

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.4

## 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 | 07/20/2024 | Darrell Walker | Initial creation of the software design document. |
| 1.1 | 08/03/2024 | Darrell Walker | Updated evaluation section and table format. |
| 1.2 | 08/04/2024 | Darrell Walker | Revised evaluation - added licensing cost and impact on development team. |
| 1.3 | 08/04/2024 | Darrell Walker | Changed format of recommendations section. |
| 1.4 | 08/12/2024 | Darrell Walker | Updated recommendations section (added more details) |

**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 game, Draw It or Lose It, from an Android-only app to a web-based game that can be accessed from multiple platforms.

## Requirements

The key software requirement is to allow multiple teams, each with multiple players, and ensure unique game and team names to avoid conflicts. Additionally, only one instance of the game should exist in memory at any time. This document outlines the software design and implementation plan, addressing these requirements and providing a robust, scalable, and user-friendly solution.

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based distributed environment presents several roadblocks:

1. **Scalability**: The application must handle multiple concurrent users efficiently.
2. **Consistency**: Unique identifiers must ensure no duplicate game or team names.
3. **Performance**: The system must provide fast response times for user interactions.
4. **Security**: User data must be protected across all platforms.
5. **Platform Compatibility**: The application must run smoothly on various operating systems (Windows, Mac, Linux) and mobile devices.

These constraints impact the development process by necessitating careful planning of the system architecture, choice of development tools, and strategies for data management and security.

## [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 represents the structure of the game application. It shows the relationships between various classes and their attributes and methods. Here is a detailed description of each class and how they relate to each other:

1. **Entity Class**
   * **Attributes**: id (long), name (String)
   * **Methods**:
     + Entity(): Constructor
     + Entity(id: long, name: String): Constructor with parameters
     + getId(): long: Returns the ID of the entity
     + getName(): String: Returns the name of the entity
     + toString(): String: Returns a string representation of the entity
   * **Description**: The Entity class is a base class that contains common attributes (id and name) and methods for all entities in the game. It provides a common structure that can be inherited by other classes.
2. **GameService Class**
   * **Attributes**:
     + games (List<Game>)
     + nextGameId (long)
     + nextPlayerId (long)
     + nextTeamId (long)
     + service (GameService)
   * **Methods**:
     + GameService(): Private constructor to implement singleton pattern
     + getInstance(): GameService: Returns the single instance of GameService
     + addGame(name: String): Game: Adds a new game
     + getGame(id: long): Game: Retrieves a game by its ID
     + getGame(name: String): Game: Retrieves a game by its name
     + getGameCount(): int: Returns the number of games
     + getNextPlayerId(): long: Returns the next player ID
     + getNextTeamId(): long: Returns the next team ID
   * **Description**: The GameService class is a singleton that manages all game instances. It ensures only one instance of the service exists and provides methods to add and retrieve games.
3. **Game Class**
   * **Attributes**:
     + teams (List<Team>)
   * **Methods**:
     + Game(id: long, name: String): Constructor with parameters
     + addTeam(name: String): Team: Adds a new team
     + toString(): String: Returns a string representation of the game
   * **Description**: The Game class represents a game with a unique ID and name. It contains a list of teams that participate in the game and provides methods to add teams and get a string representation of the game.
4. **Team Class**
   * **Attributes**:
     + players (List<Player>)
   * **Methods**:
     + Team(id: long, name: String): Constructor with parameters
     + addPlayer(name: String): Player: Adds a new player
     + toString(): String: Returns a string representation of the team
   * **Description**: The Team class represents a team within a game. It has a unique ID and name, and it contains a list of players. It provides methods to add players and get a string representation of the team.
5. **Player Class**
   * **Attributes**:
     + id (long)
     + name (String)
   * **Methods**:
     + Player(id: long, name: String): Constructor with parameters
     + toString(): String: Returns a string representation of the player
   * **Description**: The Player class represents a player within a team. Each player has a unique ID and name, and it provides a method to get a string representation of the player.
6. **ProgramDriver Class**
   * **Methods**:
     + main(): The main method to start the application
   * **Description**: The ProgramDriver class contains the main method that initializes the game application and tests its functionality.
7. **SingletonTester Class**
   * **Methods**:
     + testSingleton(): Tests the singleton behavior of the GameService class
   * **Description**: The SingletonTester class tests the singleton behavior to ensure that only one instance of GameService exists.

