.

At the end of a session, users can **end the session**, which includes **tracking progress** and **analyzing session data**. The **database** stores all relevant user activity and feedback. This structure supports therapeutic engagement and research insights in managing anxiety through VR.



3 **Design Overview**

The VR Coping Application is built with a modular architecture focusing on clear separation of concerns. It features a user-friendly VR menu for environment selection, each linked to specific breathing exercises and background audio. Core modules handle menu navigation, session control, environment loading, and optional user management. This design ensures easy maintenance and future scalability while meeting current functional requirements

3.1 Design Goals and Constraints

The primary design goal is to create an immersive and user-friendly VR application that helps users cope with anxiety through guided breathing exercises, calming environments, and interactive therapy modules. The system must support multiple user roles (patients, therapists, researchers) and ensure smooth VR session flow with real-time feedback and progress tracking. Key constraints include cross-platform compatibility with PC-based VR systems, integration with existing user databases, adherence to data privacy standards, and a modular development approach for scalability. The project must align with a tight development schedule and use Unity for VR development and OpenXR and oculus for testing of the VR application

3.2 Design Assumptions

The design assumes that users will have access to a compatible PC and VR headset. Therapists and researchers are expected to have basic familiarity with VR controls.

3.3 Significant Design Packages

Design Hierarchy

Layer Name	Components / Modules	Description
Hardware Abstraction Layer	`InputHandler`, `Meta Quest Link`, `SteamVR/OpenXR`	Interfaces with VR hardware and standard APIs to handle input and device I/O.
System Layer	`Core`, `PerformanceMonitor`	Provides foundational utilities, configuration management, and runtime metrics.
Logic Layer	`SessionController`, `AudioManager`, `JournalManager`, `EnvironmentManager`	Implements application logic for guided therapy sessions and environment control.
Interface Layer	`UI` (Unity Canvas / World Space Elements)	Manages user-facing components, including menus, prompts, and

interaction cues.

3.4 Dependent External Interfaces

The table below lists the public interfaces this design requires from other modules or applications.

External Application and Interface Name	Module Using the Interface	Functionality/ Description
Meta Quest Link	Headset Streaming Bridge	Allows Meta Quest to connect to PC via USB or Air Link, enabling high-performance PC VR rendering and interaction.
Unity Input System	Interaction Module	Detects controller input, gaze, and click interactions for journaling and game elements.
Unity OpenXR Plugin	VR Interaction Engine	Provides abstraction over VR headset hardware APIs for input tracking, rendering, and haptic feedback.

3.5 Implemented Application External Interfaces (and SOA web services)

The table below lists the implementation of public interfaces this design makes available for other applications.

Interface Name	Implementing Module	Functionality / Description
Audio Control Interface	Audio Playback Manager	Exposes methods for play/pause, volume control, and ambient audio replacement at runtime.

Guided Session Controller	Meditation Module	Provides structured session flow, integrating narration, visuals, and breathing animations.
Environment Loader	Scene Manager Module	Enables switching between VR environments (beach, forest, etc.) via function calls or config.

4 Logical View

This section outlines the internal structure of the VR Coping Application based on its current features. The system employs a modular design with clear separation of concerns to deliver an immersive VR experience focused on environment selection and guided breathing exercises.

On startup, the application loads a **main menu interface** that lets users choose from multiple virtual therapy environments. Each environment is associated with a unique breathing exercise and corresponding background audio to create a calming and therapeutic atmosphere.

The system consists of core modules handling menu display, environment loading, exercise guidance, and audio playback. Data handling is minimal and centralized for future extensibility.

4.1 Design Model

The app consists of these main modules and their classes:

- **Menu and Environment Selection Module**
  Classes: `MenuController`, `EnvironmentSelector`
  Responsibilities: Display main menu, allow users to browse and select therapy environments.

- **Therapy Session Module**
  Classes: `SessionManager`, `EnvironmentLoader`, `BreathingExerciseGuide`, `AudioPlayer`
  Responsibilities: Load selected environment, start the associated breathing exercise, play background audio, and control session flow.

- **User Management Module**
  Classes: `UserAdmin`, `AccountManager`
  Responsibilities: Administer user accounts, control access permissions.

These modules communicate through well-defined interfaces and use a centralized `DataService` class for consistent data operations.

4.2 Use Case Realization

Load Menu and Select Environment:
`MenuController` displays the menu → User selects environment via `EnvironmentSelector` → control passes to `SessionManager`.

Start Therapy Session:
`SessionManager` triggers `EnvironmentLoader` to load the chosen environment → `AudioPlayer` starts background audio → `BreathingExerciseGuide` initiates the guided breathing exercise.

5 Exception Handling

This section describes exceptions defined within the VR Anxiety Management Application, the conditions under which they are triggered, how they are handled, how they are logged, and the follow-up actions required.

