

The Game Room

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 10/07/24 | Nathan Sakhichand | Modifications were implemented to the cover page, document revision history, executive summary, design constraints, system architecture view, domain model, and recommendations. |

## [Executive Summary](#_sbfa50wo7nsh)

The purpose of this software design document is to articulate the proposed solution for developing the web-based game application "Draw It or Lose It" for our client, The Gaming Room. This game draws inspiration from the beloved TV game show Win, Lose, or Draw, where teams compete in a fun and interactive format to guess what is being illustrated. The proposed solution involves utilizing a library of stock drawings to provide visual cues, enabling multiple teams to engage in an exciting four-round gameplay format. Transitioning to a web-based platform will ensure that the game is accessible on various devices, from desktops to tablets and smartphones, thereby enhancing the user experience and broadening the potential audience.

To effectively implement this solution, it is crucial to address several key factors: ensuring a robust architecture that supports seamless multiplayer interactions, implementing mechanisms for unique identifiers for games and teams to prevent conflicts, and designing an intuitive user interface that fosters engagement and enjoyment. Additionally, the development process will prioritize scalability to accommodate future growth and potential feature enhancements, such as leaderboards and user profiles.

For The Gaming Room to proceed with this project, it is important to have access to existing game assets, including artwork and branding materials, as well as any specific requirements or preferences regarding the user experience. This information will be vital in ensuring that the final product aligns with The Gaming Room's vision and meets the expectations of its target audience. Overall, the proposed design aims to create a user-friendly, engaging, and innovative gaming experience that leverages modern web technologies to captivate players.

## Requirements

1. Web-Based Accessibility: The game must be developed as a web-based application, ensuring compatibility across various devices and operating systems, including desktops, laptops, tablets, and smartphones.

2. Team Participation: Each game session should support multiple teams, with the capability to assign several players to each team. This flexibility will enhance the competitive aspect of the game.

3. Unique Naming Protocol: To avoid conflicts, both game and team names must be unique. The application should include a feature that allows users to check the availability of names during the team creation process.

4. Single Game Instance: The application must enforce a design constraint that allows only one instance of the game to exist in memory at any time. This can be achieved through the implementation of unique identifiers for each game, team, and player, ensuring clarity and consistency in game management.

5. Timed Game Rounds: Each round of the game should have a predetermined time limit, such as one minute. Additionally, drawings should be progressively revealed, with the complete image displayed at the 30-second mark to maintain engagement and excitement.

6. Guessing Mechanism for Remaining Teams: If a team is unable to guess the drawing within the allotted time, the remaining teams should be given the opportunity to make one guess each within a 15-second time limit. This feature encourages participation and keeps the gameplay dynamic.

## [Design Constraints](#_2et92p0)

The game application is required to be developed for a web-based platform, which introduces specific constraints pertaining to network communication, security protocols, and compatibility across various web browsers and devices. This necessitates careful consideration of how data is transmitted over the internet, ensuring robust security measures are in place to protect user information, and implementing responsive design techniques to ensure seamless functionality on a wide range of devices.

The system must enforce a strict policy regarding the uniqueness of game, team, and player names. This is crucial to prevent naming conflicts and enhance the overall user experience during the processes of creating and joining games. Implementing a name availability checking feature will allow users to verify if their desired names are already in use, thus streamlining the onboarding process and reducing potential confusion.

To ensure the game functions as intended, the design must incorporate a limitation that permits only one instance of the game service to be active in memory at any given time. This restriction is essential for maintaining the integrity of game sessions and preventing conflicts that may arise from multiple instances attempting to manage the same game state. Implementing unique identifiers for each game, team, and player will facilitate this requirement and ensure that the game operates smoothly and reliably.

## [System Architecture View](#_ilbxbyevv6b6): : N/A

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram provided illustrates a structured design for managing games, teams, and players within a software application. At the top of the hierarchy is the ProgramDriver class, which serves as the entry point for the application, containing the `main()` method that initiates the program's execution. It interacts with the SingletonTester class, as indicated by the <<uses>> relationship, to test or validate the singleton behavior of other components, particularly the `GameService` class. The Entity class acts as a base class, providing common attributes like `id` and `name`, and methods such as `getId()`, `getName()`, and `toString()`. This promotes code reuse and consistency across its subclasses, which include the `Game`, `Team`, and `Player` classes, all inheriting these shared characteristics.

