

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 4-5**

**Evaluation 5-7**

[**Recommendations**](#_Toc115077326) 8

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/20/2025 | Maxwell J. Sciola | First edition |
| 1.1 | 02/03/2025 | Maxwell J. Sciola | Update to Evaluation Section |
| 1.2 | 02/17/2025 | Maxwell J. Sciola | Update to Recommendations Section |

## [Executive Summary](#_sbfa50wo7nsh)

The goal of this software design document is to organize and present the development of the web-based game application *Draw It or Lose It*, to our client, The Gaming Room. The web-based game was inspired by the 1980’s TV show *Win, Lose or Draw*, where teams compete to guess images being drawn. In this version multiple teams within four rounds of gameplay will guess images from a stock drawing library as they render. To enhance the overall gaming experience, we will be making a web-based game accessible on various devices. This proposed design is in alignment with the client’s requirements and aims to produce an entertaining, engaging and user-friendly application.

## Requirements

* *Draw It or Lose It* **must** be on a web-based platform.
* The game **must** work across all devices and operating systems seamlessly.
* During the game, **one or more** teams can **participate**.
* Teams are comprised of **multiple players** per team. (Probably 4 members per team)
* Game and team names **must** be individualized. (Users will be able to verify name availability during team creation)
* Only **one** instance of the game runs at a time. (Proper implementations of unique name identifier for games, teams and players will accomplish this)
* Game rounds **will** be limited to **one minute per round** until the drawing incrementally reveals itself at the 30-second time mark.
* **If** a team is **unable** to solve the puzzle within the given time, then **other teams** are given 15-seconds to make one guess each to solve the puzzle.

## [Design Constraints](#_2et92p0)

* *Draw It or Lose It* must be developed for a web-based platform, presenting constraints in network communication, security and compatibility issues with different web browsers and devices.
* The game is to have a naming system that enforces unique name selection for games, teams and players that is focused on preventing name conflicts and to provide a seamless user experience when joining or creating a game.
* To ensure smooth functionality, the system will limit the game service to one active instance in memory at a time.

## [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 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.**

The diagram above shows all the classes that form the *Draw It or Lose It* game. The Entity class serves as the base superclass for all the other entities within the system. Within the Entity class has attributes like id and name to make sure all other entities in the application have a unique name identifier. The main components of the application are the Game, Team and Player classes, which can be seen by extending from the Entity superclass. A Game is made up of multiple Teams and Teams are made up of multiple Players.

The GameService class has a direct relationship with the Game class, representing its role in managing the lifecycle of Game instances and maintaining references to them. Similarly, the Game class is made up of Team instances, while the Team class is made up of Player instances, demonstrating the hierarchical structure of relationships between them.

The application’s main function is within the ProgramDriver class and acts as the entry point. The ProgramDriver is responsible for creating the singleton instance of the GameService class, making sure only one instance can exist throughout the application. Also, the ProgramDriver class assists the addition of games, teams and players through the GameService instance but also depends heavily on the SingletonTester class, which is indicated by the <<uses>> arrow within the diagram.

Key object-oriented programming principles are highlighted in this UML class diagram such as inheritance, encapsulation, and abstraction. You can see inheritance through the relationship between the Entity superclass and the subclasses of Game, Team and Player, allowing these subclasses to inherit attributes and shared behaviors. This reduces code duplication by allowing subclasses to use the super keyword to call within the Entity constructor.

Encapsulation is manifested in the GameService class by restricting direct access to its attribute, like the constructor and the list of games. It also provides control methods for interaction. By having this design, it promotes privacy in data and confirms only a single instance is maintained. The last object-oriented programming principle that is showcased is abstraction, which can be seen through the simplified management of games, teams and players through the hiding of unnecessary implementation details but also focusing on the essential functionalities.

