

**Software Design Document for Draw It or Lose It**

# **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 | 02/09/2025 | Donna Evers | Initial Draft |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has tasked Creative Technology Solutions (CTS) with developing a web-based version of their popular game, *Draw It or Lose It*. This game must serve multiple platforms while preserving the integrity of the gameplay and ensuring scalability. The proposed solution leverages object-oriented design principles, efficient development patterns (e.g., singleton and iterator), and platform-agnostic technologies to create a scalable and robust application that meets client requirements. This document outlines the design constraints, architecture, and the domain model for the application.

## Requirements

The Gaming Room’s primary requirements include:

* The ability to support one or more teams within a game, with each team having multiple players.
* Unique names for games, teams, and players.
* Only one instance of the game can exist in memory at any given time.
* Users must be able to verify name availability before assigning team names.

## [Design Constraints](#_2et92p0)

**Platform Constraints**

The game must be web-based and accessible across multiple platforms, including Windows, macOS, Linux, and mobile devices. This requires a distributed architecture that supports various client types while maintaining seamless functionality.

**Performance Constraints**

The application must efficiently handle multiple concurrent users. This necessitates implementing efficient data structures and algorithms to manage teams, players, and game state.

**Security Constraints**

The system must protect user information and ensure secure communication across platforms. Measures like encryption and authentication protocols are required to safeguard data integrity and privacy.

**Scalability Constraints**

The architecture must be designed to scale horizontally and vertically to accommodate future growth, such as adding more players or integrating new game features.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

**UML Class Diagram Description**

The UML class diagram for *Draw It or Lose It* consists of the following core classes:

1. **Entity Class:**
   * Base class with common attributes id (long) and name (String).
   * Common methods include getId(), getName(), and toString().
2. **GameService Class:**
   * Manages all games, teams, and players.
   * Implements the singleton pattern to ensure only one instance exists.
   * Provides methods for adding and retrieving games, teams, and players.
3. **Game Class:**
   * Represents a game instance with a list of teams.
   * Inherits from the Entity class.
   * Methods include addTeam() and toString().
4. **Team Class:**
   * Represents a team within a game, with a list of players.
   * Inherits from the Entity class.
   * Methods include addPlayer() and toString().
5. **Player Class:**
   * Represents a player within a team.
   * Inherits from the Entity class.

**Object-Oriented Principles Demonstrated**

* **Encapsulation:** Attributes like id and name are private, with public getter methods to access them.
* **Inheritance:** Common functionality is abstracted into the Entity base class, reducing redundancy.
* **Polymorphism:** Methods like toString() can be overridden in child classes to provide specific implementations.
* **Design Patterns:**
  + Singleton Pattern: Ensures a single instance of GameService exists.
  + Iterator Pattern: Used to traverse and manage collections like games, teams, and players efficiently.

**"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.

Development Requirements

Mac:

- Server Side: macOS is well-integrated with Unix-based tools, but it offers limited options for enterprise hosting. Its reputation for stability, security, and scalability makes it a popular choice for hosting web applications, supported by a strong open-source community.

Linux:

- Server Side: Linux is widely utilized in enterprise environments and works effectively with Microsoft-based enterprise tools. However, licensing fees for Windows Server can be high. Additionally, Linux is primarily focused on client-side applications, limiting its hosting capabilities.

Windows:

- Server Side: Windows Server is favored for enterprise solutions, though the associated licensing costs can be significant.

Mobile Devices:

- Mobile devices are typically not used as hosts for web applications.

Client Side

- Mac: While it offers excellent compatibility for development tools, it often comes with higher hardware costs. The platform excels in providing strong support for creative and development tools such as Xcode and IntelliJ IDEA.

- Linux: This platform features low-cost, open-source tools; however, compatibility can vary among distributions, which grants it flexibility for development.

- Windows: Windows provides high compatibility with a broad array of development tools; however, it incurs licensing fees and may entail increased system overhead. Platform-specific adaptations are required for effective development.

- For mobile development, iOS requires Xcode and macOS, while Android development can be accomplished with Android Studio, which is cross-platform.

Development Tools

- For Mac: Development tools such as Xcode, IntelliJ IDEA, and Eclipse are necessary for iOS development.

- For Linux: IntelliJ IDEA, Eclipse, and NetBeans offer robust support for open-source development environments.

- For Windows: IntelliJ IDEA, Eclipse, and NetBeans integrate seamlessly with Microsoft Visual Studio for building Windows applications.

- Cross-Platform: Android Studio is used for Android development, while Xcode is essential for iOS development. Cross-platform frameworks like Flutter and React Native can help streamline the development process.

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** | Reliable for hosting; macOS integrates well with Unix-based tools. | Known for stability, security, and scalability. | Widely used; integrates well with enterprise tools. | Limited for hosting; primarily client-side support. |
| **Client Side** | High compatibility for development but higher cost. | Low-cost, open-source tools available. | High compatibility; requires licensing fees. | Requires platform-specific adaptations. |
| **Development Tools** | Xcode, IntelliJ IDEA, Eclipse. | IntelliJ IDEA, Eclipse, NetBeans | IntelliJ IDEA, Eclipse, NetBeans | Android Studio, Xcode for iOS. |

## 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:**  
   Linux is recommended as the server operating system for its stability, scalability, and security. It is widely used for web-based applications and can efficiently handle distributed system architectures.
2. **Operating Systems Architectures:**  
   The system architecture will follow a client-server model with RESTful APIs for cross-platform communication. A cloud-based infrastructure, such as AWS or Google Cloud, will be used to ensure high availability.
3. **Storage Management:**  
   A combination of relational and non-relational databases, such as MySQL and MongoDB, will be used for structured and unstructured data. Cloud-based storage solutions, such as AWS S3, will handle media assets efficiently.
4. **Memory Management:**  
   The recommended Linux-based platform will use memory management techniques such as paging, caching, and virtual memory allocation to optimize performance. Load balancing will ensure efficient memory utilization across servers.
5. **Distributed Systems and Networks:**  
   A microservices-based architecture with containerization (Docker, Kubernetes) will allow seamless communication between various platforms. Content Delivery Networks (CDNs) and edge computing will reduce latency and improve connectivity.
6. **Security:**
   * SSL/TLS encryption for secure communication.
   * OAuth and JWT authentication for user verification.
   * Data encryption for stored user information.
   * Regular security audits and updates to protect against vulnerabilities.

By implementing these recommendations, *Draw It or Lose It* will be well-equipped for expansion across multiple platforms while ensuring security, performance, and scalability.