

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 9/11/2023 | Lee Kitchen | First version |
| 1.1 | 9/25/2023 | Lee Kitchen | Updated Evaluation section |
| 1.2 | 10/09/2023 | Lee Kitchen | Update Recommendations |

**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)

Creative Technology Solutions (CTS) is partnering with The Gaming Room to develop a cross-platform web-based game inspired by their Android app, "Draw It or Lose It," which emulates the 1980s TV game Win, Lose or Draw. The project addresses challenges faced by The Gaming Room in setting up the development environment and aims to fulfill critical software requirements. These include enabling team-based gameplay, implementing unique naming for games and teams, and ensuring a single game instance in memory. CTS will begin by creating a detailed software design document as the foundation for the project, guaranteeing a seamless and unique gaming experience across platforms. Our technical manager will provide ongoing support and oversight to ensure the project's success.

## Requirements

* *A game will have the ability to have one or more teams involved.*
* *Each team will have multiple players assigned to it.*
* *Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.*
* *Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.*

## [Design Constraints](#_2et92p0)

Developing a web-based distributed game application involves addressing several critical design constraints. Cross-platform compatibility is a primary concern, as the game must run smoothly on diverse web browsers and operating systems. This necessitates responsive design, HTML5, CSS3, and extensive testing to ensure a consistent user experience. Network latency poses challenges for real-time gameplay, demanding the implementation of techniques like client-side prediction and server synchronization to maintain smooth interactions.

## [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)

The UML class diagram illustrates the structure of a software system and how various classes relate to each other. In this diagram, several object-oriented programming principles are demonstrated to efficiently fulfill the software requirements.

**Singleton Design Pattern**:

- The ProgramDriver class includes a `main()` method, serving as the entry point for the program. This class may act as a driver for the application.

- The SingletonTester class contains a testSingleton() method, which responsible for testing the Singleton pattern.

**Encapsulation**:

- The Entity class encapsulates data fields: id and name and it provides public methods getId(), getName(), toString() to access and manipulate this data. This follows the encapsulation principle by hiding the internal details of the class while providing controlled access.

**Singleton Pattern Implementation**:

- The GameService class is designed as a Singleton, with a private constructor and a static getInstance() method. This ensures that there is only one instance of GameService throughout the application, fulfilling the Singleton pattern requirement.

**Association and Aggregation**:

- The relationships between classes demonstrate associations and aggregations:

- GameService has an association with Game (0...\*) since it can manage multiple games.

- Game has an aggregation relationship with Team (0...\*) as a game consists of multiple teams.

- Team has an aggregation relationship with Player (0...\*) as a team consists of multiple players.

**Constructor Overloading**:

- The Entity class provides multiple constructors Entity() and Entity(id: long, name: String) demonstrating constructor overloading, allowing objects to be created with or without initial values.

**"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.

**Server Side**

| **Operating Platform** | **Characteristics** | **Advantages** | **Weaknesses** | **Server-Based Deployment** | **Potential Licensing Costs to The Gaming Room** |
| --- | --- | --- | --- | --- | --- |
| **Mac** | macOS, Unix-based, stable, secure, good performance, scalability | Ease of use, developer-friendly tools, compatibility with open-source software | Less cost-effective and powerful for large-scale applications, limited hardware options | Yes, can be hosted on macOS servers | Potentially lower than Windows but higher than Linux |
| **Linux** | Stable, secure, cost-effective, highly customizable, supports various software stacks | Cost-effectiveness, high customizability, broad software support | May require more expertise for setup, limited support for proprietary software | Yes, widely used for hosting web applications | Generally low or open-source |
| **Windows** | Familiar environment, robust features, commonly used in enterprises | Ease of use, compatibility with Microsoft technologies, strong support for .NET and ASP.NET | More expensive, slightly lower performance in some cases, less common for open-source technologies | Yes, can be hosted on Windows servers | Higher than Linux in many cases |