The GameService class is pivotal, managing game instances and implementing the singleton pattern to ensure only one instance exists, thereby centralizing game management and preventing conflicts. It has a one-to-many relationship with the Game class, signifying its capability to oversee multiple game instances. Within each Game class, there's a composition relationship with the Team class, as each game can comprise multiple teams. Similarly, the Team class contains a list of Player objects, illustrating a whole-part relationship where teams consist of various players. These relationships embody several object-oriented programming principles such as encapsulation, inheritance, and composition, efficiently fulfilling the software requirements. Encapsulation protects data within classes, inheritance promotes reusability, and composition allows for flexible and modular design, all contributing to a robust and scalable application architecture.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | **Characteristics:**  - Unix-based operating system with a strong focus on security and stability  - Supports a wide range of open-source software and development tools  - Known for its high performance and reliability in server environments  **Advantages:**  - Enhanced security through built-in firewalls, sandboxing, and code signing  - Cost-effective as it does not require licensing fees for servers  - Extensive community support and documentation for troubleshooting and optimization  **Weaknesses:**  - Limited hardware compatibility compared to Windows or Linux  - Fewer commercial software options available compared to Windows  - Requires more technical expertise to manage and configure than Windows or Linux | **Characteristics:**  - Open-source and highly customizable, allowing for tailored configurations and flexibility.  - Known for its stability, security, and reliability, minimizing downtime and data breaches.  - Supports a wide range of hardware, architectures and software applications, providing compatibility and scalability options.  **Advantages:**  - Cost-effective and scalable solution, offering a low-cost alternative to proprietary operating systems.  - Robust security features, including encryption, access controls, and intrusion detection systems, enhance data protection.  - Extensive community support and documentation, facilitating troubleshooting and knowledge sharing.  **Weaknesses:**  - Limited user interface and graphical tools, requiring technical expertise for management and configuration.  - Compatibility issues may arise with certain software applications and hardware devices.  - Requires regular updates and maintenance to ensure stability and security. | **Characteristics:**  - Windows is one of the most widely used operating systems in the world, with a large installed base. This makes it a good choice for hosting web-based applications that need to reach a broad audience.  - High performance: Windows is a high-performance operating system that can support demanding web-based applications.  - Scalability: Windows can be scaled to support large web-based applications with high traffic volumes.  - Security: Windows has a strong security track record and offers a variety of security features to protect web-based applications.  - Cost-effective: Windows is a relatively cost-effective operating system, making it a good choice for businesses on a budget.  **Weaknesses:**  - Can be complex to administer.  - Windows is not as open source as other operating systems, which can limit customization and flexibility.  - Can be susceptible to malware. | **Characteristics:**  - Compact and Portable  - Touchscreen Interface: Intuitive and user-friendly, enabling easy navigation and input.  - Wireless Connectivity: Seamless access to the internet and network resources.  - Limited Processing Power: Compared to laptops or desktops, mobile devices have lower CPU and memory capabilities.  -Battery Life: Limited battery life can restrict usage time.  **Advantages:**  - Mobility: Enables access to applications on the go, providing flexibility and convenience.  - Location Awareness: Mobile devices can leverage GPS to provide location-based services within applications.  - Camera and Microphone: Built-in cameras and microphones allow for image and audio capture, enabling features like video conferencing.  **Weaknesses:**  - Limited Screen Size: Small screen size can restrict the display of complex content and make navigation challenging.  - Limited Input Options: Touchscreen keyboards can be less efficient than physical keyboards.  - Security Concerns: Mobile devices may be more vulnerable to security threats due to their wireless mature.  - Data Storage Limitations: Storage capacity on mobile devices is limited, which can impact the size of applications and data that can be stored. |
