

Draw It or Lose It

# **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 |
| --- | --- | --- | --- |
| 3.0 | <10/27/24> | THOMAS SEIBERT | This version is a review of Version 1.0 and Version 2.0’s recommendation to The Gaming Room on which platform to use. I have elected to stick with my initial choice of Linux (server) in my recommendation. |
| 2.0 | <10/27/24> | THOMAS SEIBERT | After reviewing Version 1.0's evaluation of various operating platforms, which I already highlighted, I decided to make no changes to the document. All the information regarding the server side, client side, and development tools is still valid. |
| 1.0 | <10/24/24> | THOMAS SEIBERT | Updated summary, client requirements, design constraints, architecture view, domain model, evaluation and final recommendation |

**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)

In this executive summary, I will introduce the software design problem and present an appropriate solution. As a technology consultant for Creative Technology Solutions (CTS), my job here is to help our new client, The Gaming Room, develop a web-based game which serves multiple platforms. This new web-based game is based on their current game, titled "Draw It or Lose It", and this game is currently only available on Android.

This document will outline the transition from an Android-only application to a multi-platform web application. In this way, the game will be more accessible and interactive for a broader clientele of players. The new web-based game will allow for simultaneous gameplay, and thus permit the ability to have one or more teams involved. More than one player will be assigned to each team. Additionally, game and team names will be unique to allow users to check when a name is in use when choosing a team name.

## Requirements

1. Web-based and multi-platform: The new game will be universally accessible through the internet, which will ensure an optimal compatibility with all kinds of devices, web browsers, and operating systems.

2. Multiple teams and multiple players: Every game will allow for the engagement of multiple teams, and each team will be able to have multiple players.

3. Consistency: There will be consistency in the transition between the Android application and the web-based application, as this new game will be based on the Android only version.

4. Unique identifiers: Games and teams will be unique in their identifiers (or names) to prevent conflict and check for availability when creating a new team.

5. In-game purchases: The web-based, multi-platform version will allow for in-game purchases which will optimize and maximize monetization for the client.

6. Time limits: This new game will have time limits, including four one-minute rounds, where drawings will be rendered at a steady rate and are fully complete at the 30-second mark. If the team does not guess the puzzle before time expires, the remaining teams have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit.

7. Flexibility and scalability: The web-based multi-platform game will have an infrastructure which will always support an ever-increasing number of users.

## [Design Constraints](#_2et92p0)

1. Multi-platform compatibility: The game must be universally functional across all devices and web browsers. This means optimal and up-to-date software, security patches, and communication protocols on the game server and OS server level.

2. Multiple players on multiple teams: There must be centralized and interactive support for all teams and all players regardless of the number of teams or players on each team.

3. Optimal performance: Latency must be reduced and minimized as much as possible where it is possible.

4. Unique identifiers: Each team, player, and game name must be unique names (identifiers) to provide the best gaming experience for the user.

5. A single instance: The game can only have a single instance active in memory at any given 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 above UML diagram is a blueprint of the structure, organization and relationship between each class for the initial software application, "The Gaming Room". Now I will highlight how each class functions, what each class contains, and how each class relates to one another, along with their embodied Object-Oriented Principles (OOP).

Entity: is the base class for the following subclasses -- Game, Team, and Player. Entity provides private attributes such as id and name, and these attributes are then inherited by those sub-classes. The OOP that Entity embodies are abstraction and reusability, which avoids duplication by the subclasses of these attributes. Abstraction also exists for these attributes, via subclasses expanding on said attributes. In summary, this base class permits common functionality across all classes and entities.

Game: This subclass inherits from Entity and contains a list of Team instances. Every game should be unique and identifiable by id and name, and each game should be able to facilitate more than one team. The OOP embodied by Game are that of inheritance (from Entity), and an aggregation of the list of Team objects.

Team: This subclass also inherits from Entity, which contains a list of Player instances. Each team consists of players. This subclass contains a method to add team names and players. The OOP embodied by Team are inheritance (from Entity), which allows for reusability, and composition which manages Player objects.

Player: This subclass also inherits from Entity, and it contains a player id and a player name. The OOP embodied by Player are abstraction and, as mentioned, inheritance. Abstraction via the fact that each Player can represent a specific entity with unique attributes.

GameService: There can only be one instance of this class at any given during each game. This is called a singleton class. It contains a list of Game instances, attributes to get player id, game id, and team id. It also contains methods to add a game and count a game. The OOP embodied by GameService is encapsulation, since this class controls access to the game instance, and it provides methods to interact with the game list. The game list and management of id's are private and thus hidden from all other classes. In summary, this class follows a singleton pattern, which also means that games are managed centrally by this class, and it is ultimately responsible for the facilitation of the game.

In summary, this UML diagram depicts how the game is organized on the software level. Diagrams like this help to ensure consistency and functionality for the end user.