**Object-Oriented Programming Principles**

1. **Encapsulation**: Each class has its own attributes and methods, which are hidden from other classes. This helps in managing the complexity by keeping the internal workings of each class private.
2. **Inheritance**: The Entity class is a base class, and Game, Team, and Player classes inherit common attributes (id and name) and methods from it. This promotes code reuse and a hierarchical class structure.
3. **Polymorphism**: The toString() method is overridden in Game, Team, and Player classes to provide specific string representations for each class. This allows objects to be treated as instances of their base class Entity while still using their specific implementations.
4. **Singleton Pattern**: The GameService class uses the singleton pattern to ensure that only one instance of the game service exists. This is crucial for managing the game state consistently across the application.

By using these principles, the software design ensures that the game application is modular, easy to maintain, and scalable. The relationships between classes are clearly defined, and common functionalities are centralized, making the codebase efficient and organized.

**"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)

| **Development Requirements**  **for**  **Server Side** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| Mac servers are known for their reliability and security, but they are expensive and not commonly used for web hosting. They are typically used for specific applications that require high performance and security. | Linux servers are very reliable, cost-effective, and widely used in web hosting. They offer excellent performance, scalability, and security, making them a popular choice for web-based applications. | Windows servers are easy to use and integrate well with other Microsoft products. They offer good performance and user-friendly management tools, but they can be more expensive and less secure than Linux. | Mobile devices are not typically used for hosting web applications. They are primarily used to run the client-side of applications, allowing users to interact with the game. Hosting on mobile devices is impractical due to limited resources and scalability. |
| **Potential Licensing Costs** | Mac OS X Server license costs around $20 per user annually. | Linux is usually free to use, with no licensing costs. | Windows Server licenses can range from $500 to $6,000 depending on the edition and number of users. | No licensing costs for server-side as mobile devices are not used for hosting. |
| **Impact on Development Team (Server Side)** | Developing for Mac server requires expertise in macOS server environments, which may increase costs and require specific training. | Linux server development can be handled by a team proficient in open-source tools, reducing costs and leveraging a wide community support base. | Windows server development requires familiarity with Microsoft server technologies, increasing costs for licenses and possibly requiring additional training. | Server-side development is not applicable for mobile devices. |

| **Development Requirements**  **for**  **Client Side** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| Developing for Mac requires specific expertise in macOS, which can be time-consuming and costly. However, Macs are popular among creative professionals and offer a consistent user experience. | Developing for Linux clients can be cost-effective and requires knowledge of different distributions. Linux offers flexibility and is preferred for environments that need customization. | Developing for Windows is straightforward, with many tools and resources available. It is the most widely used desktop operating system, which makes it essential to support. Development costs are moderate, and there is a large pool of expertise available. | Developing for mobile devices requires knowledge of both iOS and Android platforms. It can be expensive and time-consuming due to the need to create and maintain apps for multiple platforms. However, mobile apps are crucial for reaching a broad audience. |
| **Potential Licensing Costs** | Xcode is free, but macOS and iOS development requires Apple hardware, which can be expensive. Apple Developer Program costs $99/year. | Most Linux development tools are free and open-source. | Visual Studio Community is free for small teams, but larger teams may require Visual Studio Professional or Enterprise, which have licensing fees. | Android Studio is free. Xcode is free, but iOS development requires Apple Developer Program ($99/year). |
| **Impact on Development Team (Client Side)** | Developing for Mac requires a team proficient in macOS and iOS development. Costs include Mac hardware and software licenses. | Linux development can be handled by a team proficient in open-source tools, reducing costs. | Windows development requires a team familiar with Visual Studio and .NET, with costs associated with Windows licenses and Visual Studio subscriptions. | Mobile development requires separate teams or individuals proficient in iOS and Android development, increasing costs and complexity due to maintaining multiple codebases. |

