

**Draw It or Lose It**

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc206288794)

[Table of Contents 2](#_Toc206288795)

[Document Revision History 2](#_Toc206288796)

[Executive Summary 3](#_Toc206288797)

[Requirements 3](#_Toc206288798)

[Design Constraints 3](#_Toc206288799)

[System Architecture View 3](#_Toc206288800)

[Domain Model 4](#_Toc206288801)

[Evaluation 4](#_Toc206288802)

[Recommendations 7](#_Toc206288803)

[References 9](#_Toc206288804)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/19/2025 | Michael Peck | Initial Assessment of Project Requirements |
| 1.0.1 | 8/3/2025 | Michael Peck | Adding Evaluation |

**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 wants to release a web-based game that is compatible with multiple platforms. The game, Draw It or Lose It, is currently on the Android market only.

## Requirements

The system must mirror the game currently on the Android market (i.e., mechanics, visuals, etc.)

The system must be compatible with multiple platforms (Android, iOS, etc).

The system must have the ability to have one or more teams involved. (Southern New Hampshire University, 2025)

The system must allow teams to have multiple users assigned. (Southern New Hampshire University, 2025)

The system must allow game instance names and team names to be unique, so users can verify a name currently in use. (Southern New Hampshire University, 2025)

The system must allow only one instance of the game to exist in memory at any given time. (Southern New Hampshire University, 2025)

## [Design Constraints](#_2et92p0)

The game files and resources used must be compatible with multiple platforms for uniformity.

The code between platforms (Android, iOS) must behave as if they are mirrored, but with opposing languages.

A class governing team formation must be implemented within the game instance class, with functions to assign and remove players or to balance uneven teams (matchmaking)

Input validation must be implemented when naming game sessions or team names to prevent duplications (session ids will also be unique, but hidden)

Following the singleton model, only one instance of the game control class can be 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 ProgramDriver class is the main entry point for the program. Its main purpose is to create a single instance of the GameService class and call back that reference as needed. The Entity class will be the parent class for the Game, Team and Player classes, and will share many attributes (or inherit) among them to protect and encapsulate certain data fields from being altered by classes not authorized to. Finally, the GameService class will be the driving force to create games, teams and sessions. While only one instance of this will be running, it will be the one controlling much of the games.

**"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.

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** | * Known for stability and reliability but optimized for applications developed in iOS. (Whatley, 2024) * Apple no longer makes OS Server. Mac PCs can act as server devices. | * Most flexible server options and full customization (Ha, 2024). * Linux is the most cost-effective platform of all available for web development. | * Highly compatible with a wide range of software. (Whatley, 2024) * Integration with existing tools offers ease of management (Whatley, 2024). | * While possible, multiple barriers prevent extended use. * Ideal for basic websites, personal pages, business cards (3HCloud, 2024). * Network connectivity relies on cell service data and Wi-Fi, can be unreliable with enhanced traffic load (berrybakedoatmeal, 2023). |
| **Client Side** | * Due to End-of-Life (E-o-L) of OS Server, multiple Mac devices will need to be bridged together, so time and costs will be high, and support will be low, or minimal. * Proprietary software could cause price gouging, as well as deployment costs and licenses per application. | * Linux is cost-effective, so scalability is feasible in any-size company. * Versatility allows multiple language support, and deployment options. Supports UNIX and Windows systems, either as custom installs or libraries. | * Cheaper to run than MacOS, only out-performed by Linux/Open-Sourced OS’s. * Versatility similar to Linux-based systems, with multiple languages/IDEs supported. * Cheaper to develop, test applications and more user-control than UNIX systems. | * Due to limited resources of mobile devices, cost will be astronomical for even a small-scale distributed web-based application, as many devices will be required to match specs. * Time to configure devices does not outweigh any cost margin (likely operate at a loss) * ARM architecture education required for modifying a mobile device to be used as server. |
| **Development Tools** | * Xcode is required for iOS development in Objective-C/Swift. * Docker may be needed for deployments * May require additional middleware to support EoL devices since OS Server is no longer supported. * Some form of version control (Git, SVN, PerForce) recommended. * Software compatibility is limited but offers support for popular web servers. (Whatley, 2024) | * Linux supports many IDEs and can be deployed in almost any environment (Ha, 2024). * Terraform and Ansible can be utilized to manage multiple VMs at once (Ha, 2024). | * Windows is open to a wide range of supported languages and IDEs (Visual Studio/Code Eclipse, etc) * Docker may be needed for deployments * Some form of version control (Git, SVN, PerForce) recommended. | * Tools are limited to Android and iOS and supported languages (Kotlin, Java for Android; Xcode and Objective-C/Swift for iOS). (Android Developers, 2025) |