## [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 has robust and secure architecture. It is user-friendly and it is known for stability. Mac supports a wide range of software and hardware. Mac is known for its strong security features, which is crucial for a server hosting a web-based application. Mac is highly stable, reducing the chances of server crashes and downtime. In addition, Apple provides real-time support for their products, which can be beneficial in case of any issues. In terms of cons, Mac software is generally more expensive than other platforms. Overall, Mac might not be the best choice for large-scale server deployments due to financial cost and hardware limitations. | Linux is an open-source operating system. Linux is known for its robust security features. Linux servers are known for their high uptime and stability. Being open-source, Linux is free to use, which may reduce overall costs. Linux is flexible and can be customized to meet specific needs. There is also a large community of developers who can provide support in real-time. In terms of cons, Linux may appear to be complex and foreboding to users who are not already familiar with its command-line interface. Additionally, not all software is compatible with Linux, limiting its operations. | Windows servers have a GUI that is easy to use, and user friendly. Windows servers are compatible with .NET, which can be beneficial for various applications. Microsoft provides real-time support for its server products. Windows servers integrate properly with other Microsoft products, which can simplify the setup and management of a server. Windows servers have a high scalability, making them suitable for a wide variety of applications, small and large. Furthermore, in terms of security, Windows servers come with built-in features and regular updates. In terms of the cons, Windows servers are usually more expensive than other options. Also, Windows servers can require more resources (in terms of CPU and memory) than other server types, which will impact performance. | Phones and other mobile devices are obviously very portable and can be used anywhere with an internet or network connection. Compared to desktops, mobile devices have a small amount of processing power, memory, and storage. On the other hand, mobile devices allow users to access the game from anywhere. Additionally, with the proliferation of smart cell phones, the user base is vast. In terms of cons, their limited processing power and memory can affect the performance of the game. Also, the game's performance can be affected by the quality of the user's internet connection. |
| **Client Side** | Mac development tools and hardware are generally expensive, and additional costs may be incurred for testing on multiple Mac hardware configurations. Developing for multiple platforms can increase the time spent in the development phase. In terms of cons, developers need to be familiar with Mac-specific languages, tools and features. | As many development tools for Linux are open source, the software costs can be lower. If the development team is not familiar with Linux, there may be a learning curve that could extend the development timeline. Developers need to have a good understanding of the Linux operating system and its command-line interface. | Developing multiple client types can increase costs due to the need for additional testing, different development tools, and different languages. Supporting multiple client types may also increase the development time as each client type may require different code, testing, and debugging. Lastly, developing multiple client types usually requires a team with many skills and comprehensive knowledge of different programming languages and platforms themselves. | Like other OS's, developing for multiple types of systems can increase costs due to the need for additional testing and development resources. Supporting multiple types of clients can increase the development time as each client may require different code or adaptations. Developing for multiple types of clients requires extensive knowledge in different programming languages and platforms. |
| **Development Tools** | Mac has two platform specific programming languages: Swift and Objective-C. They are both Apple's programming languages for iOS and macOS app development. As for an IDE, Xcode is Apple's IDE for Mac and iOS development. Another option is Visual Studio. | In terms of languages compatible with Linux, Python is supported. JavaScript is essential for web-based applications. C++ is also supported. In terms of IDE, Eclipse is compatible. NetBeans is also open source and is well-suited for web development. | Commonly used and relevant programming languages: C#, very useful for a wide variety of applications including .NET. JavaScript, which is essential in web-based applications. HTML/CSS, commonly used in web development. In terms of IDE's: Visual Studio, which is a well-rounded and comprehensive environment developed by Microsoft. .NET Framework, which is a software framework made by Microsoft for Windows. There is also SQL Server, which is a database management system. | Relevant programming languages include Java for Android, Swift and Objective-C for iOS, and JavaScript when applicable for cross platform usage. Relevant IDE's: Android Studio for Android, Xcode for iOS, and Visual Studio code when applicable for cross platform usage. Unreal and Unity are both platforms for developing games which support multiple platforms. Cordova allows for the development of mobile apps using HTML/CSS and JavaScript. Finally, React Native is a framework that allows for cross platform mobile application development. |

## 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**: In this case, I recommend Linux server, after reviewing the client's requirements and constraints. Linux is open-source, cost-effective, and it supports a wide range of hardware and software. It is very customizable, which makes it ideal for game development.
2. **Operating Systems Architectures**: Linux uses a monolithic kernel, which means the OS works in kernel space. This includes device drivers, system management, and system server processes. The Linux architecture consists of the following components:

a) Kernel. The kernel is the core part of the OS. It interacts directly with the hardware and controls the hardware resources.

b) Shell. The shell is the interface through which users interact with the kernel.

c) Utilities. Utilities are software functions that allow users to manage their computers.

d) Applications. Apps are programs that run on the OS.

1. **Storage Management**: In terms of an appropriate storage management system to be used with Linux, I recommend a Relational Database Management System, such as MySQL for example. RDBMS is compatible with Linux and provides an efficient and secure storage solution.
2. **Memory Management**: There are a few memory management techniques, which can be utilized by Linux for the 'Draw It or Lose It' software:

a) Virtual Memory, which allows the system to use disk space as additional memory.

b) Swapping, which involves moving inactive processes from RAM to disk storage to free up memory.

c) Page Caching, which involves storing frequently accessed data in memory to improve performance.

1. **Distributed Systems and Networks**: To enable communication between different platforms, 'Draw It or Lose It' can be developed as a distributed system: divide the game into smaller, independent components that communicate over a network. Host the game server on a Linux server where players may connect to it from whatever device they are using. Use TCP/IP for delivery of data.
2. **Security**: In terms of security and user protection, a few measures can be implemented:

a) Encryption. Data sent over the network must be encrypted using protocols such as SSL/TLS.

b) Authentication. This can be done with a password and e-mail, but users should be required to authenticate themselves before accessing the game.

c) Access Control. Users should have access only to their own data and sessions.

d) Firewalls. This feature can be used to block unauthorized access to the game server.

e) SELinux. A built-in Linux feature which provides mandatory access control.