## [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** | Foundation is Unix-based. Higher hardware costs (approx. $2500 not including $500 for server licenses) and limited scalability compared to Windows and Linux. Mac has a strong ecosystem and is developer friendly but is less common for web hosting (Uses Apache) | OP offers customization, flexibility and access to a wide range of tools due to its open-source nature. There is some hardware compatibility issues and GUI limitations. Servers for Linux can range from as low as $5 per month to $200+ per month depending on the level of performance needed. Linux has no licensing cost and is widely used for web hosting on Apache. | Strong developer ecosystem and wide compatibility with software. High susceptibility to security vulnerabilities. Windows has extensive hardware support and extensive documentation. Normally Windows does not have service fees for hosting websites, but they are known for having more security vulnerabilities. | Smaller screens with touchscreen and gesture-based interactions. Across devices, hardware capabilities vary widely. The cost will be significantly different from device to device. Most likely will require cloud hosting integration that is deployed by AWS or Firebase. |
| **Client Side** | Short learning curve (very user-friendly). Known for its consistent interface. To develop and maintain multiple clients’ costs money, time and specialized expertise. Hardware configurations are limited. For browser-based compatibility, Mac uses HTML5, JavaScript and CSS. | Highly customizable but has a steep learning curve and needs expertise for different clients. The hardware can be expensive.  It is free to distribute and use but performance varies. Need more testing to ensure it is compatible with all browsers. | Windows’ cost of licensing is usually higher than other alternatives (open-sourced). Works on web browsers but again, known to have security vulnerabilities. Performance depends on the choice of browser (Chrome is commonly used for gamming) | Most devices have cameras, touchscreens, GPS and push notifications that can be used. It is important to consider connectivity limitations and responsive design. Frameworks like Native, Flutter and React will help with cross-platform support. |
| **Development Tools** | Common IDEs are VSCode and Xcode and for development, Swift, Node.js and JavaScript. Cross-platform compatibility is limited. There is a $99 per year fee for Apple Developer Program. | Command-line interface is strongly supported.  IDEs are VSCode, Atom and Sublime Text with apt and yum as package management systems. | C# is commonly used, along with .NET Framework. Visual Studio and JetBrains products are popular IDEs. Companies must use the paid version of Visual Studio. | To develop software on mobile devices, developers commonly use programming languages like Java for Android and Swift for iOS. There is a one-time fee of $25 for Google Play while Apple has a $99 per year fee. |

## 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**: To allow the Gaming Room to develop *Draw It or Lose It,* it is recommended that we transition a web-based platform to multiple computing environments. By completing this approach, the game will be accessible on multiple devices by web browsers including desktop, laptops, tablets and even smartphones. This design makes sure that the user experience is consistent across all platforms while also expanding the game’s audience.
2. **Operating Systems Architectures**: There will be two primary architectures for the *Draw It or Lose It* game, one on the client side and the other on the server side. The client side will be rooted in web technologies like HTML, CSS and JavaScript that will manage the game’s user interactions and interface. While the server side will be multi-tiered containing presentation, application and data layers to ensure smooth operations and scalability.
3. **Storage Management**: There will be a hybrid approach to store management. We will use cloud storage to handle stock images while a relational database management system (RDBMS) will be used for user profiles, game statistics and progression. This way we ensure scalability, accessibility and management of game resources will be efficient.
4. **Memory Management**: Developers will be free from manual memory management tasks because this web-based platform will use modern web browsers’ automatic memory management capabilities, like garbage collection. Features like garbage collection handles memory allocation and reduce the risk of memory leaks and optimizes memory usage.
5. **Distributed Systems and Networks**: To ensure a smooth user experience, the game will use distributed architecture and network connectivity. There will be neither a centralized server nor cloud server acting as the communication center, focusing on game synchronization, player interactions and real-time updates. While an error-handling mechanism will manage potential network issues like low bandwidth or connectivity interruptions.
6. **Security**: To keep user data safe across all platforms, dynamic security measures will be in place. The game will use secure communication procedures and apply user authentication as well as authorization methods for usernames and passwords. To protect sensitive information, data encryption will be utilized while in transit across networks and while being storage in a database.