5.1 Defined Exceptions and Handling Mechanisms-

Exception Name	Trigger Condition	Handling Strategy	Logging Details	Follow-up Action
MissingAssetException	An audio file, texture, or environment prefab is not found at runtime	Load default fallback asset (e.g., default ambient audio or neutral scene)	Log asset name and missing path in `log.txt`	Developer to verify asset bundle or scene linking
DeviceNotConnected Exception	VR headset is not detected or disconnected mid-session	Display overlay with prompt to reconnect headset; pause environment	Log timestamp, device ID (if available), and connection status	User reconnects device; app resumes automatically
SceneLoadFailure Exception	VR scene fails to load due to missing data or corrupt files	Return to main menu; notify user with	Scene name, stack trace, Unity error	Developer inspects asset integrity and

		friendly message	message	Unity scene setup
ConfigLoadException	Error reading or parsing configuration parameters from settings file	Use internal defaults for affected parameters; suppress further config errors	Log specific parameter, file path, and fallback value used	Developer to correct malformed config file
InputDeviceFailure Exception	VR controller or input tracking is unresponsive or disconnected	Pause input-dependen t interactions; notify user via in-VR alert	Log device name and failure details	User re-pairs device; system resumes tracking
UnhandledApplication Exception	Any unexpected exception that is not otherwise caught	Catch globally; show safe error screen or return to main menu	Full stack trace, active scene, memory usage snapshot	Developer inspects logs and implements specific catch
PerformanceDrop Warning	Frame rate drops below 30 FPS for sustained period (>5 sec)	Lower graphics settings dynamically; notify user only in debug mode	FPS values, active scene, active object count	Optimize affected scene or reduce load on GPU

5.2 Logging Mechanism

- All exceptions are logged into a centralized `log.txt` file stored locally within the application directory.
- Each log entry contains:
    - **Timestamp**
    - **Exception name and message**
    - **Stack trace (if applicable)**

- **Scene/environment details**
- **User context (e.g., environment selected, device model)**

5.3 Follow-up Actions

- Minor exceptions (e.g., missing ambient sound) require no immediate user intervention but are flagged for **developer review**.
- Critical exceptions (e.g., scene load failure or device disconnect) trigger **non-intrusive user notifications** with safe fallback or recovery paths.
- Logs are reviewed regularly during testing; optionally users can be prompted to export logs for bug reports in future updates.

6 Configurable Parameters

This table describes the simple configurable parameters (name / value pairs)

Configuration Parameter Name	Definition and Usage	Dynamic?
render_quality_level	Controls the visual fidelity settings for performance adaptation	No
environment_selection	Specifies which VR environment to load (e.g., beach, forest, waterfall)	Yes
meditation_duration	Sets the length of guided meditation sessions (in minutes)	Yes
ambient_audio_volume	Controls the volume level of background ambient music and nature sounds	Yes

7 Quality of Service

The VR Coping Application is designed as a standalone, locally executed system optimized for smooth and reliable performance on mid-to-high-end PCs with VR headsets. It ensures high

availability by functioning independently of network connectivity, with fast startup and environment loading times.

Security is minimal in the current single-user setup, with plans for enhanced authorization and data protection in future multi-user versions. Performance targets prioritize a steady 60 FPS frame rate and low audio latency to maintain user immersion. Optimization techniques like LOD models and dynamic rendering are employed to support a range of hardware.

The system operates without background services or external monitoring, simplifying control and minimizing resource usage.

7.1 Availability

- The VR application is designed for **local execution** on a Windows PC with a VR headset (e.g., Oculus in Link mode).
- Since the system is standalone, **network outages** will not affect availability.
- Application startup and environment switching are optimized to occur within **10 seconds**.
- Mass updates or environmental asset changes may require **app restarts** or reloading.
- All **journal entries are stored locally**, and no external server is required for core functionality, improving uptime.

7.2 Security and Authorization

As this is a **locally deployed, single-user therapeutic app**, minimal authorization is required.

No user authentication is implemented in the current version.

Future expansions into **multi-user environments or metaverse transitions** will require:

- OAuth-based user login
- Role-based feature access (admin/user/guest)
- Encrypted journal storage and secure cloud sync options

In current design, **user data is stored locally** and not transmitted externally.

7.3 Load and Performance Implications

The application is designed to run on mid-range to high-end PCs with discrete GPUs (GTX 1060 or higher).

Target performance:

- **Frame rate**: ≥ 60 FPS for a smooth and nausea-free VR experience.
- **Audio latency**: < 50ms to maintain immersion during guided sessions.

Performance is optimized through:

- Use of LOD (Level of Detail) models
- GPU instancing for repeated assets (e.g., trees, grass)
- Dynamic render scaling for weaker hardware

No concurrent user handling is needed.

7.4 Monitoring and Control

The application does not include external daemons or background processes.

Internally monitored metrics:

- FPS logging (displayed on debug HUD in test mode)
- Environment load time
- Audio playback consistency (via Unity's AudioSource API)

Control interfaces:

- In-app debug panel (visible only to testers)
- Adjustable parameters through settings menu (with real-time updates where supported)

AUTHORS:
Aditya Rachakonda - Project Lead
P V Kesava Rao - Environment Designer
Revanth Reddy Kancherla - Quality Tester
Geethika Choudhary Yadlapalli - Documentation & Research Lead
Deekshitha Karvan - Documentation and Environment Assistance