

Draw It or Lose It

# **CS 230 Project Software Design Template**

**Joshua Ridel**

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 | <07/12/23> | <Joshua Ridel> | Executive Summary, Design Constraints, and Domain Model were all described and completed. |

**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 client, the Gaming Room, wants to develop a web-based game titled *Draw it or Lose it*. For the game to run as intended, it will require the ability to have one or more teams involved in a game instance, each team to have multiple players assigned to it, each team name to be unique to allow users to check if a name is already in use when deciding a team name, and for only one instance of the game to exist in memory at any time.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

Specifically being a web-based game, the game must be developed in a web-compatible coding language, such as Python or Java. The game will also need a method of storing data. Since it is a web-based game, and thus must always be connected to the internet, saving to the cloud would be appropriate.

Multiple teams and players will be implemented simply as objects. A singleton pattern will be used to check and ensure team names are unique. An iterator pattern will be used to create unique Ids for each of the game’s objects, which will allow for only one instance of a game to exist in memory.

## [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 Program Driver class runs the application, inheriting and utilizing the SingletonTester class to check for team name uniqueness. The Game, Team, and Player classes inherit from the Entity class, letting them utilize and share the Entity classes attributes. This allows for less repeating code. The Game, Team, Player, and GameService classes have a zero-to-many relationship, allowing each of them to run any number of instances of each other. This is required to check for new instances of Games, Teams, and 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)

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 several server options, including both the affordable Mac OS X server, and inherent Apache web-server command line support, giving users access to Python commands and additions. Both of these options would serve to support the server.  However, Mac does not have nearly as much 3rd party hardware configuration support as other OSs and server upkeep can grow more expensive in time. | Linux servers are open-source, meaning that they are very low-cost to upkeep and have a wide variety of support. Multiple different languages, 3rd party configurations, etc. Linux is also a trusted server brand, being used by many larger companies such as Google.  However, Linux is one of the more niche OSs, so it may be difficult to train or find a technician proficient in its use. | Windows offer servers that have been proven the easiest to learn and use and offer the most 3rd party support when it comes to their configuration. Web extensions, SQL, and ASP.NET support.  The downside to all of this is that Windows Servers are the most expensive to upkeep. | There is very little in terms of support for mobile servers.  While companies do exist to assist with mobile server implementation (Such as Oracle), the lack of any 3rd party support would require developers to write common code functions from scratch for multiple different mobile devices. |
| **Client Side** | Mac is one of the more expensive OS options, rivaling that of Windows.  However, it is also the second-most common OS, so it would not take sufficient time to learn and implement, nor would expertise on the system be rare. | Linux, due to its open-source nature, is easily the most cost-effective of the proposed OS’s.  However, it is a niche OS, thus time to learn and implement may be costly, and expertise/client support is rare. | Windows has the most configuration support of all the OSs and is highly regarded as one of the most efficient to implement, update, and use.  However, the monetary cost of implementation is high. | Mobile would not be costly to implement, and is possibly one of the most convenient and easiest to use due to its mobile nature.  However, it would take the most time to implement, due to requiring ground-up coding functionality, and compatibility for all the different mobile avenues (Apple, Android, Tablets, etc). |
| **Development Tools** | Mac heavily utilizes the Xcode IDE. Most code for Mac applications are written in Swift, though there is also Javascript support.  HTML support exists for user-interface. | Eclipse is a common IDE for Linux, thus it has support for both Javascript and C++. Another IDE is Atom.  Also includes Visual Studio and HTML support. | Windows makes use of both Visual Studio and Eclipse, giving it access to Javascript, C++, HTML, and SQL support. | Supported developmental tools depends on the particular mobile device.  IPhones have similar support to that of Macs, while Androids share Linux support. |

## 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**: I recommend Linux to be the OS of choice for The Gaming Room. Its open-source nature will keep server-upkeep and ownership costs low, while maintenance easy and swift. It has many supportive IDEs in-comparison with other platforms, resulting in simple and efficient use, while also cutting costs.
2. **Operating Systems Architectures**: Linux is a more modular system, only including core components and requiring separate add-ons for specific operations. This will allow the system to only contain what is required of it, resulting in lightweight code, saving space, and avoiding complexity. It's also worth noting most of Linux’s operations are triggered from system libraries separate from the core OS, resulting in better security.
3. **Storage Management**: Linux supports functionality for a direct access storage method, allowing for immediate access to stored data on saved disks. This would be supported by multiple servers which would all be accessible by the core driver. This method would result in the fastest possible loading times for data, which I find to be paramount regarding the system’s need to accurately load the many high-definition pictures for the game.
4. **Memory Management**: There are several methods that could aid in managing memory. First is to take advantage of the game’s engine of choice, utilizing its own library to save on our own system’s space. Secondly, the Linus OS will allow us to bypass physical memory limits and run multiple instances of the game (a requested feature) at a time by executing the game only partially in memory. This is done through utilizing demand paging and virtual memory. This also results in faster load times.
5. **Distributed Systems and Networks**: The best way to connect to multiple clients regarding Linux is through load-balancing and APIs. APIs are simple, light, and portable, easily allowing for separate applications across different platforms to communicate with one another. Combining the API with load-balancing to direct users to better servers when others are down, this multi-platform method also has solid connectivity. It's also worth noting the game’s engine of choice can also support access to other platforms, for the vast majority of game-engines are cross-platform by nature.
6. **Security**: As earlier stated, Linux operations are actually called from system libraries which are separate from the core components, allowing for more inherent security. It's also recommended to utilize gold standard multi-step user authentication methods, including login information, passwords, and security questions. A “role” system can also be used for extra security on top of user authentication. Each user is applied a role, either “player” or “administrator” which dictates what permissions the user has access to, from simply playing the game to actually making changes.