

Win, Lose or Draw

# **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 | 03/19/2023 | Bradley Mills | Updated the executive summary, constraints, requirements, domain, sys architecture, system analysis and recommendations. |
| 1.1 | 3/29/2023 | Bradley Mills | Added evaluations. |
| 1.2 | 4/15/2023 | Bradley Mills | Added Recommendations |
|  |  |  |  |
|  |  |  |  |

## [Executive Summary](#_sbfa50wo7nsh)

Draw It or Lose It is a web-based game application inspired by the 1980s television game Win, Lose or Draw. The game involves teams guessing puzzles based on rendered stock drawings. The game consists of four rounds, with each round having a one-minute timer. The application aims to provide a fun and engaging experience for users while addressing the software requirements specified by the client.

## Requirements

*When designing a game that meets specific requirements, it is crucial to incorporate features such as support for one or more teams within the game, enabling players to join forces or compete against each other. To fulfill this requirement, the game must allow for the assignment of multiple players to each team, fostering collaboration and a sense of camaraderie. In order to simplify communication and recognition, the game should generate unique game and team names, making it easy for players and spectators to identify various elements within the game. Moreover, to ensure efficient resource management and seamless interaction between game components, the system must be designed to maintain only a single instance of the game in memory at any given time. Unique identifiers should be assigned to each game, team, and player instance to facilitate proper tracking and organization. By meeting these requirements, the game will create an engaging and enjoyable experience for all participants.*

## [Design Constraints](#_2et92p0)

In the development of a web-based application, it is crucial to consider specific design constraints to ensure an optimal user experience. The application must be accessible via modern web browsers, enabling users to interact with the game on various platforms without the need for additional software. A responsive design is essential to accommodate different devices and screen sizes, ensuring a seamless and enjoyable experience for players, whether they are using a smartphone, tablet, or desktop computer. Furthermore, the application should feature a user-friendly interface, prioritizing easy navigation and intuitive game interactions. By adhering to these design constraints, developers can create an engaging and accessible gaming experience that appeals to a wide range of users across multiple devices and platforms.

## [System Architecture View](#_ilbxbyevv6b6)

The proposed system architecture for this gaming application is designed to provide a seamless and engaging experience for users. On the front-end, the user interface and interactions are built using HTML, CSS, and JavaScript, ensuring a responsive and visually appealing design. The back-end leverages a Node.js server to handle essential components such as game logic, user authentication, and data storage, providing a robust and efficient foundation for the application. To store game and user data, the architecture employs a database system, which could be MongoDB or PostgreSQL, offering flexibility and scalability to accommodate the needs of the application. Finally, real-time communication between clients and the server is facilitated through the use of WebSockets or Socket.IO, enabling instantaneous updates and interactions for an immersive gaming experience. By incorporating these elements into the system architecture, the application delivers a reliable and enjoyable platform for users to enjoy.

## [Domain Model](#_8h2ehzxfam4o)

The Game, Team, and Player classes all have an "is-a" relationship with the Entity class, which means that they inherit from Entity. This inheritance can be represented in UML with an arrow pointing from the subclass to the superclass. Since Game, Team, and Player share common attributes, such as id and name, these attributes are defined in the superclass Entity.

The relationship between Team and Player is a "has-a" relationship, which means that a Team has Players. Similarly, a Game has Teams, and a GamesService organizes (has) Games. In UML, this can be represented through Aggregation. A has-a relationship indicates that an instance of one class holds a reference to an instance of another class. In the given diagram, GamesService holds references to Games, Games hold references to Teams, and Teams hold references to Players. A single GamesService can have multiple Games, each Game can have multiple Teams, and each Team consists of multiple Players.

**"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)

Server-side Evaluation

All three traditional operating platforms (Linux, Windows, and MacOS) offer server-based deployment methods for hosting websites. Linux has the lowest licensing costs, while Windows has the highest licensing costs, and MacOS is in between. To scale up to thousands of players, all three platforms can be used for hosting the software application. However, Linux is typically preferred for its cost-effectiveness and open-source nature.

Client-side Evaluation

To support players on iOS and Android mobile platforms, as well as traditional desktop-based operating systems, the application must be developed using responsive web design principles. The application should be compatible with all web browser platforms and mobile devices, requiring extensive testing and optimization efforts. Developing for mobile platforms (Android and iOS) may require additional expertise and involve higher costs and longer development time compared to desktop platforms.

Development Tools

For server-side development, languages such as Python, JavaScript (Node.js), and PHP can be used across all platforms. Front-end development requires HTML, CSS, and JavaScript.

Popular IDEs for web development include Visual Studio Code, Sublime Text, and JetBrains IDEs, which are available on all major platforms.

The use of these languages and tools may not require multiple development teams, but mobile platforms might benefit from a dedicated team due to the unique challenges and expertise required.

Licensing costs for development tools vary, with some tools being free and open-source, while others may require a subscription or a one-time fee.

