

<Name-of-Software-Application>

# **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 | <mm/dd/yy> | <Your-Name> | <Brief description of changes in this revision> |

**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 seeks to expand its Android-based game "Draw It or Lose It" to a web-based version supporting multiple platforms. This game mimics the television show "Win, Lose or Draw" by using stock drawings for players to guess within timed rounds. The goal is seamlessly transitioning to a web-based environment, retaining the game's unique aspects while enhancing accessibility.

To achieve this, the game will be developed as a web application using Java, incorporating object-oriented programming principles and design patterns to ensure efficient and scalable code. The game will support multiple teams and players, ensuring unique identifiers and names for each entity. It will be implemented using the Singleton and Iterator design patterns for optimal performance and maintainability.

**Key Features:**

* Support for multiple teams and players
* Unique identifiers and names for games, teams, and players
* Singleton way of providing a single instance of the game
* Iterator pattern for managing game and team names

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

**Constraints:**

1. **Single Instance:** To prevent conflicts and ensure data integrity, only one instance of the game can exist at any given time. This will be achieved using the Singleton design pattern.
2. **Unique Identifiers:** Each game, team, and player must have unique identifiers to avoid duplication and ensure easy management of entities.
3. **Unique Names:** Game and team names must be unique to prevent confusion and ensure a smooth user experience.
4. **Web-Based Environment:** The game must be accessible via web browsers on multiple platforms, requiring compatibility with various operating systems and devices.
5. **Scalability:** The design must support future scalability, allowing additional features and improvements without significantly restructuring the code.

**Implications:**

* Using the Singleton pattern will require careful implementation to manage the lifecycle of the game instance.
* Ensuring unique identifiers and names will involve implementing robust checks and balances within the code.
* Developing a web-based environment will necessitate using technologies and frameworks that support cross-platform compatibility.
* Scalability considerations will influence the choice of data structures, algorithms, and overall architecture to ensure the system can handle increased loads and additional features in the future.

## [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)

**Domain Model Explanation:** The UML diagram provided represents the core structure of the game application, depicting the relationships between different classes and their attributes. The key classes include:

* **Entity:** A base class holding common attributes (id and name).
* **GameService:** Manages games, teams, and players, ensuring unique identifiers and names. Implements Singleton and Iterator patterns.
* **Game:** Inherits from Entity, holds a list of teams.
* **Team:** Inherits from Entity, holds a list of players.
* **Player:** Inherits from Entity.

**Object-Oriented Programming Principles:**

* **Inheritance:** The Game, Team, and Player classes inherit from the Entity class, promoting code reuse and reducing redundancy.
* **Encapsulation:** Each class encapsulates its data and provides methods to interact with it, ensuring data integrity and security.
* **Design Patterns:** The Singleton pattern ensures a single instance of the GameService class, while the Iterator pattern facilitates the management of game and team names.

**"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** | Mac servers offer robust performance and a user-friendly interface, being Unix-based, which ensures strong integration with Apple devices. However, they have higher licensing costs and are less commonly used in enterprise environments compared to Linux and Windows. | Linux is an open-source platform known for its high customizability and is widely used in server environments due to its excellent performance, scalability, and security. However, it requires more technical expertise, and the diverse distributions can cause compatibility issues. | Windows servers are widely used in enterprises and support a broad range of applications. They are easy to use and configure with strong support for Microsoft services. However, they have high licensing costs and are more vulnerable to certain security threats. | Mobile devices are typically not used as servers but can act as clients with high accessibility and portability. They have limited processing power and storage, and battery life constraints. |
| **Client Side** | Developing for Mac requires knowledge of macOS and possibly Swift. Essential tools include Xcode, which is a comprehensive IDE for macOS development. Development costs and time are moderate to high, requiring developers with high expertise in macOS development. | Developing for Linux involves understanding various Linux distributions and using tools like GCC and Eclipse. The development costs are low, but the time and expertise required are moderate to high, demanding developers with substantial experience in Linux environments. | Developing for Windows typically uses Visual Studio with languages like C# or JavaScript. The development costs and time are moderate, requiring developers with moderate expertise in the Microsoft development environment. | Developing for mobile devices involves using Swift and Xcode for iOS, and Java/Kotlin and Android Studio for Android. The development costs and time are moderate to high, requiring developers with high expertise in mobile development. |
| **Development Tools** | The main tools for Mac development include Swift, Objective-C, Xcode, and Homebrew. These tools impact the development team by requiring proficiency in macOS development environments and are moderately costly. | Common tools for Linux development include Python, Java, C/C++, GCC, Eclipse, and VS Code. These tools are typically free or low-cost, but they require developers to have strong expertise in open-source environments. | The primary tools for Windows development include C#, .NET, Visual Studio, and PowerShell. These tools are moderately costly and require developers familiar with the Microsoft ecosystem. | Essential tools for mobile development include Swift (iOS), Java/Kotlin (Android), Android Studio, and Xcode. These tools are moderately costly and require developers with expertise in both iOS and Android development environments. |

## 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**

**Recommendation: I suggest using a Linux server to help "Draw It or Lose It" expand to other platforms.**

**Why Linux:**

* **Flexible and Scalable: Linux can be customized to fit the game's needs, making it a good choice for different platforms.**
* **Good Performance and Security: Linux works well on servers and is very secure, essential for keeping user data safe.**
* **Affordable: Since Linux is open-source, it is free to use, so you do not have to pay for expensive licenses.**

1. **Operating Systems Architectures**

**Description: To keep the game running smoothly on the Linux system, we should use memory pools, lazy loading, and simple data structures.**

**Key Points:**

* **Memory Pools: Helps manage memory better so the game does not lag.**
* **Lazy Loading: Loads resources only when needed, saving memory.**
* **Simple Data Structures: Using accessible structures like arrays helps the game use less memory, making it faster.**

1. **Storage Management**

**Recommendation: We should use a cloud storage system that can compress files and has backup options for storing game data.**

**Why Cloud Storage:**

* **File Compression: Compressing files like images saves space and keeps the game running well.**
* **Backup: Having backups ensures that data is safe even if something goes wrong with the hardware.**

1. **Memory Management**:

**Explanation: The Linux platform uses several methods to manage memory efficiently, so the game runs without issues.**

**Key Techniques:**

* **Quick Rendering: Proper memory use helps images load quickly, avoiding lags.**
* **Garbage Collection: Automatically cleans up unused memory, which helps maintain performance.**
* **Optimized for Each Platform: Tailoring memory use ensures the game works well everywhere.**

1. **Distributed Systems and Networks**

**Explanation: To ensure "Draw It or Lose It" can be played on different devices, we should use a distributed systems approach supported by a strong network.**

**Key Points:**

* **Works on All Platforms: The system must be compatible with various devices to keep the game accessible.**
* **Strong Network and Backup: A reliable network with backup systems ensures the game stays online despite issues.**

1. **Security**

**Explanation: Security is crucial to protect user data in "Draw It or Lose It." The system should have multiple security layers.**

**Security Measures:**

* **Protecting User Data: Using Linux's security features helps prevent unauthorized access to user data.**
* **Encryption and Backup: Encrypting data and regularly backing up keep information safe, even if the system has problems.**