| **Client Side** | **Cost**:  - Development time: Developing and maintaining software for multiple clients can be time-consuming and costly, especially if the software is complex or requires extensive customization.  - Infrastructure: Supporting multiple clients may require additional infrastructure, such as servers, databases, and storage, which can increase costs.  - Testing: Testing software across different client platforms and configurations can be complex and time-consuming, adding to the overall cost.  **Time:**  -Development: Developing software that is compatible with multiple clients can take longer than developing for a single platform.  - Testing: Testing the software thoroughly across all target platforms can be a lengthy process.  - Maintenance: Maintaining software for multiple clients can be time-consuming, as updates and fixes may need to be implemented across all platforms.  **Expertise**  - Platform knowledge: Developers need to have a deep understanding of the Mac platform, as well as the specific requirements of each client.  - Cross-platform development: Developers should be experienced in cross-platform development techniques, such as using frameworks or libraries that support multiple platforms.  - Device management: Supporting multiple clients may require expertise in device management, such as configuring and deploying software updates. | **Cost**:  - Licensing: Linux is open-source, eliminating licensing costs. However, additional software or services required for client support may incur costs.  - Development: Developing and maintaining software to support multiple clients can be resource-intensive, requiring additional staff or outsourcing.  - Infrastructure: Scaling infrastructure to support many clients can lead to increased hosting and maintenance costs.  **Time:**  - Development time: Designing and implementing software that seamlessly supports different client types can be time-consuming.  - Testing and debugging: Ensuring compatibility and functionality across multiple clients require thorough testing and debugging, which can delay project completion.  - Maintenance and updates: Ongoing maintenance and updates to software are necessary to address issues and ensure compatibility with evolving client requirements.  **Expertise:**  - Linux proficiency: Developers must have a strong understanding of Linux system administration, networking, and software development.  - Software engineering skills: Developing robust and scalable software solutions requires advanced software engineering skills, including design patterns, testing methodologies, and debugging techniques.  multiple clients.  supported clients. | **Cost:**  - Licensing: Acquiring Windows licenses for multiple clients can be a significant expense.  - Development tools: Developing software for multiple clients may require additional development tools or frameworks, which can add to the cost.  - Testing and maintenance: Ensuring compatibility with multiple client types requires extensive testing and ongoing maintenance, which can be time-consuming and costly.  **Time:**  -Development: Implementing support for multiple clients can significantly increase the development time, as it requires additional coding and testing.  - Testing: Thorough testing is essential to ensure compatibility and avoid bugs. This can be a time-consuming process, especially for complex software.  - Maintenance: Maintaining compatibility with multiple client types requires ongoing updates and patches, which can take additional time.  **Expertise:**  - Windows development: Developers need to have a strong understanding of Windows development principles and technologies.  - Multi-client support: Developing software that supports multiple clients requires expertise in cross-platform development and compatibility testing.  - Testing: Effective testing requires knowledge of different client types and their specific requirements. | **Cost**:  -Development Costs: Creating apps for multiple platforms (e.g., iOS, Android) requires separate development efforts, increasing overall costs.  - Maintenance Costs: Ongoing support and updates for multiple platforms can be substantial.  - Infrastructure Costs: Hosting and managing apps for different platforms may require additional infrastructure and resources.  **Time:**  - Development Time: Developing apps for multiple platforms takes more time than creating a single app.  - Testing and Deployment Time: Testing and deploying apps on different platforms can be a time-consuming process.  - Update Time: Maintaining and updating apps for multiple platforms requires ongoing effort.  **Expertise:**  - Platform-Specific Knowledge: Developing for multiple platforms requires expertise in different programming languages, tools, and development environments.  - Integration Skills: Integrating apps with different operating systems and devices requires specialized knowledge.  - Cross-Platform Expertise: Using cross-platform development frameworks can reduce expertise requirements but may limit customization options. |