| **Development Requirements**  **for**  **Development Tools** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| Tools like Xcode and Swift are used for Mac development. Xcode is a powerful IDE that supports all phases of development for macOS and iOS applications, but it requires a Mac to run. | Tools like GCC, Make, and various IDEs (e.g., Eclipse, IntelliJ IDEA) are used for Linux development. These tools are often free and open-source, providing a cost-effective development environment. | Tools like Visual Studio and .NET are commonly used for Windows development. Visual Studio is a comprehensive IDE that supports a wide range of programming languages and project types. | Tools like Android Studio for Android and Xcode for iOS are used for mobile development. These tools are platform-specific and provide the necessary features to build, test, and deploy mobile applications. |
| **Impact on Development Team** | Developing for Mac requires a team proficient in macOS and iOS development. Costs include Mac hardware and software licenses. | Linux development can be handled by a team proficient in open-source tools, reducing costs. | Windows development requires a team familiar with Visual Studio and .NET, with costs associated with Windows licenses and Visual Studio subscriptions. | Mobile development requires separate teams or individuals proficient in iOS and Android development, increasing costs and complexity due to maintaining multiple codebases. |
| **Licensing Costs** | Xcode is free, but macOS and iOS development requires Apple hardware, which can be expensive. Apple Developer Program costs $99/year. | Most Linux development tools are free and open-source. | Visual Studio Community is free for small teams, but larger teams may require Visual Studio Professional or Enterprise, which have licensing fees. | Android Studio is free. Xcode is free, but iOS development requires Apple Developer Program ($99/year). |

## Recommendations

**Operating Platform**

**Recommendation:** I suggest using Linux as the main operating platform for "Draw It or Lose It."

**Reasoning:** Linux is widely used because it’s reliable, secure, and doesn’t cost much. It’s flexible, meaning you can adjust it to fit your needs. It also supports many programming languages and tools, which will help your developers build and maintain the game. Linux is good at handling multiple users at once, which is important as the game grows.

**Operating Systems Architectures**

**Description:** Linux uses a type of structure called a monolithic architecture. This means that most of the operating system runs together in a single space, allowing different parts to communicate quickly. This design makes the system efficient and fast. Linux is also modular, so you can add or remove parts depending on what you need, which helps the game run better and be more adaptable.

**Storage Management**

**Recommendation:** I recommend using cloud storage options like AWS S3 or Google Cloud Storage.

**Reasoning:** These cloud storage services are dependable, can grow with your needs, and are easy to connect to a Linux system. They keep your data safe by storing it in several places, so it’s always available even if one server fails. These services also offer strong security features like encryption to protect user data.

**Explanation:** Cloud storage works by breaking data into pieces and spreading it across different servers. This setup protects your data from being lost if something goes wrong with one of the servers. Using cloud storage lets you easily increase the amount of data you can store as more people start playing the game.

**Memory Management**

**Explanation:** Linux manages memory using smart techniques that help the game run smoothly, even when many people are playing at the same time. One important method is virtual memory, which uses part of the hard drive as extra RAM. This lets the game run bigger applications than what the physical RAM alone could handle.

**Technique:** Linux also uses paging, which divides memory into small, equal-sized parts. This method makes sure that memory is used efficiently. If a program starts using too much memory, Linux has a feature called the Out-of-Memory killer that will automatically close the program to free up space. These techniques help keep the game running well, even under heavy use.

**Distributed Systems and Networks**

**Explanation:** To make sure "Draw It or Lose It" works on different devices, the game needs to be set up using a distributed system. This means that different parts of the game will be run on different servers, and they will communicate over the internet.

**Network Considerations:** A key part of this system is using RESTful APIs, which allow different devices to talk to each other in a standard way. The network should also use load balancers to spread out user requests across servers, preventing any single server from getting overwhelmed.

**Dependencies:** When setting up a distributed system, it’s important to think about network delays and keep the data consistent, especially since players need to interact in real-time. Using caching (storing data temporarily) and synchronization techniques can help reduce these issues.

**Security**

**Explanation:** Security is crucial, especially when dealing with user data. To protect this data, I recommend using encryption like SSL/TLS for sending information over the internet. For logging in, using OAuth 2.0 is a good idea because it’s a secure way to verify users without storing sensitive information like passwords directly.

**Additional Security Measures:** To further improve security, make sure to regularly update all software to fix any known vulnerabilities. Using firewalls and intrusion detection systems can help monitor and block unauthorized access to the game’s servers. Also, consider adding two-factor authentication (2FA) for tasks that need extra security, like managing the game’s servers. This adds another layer of protection by requiring a second form of identification.