**Client Side**

| **Operating Platform** | **Considerations** | **Software Development Considerations** | **Development Tools** | **Impact on Development Team** | **Licensing Costs** |
| --- | --- | --- | --- | --- | --- |
| **Mac** | Expertise in macOS app development, Swift or Objective-C, Mac hardware costs | Ensure compatibility with macOS browsers, iOS devices | Swift, Objective-C, Xcode | Requires expertise in macOS development | Potentially for hardware and Xcode |
| **Linux** | Linux development skills, cross-platform web applications or Linux-specific desktop apps | Ensure compatibility with various Linux distributions, browsers | Python, Java, C/C++, JavaScript, IDEs like Visual Studio Code | Requires Linux development skills | Generally low or open-source |
| **Windows** | Proficiency in Windows development tools, C# or JavaScript | Ensure compatibility with Windows browsers, desktops | C#, .NET, Visual Studio | Requires expertise in Windows development | Potentially for Windows hardware and Visual Studio |
| **Mobile Devices** | Mobile app development skills, platform-specific SDKs | Ensure compatibility with Android and iOS devices | Platform-specific languages and tools (Java/Kotlin, Swift/Objective-C, Flutter, React Native) | May require separate teams for Android and iOS development | App store fees, potentially for device testing |

**Development Tools**

* **Impact on Development Team:**
  + Different platforms may require specialized skills, potentially necessitating separate development teams for macOS, Linux, Windows, and mobile platforms.
* **Licensing Costs:**
  + Licensing costs may be associated with specific development tools, especially for Windows platforms. Open-source tools on Linux may mitigate this.

## Recommendations

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

**Operating Platform:**

**Recommendation:** Utilize a web-based architecture for Draw It or Lose It to ensure cross-compatibility with macOS, Linux, Windows, and mobile devices.

**Operating Systems Architectures:**

**Chosen Architecture:** Web-based architecture built upon client-server and multi-tier architectures. Clients interact with a central server that manages game sessions and data, hosted on various operating systems like Linux or Windows.

**Storage Management:**

**Recommended Storage System:** Use a relational database system (MySQL, PostgreSQL) or cloud-based solutions (Amazon RDS, Azure SQL Database) for data persistence, scalability, and reliability.

**Memory Management:**

**Memory Handling:** Dynamic memory management handled by the server software, ensuring efficient resource usage. Browser-based clients rely on server-side memory management.

**Distributed Systems and Networks:**

**Communication Mechanisms:** Implement RESTful APIs or WebSocket protocols for communication between various platforms. Consider network dependencies, error handling, and session state management for graceful handling of outages.

**Cross-Platform Compatibility:**

Implement responsive web design.

Ensure compatibility with different browsers (Chrome, Firefox, Safari) and operating systems (macOS, Linux, Windows).

Mobile devices can access via browsers or dedicated apps.

**Real-time Communication:**

Use WebSocket or server-sent events (SSE) for synchronized gameplay and timely updates.

**Load Balancing:**

Implement load balancing for scalability and redundancy.

**Failover Mechanisms:**

Use failover mechanisms and data replication to maintain data integrity in case of server failures.

**Security:**

**Security Measures:**

**Authentication:** Implement secure authentication mechanisms (OAuth, JWT) for user identity verification.

**Data Encryption**: Use HTTPS to encrypt data in transit.

**Authorization**: Apply role-based access control (RBAC) to restrict unauthorized access.

**User Privacy:** Comply with data privacy regulations (e.g., GDPR) and ensure transparent data handling practices.

**Regular Audits:** Conduct routine security audits and penetration testing to identify vulnerabilities.

**Monitoring and Incident Response:** Implement security monitoring tools and establish an incident response plan to detect and respond to security threats promptly.

**Conclusion:**

By adopting a web-based architecture and implementing the recommended measures, The Gaming Room can efficiently expand Draw It or Lose It to multiple computing environments. This approach ensures cross-platform compatibility, real-time communication, scalability, and robust security, providing users with a secure, responsive, and reliable gaming experience.