## 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**: Based on the information provided above, the most sensible platform to distribute and package the application described would be a Linux style distro. Ubuntu, Red Hat Enterprise Linux RHEL, etc would be an appropriate way to distribute across multiple device platforms.
2. **Operating Systems Architectures**: An open-source operating system gives the user full control over all aspects of the server infrastructure and allows the capability and compatibility with virtually all other device platforms. The operating system also uses fewer resources than the standard Windows or Apple servers, as they are not loaded with bloatware. The cost associated with Linux style distributions are much lower than those others compared in this document.
3. **Storage Management**: Traditional file storage systems will be sufficient as much of the data that will be stored are photos or drawings used in game events. Other information such as score keeping and player profiles, can also be written as simple text files or XML files, or relatively small overall files. Despite such small files, having a system of redundant arrays or RAID will prevent loss of data. RAID levels of 5, upwards to 10 will consume more physical resources, but can save the company money in the event of unrecoverable information. Whether host-based or cloud-based, levels of storage required for such an application will not be as costly as other components.
4. **Memory Management**: To minimize overuse of the server’s physical and virtual memory, the application will need to only load up a max of two (2) drawings in a queue for each game session, rather than load every image into memory. Loading one (1) or two (2) images at a time for each game session allows no latency issues between game rounds, while also not overwhelming the memory controller. In the event of memory latency, having the next round’s image queued up, allows the system to reallocate any memory address that is currently not being utilized, attempting to restore normal memory access speed. Further stress to the system, despite the attempts to address latency issues, will need to be addressed by adding physical and/or virtual memory to the server’s resources, or in other words, scale up as necessary.
5. **Distributed Systems and Networks**: Linux distributions are known for being the most flexible of all currently analyzed systems. Linux is compatible with both Windows and UNIX systems and are the initial foundation for Android machines. As such, they understand languages from all device platforms and are compatible with virtually all file types. Simple coding knowledge is required in allowing communication and translation between all device platform types. To avoid outages or interruptions to game services, using a cloud-based system that spans regional availability zones will help prevent these interruptions. Fully funding multiple data centers to host this application in availability zones will be overly costly and may end up costing more than intended.
6. **Security**: Depending on the monetization model for the application, one of two authentication methods could be used. If no banking information or customer’s personal identification is provided, a basic authentication system could be employed to the system to prevent bots and some low-level hackers from getting access to non-confidential information. However, if there is banking or personal identification information is held within a database or other file system, then a more advanced authentication method will need to be implemented. Private secure tunnels may be deployed, secure shells, or even multi-factor authentication. All of these methods can be deployed by the server and information passed from the client, even from a variety of device platforms using standard language.

## References

Southern New Hampshire University. (2025). *Project One Guidelines and Rubric - CS-230-15702-M01 Operating Platforms 2025 C-4 (Jun - Aug)*. Snhu.edu. <https://learn.snhu.edu/d2l/le/content/1969244/viewContent/42279302/View>

Whatley, S. (2024, March 23). *Comparing Mac and Windows Servers for Web Hosting*. MamboServer. <https://www.mamboserver.com/blog/comparing-mac-and-windows-servers-for-web-hosting/>

Ha, D. (2024, July 24). 9 reasons Linux is a popular choice for servers. LogicMonitor. <https://www.logicmonitor.com/blog/9-reasons-linux-is-a-popular-choice-for-servers>

3HCloud. (2024, February 19). Powerful cloud provider 3HCloud. 3HCloud. <https://3hcloud.com/blog/technologies/deploy-a-server-on-your-phone/>

berrybakedoatmeal. (2023, July 6). Real-time communication for mobile apps - berrybakedoatmeal - Medium. Medium. <https://medium.com/@jane.anqi.cui/real-time-communication-for-mobile-apps-8b5b37412dc7>

Android Developers. (2025). Kotlin and Android. Android Developers. <https://developer.android.com/kotlin>

‌

‌

‌

‌

‌

‌