

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 |
| --- | --- | --- | --- |
| 1.0 | <07/20/24> | Brandon Cook | Added executive summary, design constraints, and domain model descriptions. |
| 2.0 | <08/03/24> | Brandon Cook | Completed evaluation section. |
| 3.0 | <08/16/24> | Brandon Cook | Completed recommendations section. |

**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)

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know in order to proceed with the process you are proposing.>

The Gaming Room wants to develop a web-based game that is based on their popular Android app, Draw It or Lose It. The game resembles the 1980 television game *Win, Lose or Draw*, which has multiple teams compete to guess what is being drawn. A database will store stock drawings that are rendered on screen as clues. By developing this as a web-based game, it will become more accessible across multiple platforms. The game programming will need to reside on a central server that is accessible via web browsers across different platforms.

## Requirements

* The game needs to be accessible through a web browser on multiple platforms.
* The game needs to allow one or more teams to be involved.
* Each team needs to support multiple players.
* Both game and team names must be unique.
* Users need the ability to check whether a name is already in use when choosing team names.
* Only one game instance can exist in memory at a time.

## [Design Constraints](#_2et92p0)

* The game needs to be designed for use in multiple web browsers across multiple platforms and devices. The application will need to be tested across multiple different architectures for stability and functionality.
* The performance of the game needs to be consistent across browsers and platforms.
* Architecture needs to be scalable to support multiple concurrent teams and players participating in the game. Cloud services are ideal for automated scaling without the need for overhead.
* A different user interface will need to be developed for operating system browser and mobile device browsers to allow for ease of use, while remaining consistent with the overall structure of the application.
* The game needs to support multiple teams, and each instance of the game must have at least one team.
* There may only be one instance of the game active in memory at any given time.
* Checks must be in place to prevent two teams from sharing the same name.
* Each team must support multiple players.

## [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)

ProgramDriver contains the main method and runs the program. The ProgramDriver class uses the SingletonTester class to ensure that the program behaves correctly as a Singleton. The Entity class is a parent to the Game, Team, and Player subclasses, which inherit the attributes from Entity. Each subclass has a relationship with each other. Game has a one-to-many relationship with Team; one game can have many teams. Similarly, Team has a one-to-many relationship with Player; one team can have many players. There can only be one GameService for any number of Game instances. The individual instances of each class must have a unique ID.

**"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** | MacOS is known for its secure, stable environment and its interface. However, it is very expensive, and integration only exists primarily between other Apple devices. Finding suitable hosts for a web-based application in a multi-platform environment will be difficult. | Notably the most popular operating system for webhosting due to being open source with extensive community support. Licensing costs are much cheaper than other operating systems. Cloud providers typically support Linux. | Licensing costs can be expensive for server architecture, and hosting platforms aren’t as abundant as Linux. However, Windows Servers rely on a graphical user interface and Windows is primarily utilized in organizations, so there will be more familiarity. | Mobile devices typically do not have the resources required to host a multi-user service. Hardware is limited, and hosting platforms would likely have to be developed specifically. |
| **Client Side** | The Xcode IDE, which is the primary IDE for development on MacOS, is free. Libraries and frameworks may cost money, and Apple products are diverse, which requires testing on multiple different types of devices. Developers for MacOS typically have higher salaries. | Most development tools for Linux are free and open source, like Linux itself, with extensive community support. Linux supports a very wide range of hardware, so testing may be difficult. Development in Linux typically focuses on more common programming languages, so skills won’t need to be as specialized as in MacOS, which could lead to reduced costs. | Most applications are developed using the .NET framework, which is a commonly used framework for many applications. There would be no need to hire specialized skillsets. Windows is one of the most common platforms and the preferred operating systems for the majority of users. | APIs for Android OS are mostly written in Java, which can tie in well with Windows and Linux. Additionally, iOS ties in well with MacOS development. Mobile devices will allow for better portability as the application can be utilized anywhere there is a data connection. However, mobile devices have varying hardware capabilities and resources, along with various differing screen sizes, which need to be considered when testing. |
| **Development Tools** | The open-source language Swift is one of the most popular programming languages on Mac. Additionally, there are the more common languages such as Python and Java.  The IDE’s used in MacOS are Xcode, which supports Swift and Objective-C, along with PyCharm, Eclipse, CLion, Visual Studio, and other more common IDE’s. | Linux is very flexible with support for Python, C++, Java, and JavaScript. There are also more novel and modern programming languages such as Rust and Go. Bash is used for command-line scripting for automating tasks in Linux.  Linux supports many of the popular IDEs like Eclipse, Visual Studio, and NetBeans. Another popular IDE on Linux is IntelliJ IDEA for Java and Android development. | Development is primarily done using C# and .NET. Visual Studio is one of Microsoft’s most popular IDE’s and offers support for many plugins. Additionally, Windows supports many other popular IDE’s, making it a popular choice for developers due to its flexibility. | Android SDK is Java based. Android Studio is designed specifically for mobile development, supported by Google. It uses the IntelliJ IDEA core.  iOS, like MacOS, utilizes Xcode for Swift and Objective-C 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**: Linux, the most popular operating platform for server-based hosting, should be used for the application. Its versatility as an open-source platform with extensive community support makes it the best solution for a multi-platform product. It is supported by the most popular cloud providers and has support for a diverse set of programming languages. It will allow The Gaming Room to cut costs without sacrificing any stability or performance.
2. **Operating Systems Architectures**: Linux consists of the Kernel, System Library, Hardware Layer, System, and Shell utility. The kernel abstracts the details between low-level programs and hardware to the system. In other words, it is the tool that communicates between the operating system and the hardware connected to it. The library consists of methods that implement the operating platform’s functionality. The hardware layer contains all of the peripherals, such as the CPU, storage media, and memory. The shell utility allows users to input commands to interface with the kernel; this utility can be provided as a graphical interface or as simply a command-line. The setup of Linux allows it to remain portable and allows for multiple programs to be executed simultaneously.
3. **Storage Management**: There is not much storage needed for the application. There are images that should be pre-rendered on the server, along with the code to develop and run the program. For this reason, cloud storage should be the solution. It will allow The Gaming Room to only purchase what storage they need, they will not need to maintain a physical database that will incur additional costs, and if they ever need to expand the storage to add more images, cloud providers are scalable. Storing all of the required files on a cloud storage solution is more cost-effective and allows The Gaming Room to provide more resources towards developing the game.
4. **Memory Management**: Linux uses two concepts: virtual memory and paging. Virtual memory is a method of memory management that allocates an amount of virtual memory to a program or application. This allows for multiple programs to run simultaneously. Additionally, Linux breaks memory into chunks, known as pages, which are only allocated to applications when more memory is needed. If RAM is full when doing this, it will utilize storage space instead.
5. **Distributed Systems and Networks**: Cloud providers have mitigation strategies in place to minimize any potential downtime to the point that it is negligible. They can easily move data and services across geographic locations to other servers, so that if one of their servers goes down, there won’t be any impact to availability. By using a cloud provider, the only dependency is an internet connection on the client side. Information, data, and the application should always be available otherwise. Constant uptime and outage prevention are two reasons why most organizations opt for cloud hosting.
6. **Security**: Authentication services should be employed to identify each user who logs into the system. A trail of their actions should be kept for review and auditing purposes if there are any issues. A principle of least privilege should be employed so that any user only has enough access to the system to complete their tasks. For the average user, this will be nothing more than creating an account and playing the game. Linux itself utilizes firewall tools to control network traffic to secure data and protect against unauthorized access attempts, intentional or unintentional. Additionally, Linux supports encryption to further protect data.