Conclusion

Based on the evaluation, Linux is the most cost-effective choice for hosting the web-based application. For client-side development, a modern, responsive HTML interface is necessary to ensure compatibility across various platforms. While the same development languages and tools can be used across all platforms, mobile development may require a dedicated team due to the unique challenges and expertise required.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Expensive. Well done GUI allows ease of use for devs and users. Easy server configurations | Economical in nature, Linux can be challenging to navigate for some users. Similar to macOS, it features a command shell, which facilitates straightforward server configuration and accessibility. | Similar to macOS, Windows can be expensive. However, it offers a user-friendly platform with an abundance of available software options. Additionally, it provides access to the command prompt. | Affordable in nature, mobile device specifications differ among users, including Android, iOS, and Windows Phone. There are challenges in developing a game that is compatible with most or all mobile platforms. |
| **Client Side** | Time: Average for users. Some skills are needed. The costs are higher due to licensing. | The time required to assist Linux OS users can be considerable. Proficiency in Linux is essential, yet the operating system comes with minimal associated costs. | Supporting Windows OS users requires relatively little time and expertise. The cost associated with Windows is comparatively similar to that of macOS. | A significant amount of time and expertise is needed to support various mobile device users and clients, as mobile operating system platforms can be challenging to operate on different devices. |
| **Development Tools** | Similar to numerous operating system configurations, the most frequently used programming languages and IDEs on macOS encompass JavaScript, CSS, Python, HTML, among others. The tools consist of libraries that support various languages, as well as IDEs like Eclipse, Visual Studio, and more. | Commonly used languages encompass Ruby on Rails, Java, Python, CSS, JavaScript, HTML, and more. Development tools consist of Node.js, Visual Studio, GitHub, Repl.it, and command prompt. | Typical languages employed include Ruby on Rails, Java, Python, C++, C#, JavaScript, HTML, and more. Development tools comprise Visual Studio, Eclipse, Repl.it, and command prompt. | Commonly employed languages include Python, Java, CSS, JavaScript, HTML, PHP, Ruby on Rails, C++, and others. Development tools encompass Repl.it, Node.js, GitHub, Visual Studio, and command prompt (PowerShell). |

## Recommendations

1. **Operating Platform**: A web-based application for its accessibility, cost-effectiveness, and cross-platform compatibility. Web-based apps can be accessed from any device with a web browser, broadening the potential audience. They are generally cheaper to develop and maintain compared to native apps, as they require fewer resources and bypass app store fees. Additionally, web-based applications offer a consistent user experience across platforms, as they run on various operating systems and devices without separate development processes.
2. **Operating Systems Architectures**: The chosen operating platform architecture for the Draw It or Lose It software is a web-based application with a Node.js back-end server, designed to be cross-platform compatible (Windows, macOS, Linux). The architecture consists of a front-end built using HTML, CSS, and JavaScript for responsive and interactive UI/UX, and a back-end server powered by Node.js for managing game logic, user authentication, and data storage. A database, such as MongoDB or PostgreSQL, stores game and user data, while real-time communication is handled through WebSockets or Socket.IO. This architecture aims to provide a seamless and enjoyable gaming experience across various devices and operating systems.
3. **Storage Management**: Cloud-based storage for scalability and reliability (Amazon Web Services, Google Cloud, or Microsoft Azure). Cloud storage allows for easy data access from any device with internet connectivity, catering to a wider user base. It is cost-effective, as it eliminates the need for expensive on-premises hardware and maintenance, while offering flexible pricing based on usage. Additionally, cloud storage supports various platforms and operating systems, ensuring a consistent user experience across devices.
4. **Memory Management**: Memory allocation in Node.js is managed through the V8 JavaScript engine, which is responsible for dynamically allocating and deallocating memory as needed. When a new object is created or when data is stored, memory is allocated automatically. V8 efficiently handles memory allocation by segmenting memory into different spaces, such as the new space (for short-lived objects) and the old space (for long-lived objects). This allows for faster allocation and deallocation of memory for frequently used objects.
5. **Distributed Systems and Networks**: Achieving cross-platform communication for Draw It or Lose It can be accomplished through a distributed software architecture using a client-server model. The front-end clients (web browsers on various platforms) connect to a back-end server that handles game logic, user authentication, and data storage. Real-time communication can be facilitated using WebSockets or Socket.IO, ensuring seamless interaction among multiple devices and operating systems. However, it is essential to consider dependencies between components, such as network connectivity, outages, and latency, to provide a reliable and enjoyable gaming experience for users.
6. **Security**: To ensure security for Draw It or Lose It across various platforms, several measures can be implemented. These include using HTTPS for secure data transmission, employing user authentication and authorization mechanisms, protecting stored data with encryption, and utilizing secure coding practices to prevent common vulnerabilities such as SQL injection and cross-site scripting. By incorporating these security measures, the recommended operating platform can safeguard user information and provide a secure gaming experience.