| **Development Tools** | - Swift: Apple's primary programming language for macOS development.  - C++: Used for high-performance applications and system programming.  - Python: Used for scripting, data analysis, and machine learning tasks.  **IDEs:**  - Xcode: Apple's official IDE for macOS development, providing a comprehensive set of tools and features.  - Visual Studio Code: A popular cross-platform IDE with a wide range of extensions for macOS development.  - Atom: A lightweight and customizable IDE with a growing community of plugins.  - Sublime Text: A minimalist and extensible text editor with support for macOS development.  **Other Tools:**  - Interface Builder: A graphical user interface (GUI) design tool for creating macOS apps. | - C/C++: Widely used for developing high-performance system software and embedded systems.  - Python: A versatile language suitable for scripting, data analysis, and web development.  - Java: A popular language for enterprise applications, Android development, and server-side programming.  **IDEs:**  -Visual Studio Code: A cross-platform IDE with support for various programming languages, including C/C++, Python, and Java.  - Eclipse: An open-source IDE primarily used for Java development but also supports other languages.  - PyCharm: A professional IDE specifically designed for Python development.  **Other Tools:**  - Auto tools: A set of tools for configuring and building software on Unix-like systems. | - C#: Primary language for developing Windows applications.  - C++: Can be used for high-performance or low-level applications.  - Python: Can be used for scripting and automating tasks.  - Java: Can be used for cross-platform development, including Windows.  **IDEs:**  - Visual Studio: Microsoft's official IDE for Windows development.  - JetBrains Rider: A cross-platform IDE from JetBrains that supports C# and other languages.  - Eclipse: An open-source IDE that supports Java and other languages.  **Other Tools:**  - .NET Framework: A software framework for developing and deploying Windows applications.  - Windows SDK: A set of tools and libraries for developing Windows applications.  - Windows Installer (MSI): A package format for distributing and installing Windows software. | - Swift: iOS applications  - Kotlin: Android applications  - Java: Android applications (legacy)  - C#: Cross-platform mobile development using Xamarin  - JavaScript: Hybrid mobile development using React Native, Ionic, or Cordova  **IDEs:**  -Xcode: For iOS development  - Android Studio: For Android development  - Visual Studio: For C# and Xamarin development  - AppCode: For iOS and macOS development (alternative to Xcode)  **Other Tools:**  - Mobile Device Management (MDM) software: For managing and securing mobile devices  - Mobile testing frameworks: Such as Appium or Espresso  - Mobile analytics tools: Such as Google Analytics or Firebase Analytics |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: Recommend Linux as the operating platform due to its open-source nature, cross-platform compatibility, and support for various programming languages. This allows The Gaming Room to expand Draw It or Lose It to different computing environments, including Windows, macOS, and mobile devices.
2. **Operating Systems Architectures**: Linux uses monolithic kernel architecture, where the kernel and device drivers are tightly integrated. This provides high performance and stability. Additionally, Linux supports virtualization technologies (ex. Docker, Kubernetes) that enable isolated and portable software environments.
3. **Storage Management**: ZFS is a recommended storage management system for Linux. It provides advanced features such as data integrity verification, snapshotting, and data compression, ensuring data reliability and efficient storage utilization.
4. **Memory Management**: Linux uses a virtual memory system that allows it to manage physical memory more efficiently. It employs page-based memory management, where memory is divided into pages that can be swapped in and out of physical memory as needed. This optimizes memory usage and allows for smooth execution of the Draw It or Lose It software.
5. **Distributed Systems and Networks**: To enable communication between various platforms, Draw It or Lose It can be developed as a distributed system using client-server architecture. The client application runs on user devices and communicates with a central server that hosts the game logic and data. TCP/IP is a suitable network protocol for establishing reliable communication channels between the clients and server.

Dependencies between components include:

- Connectivity: Clients require stable network connections to the server.

- Outages: Server outages can disrupt gameplay and require failover mechanisms.

1. **Security**:

User Protection:

- Encryption: Data should be encrypted at rest and in transit using strong encryption algorithms (ex. AES-256).

- Authentication and Authorization: Implement mechanisms to verify user identity and control access to sensitive data.

Security Capabilities of Linux:

- SELinux (Security Enhanced Linux): Provides mandatory access control to enforce security policies.

- AppArmor: Similar to SELinux, it restricts process capabilities to enhance application security.