

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 | 05/26/2024 | Remy Welham | Updated Executive Summary, Requirements, Design Constraints, and Domain Model description. |
| 2.0 | 06/06/2024 | Remy Welham | Updated Evaluations on different platforms. |
| 3.0 | 06/21/2024 | Remy Welham | Updated Recommendations |

**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, would like to develop a web-based game on multiple operating systems based on their current game, Draw It or Lose it. This game is meant to have many teams with various players that all have unique names.

## Requirements

Draw It or Lose it is expected to have one or more teams in a game, for each team to have multiple players, and to have the game and team names be unique. The game also must have only one instance at a time by using unique identifiers. Hardware requirements will be updated at a later date.

## [Design Constraints](#_2et92p0)

-A game needs to be able to host one to many teams at the same time. The teams involved in the game need to have unique identifiers so the game can tell who has guessed, whose turn it is, and who gets a point. This can be achieved in software with programming each instance of a team with a unique ID and name that cannot be repeated.

-Each team needs to have multiple players assigned to it. Each player also needs to have a unique ID and name. In software, this can be programmed by checking each players’ name against a list of players in the team and when adding a new player, giving the player an ID.

-Only one instance of the game can exist in memory at a time. This will prevent games from conflicting with one another and keep memory usage low for the web-based design. Similarly to the team and player constraints, this can be solved with using a unique identifier for each game and using a Singleton Pattern to force only one game to be allowed at one 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)

In the UML model, we see the two driver classes, ProgramDriver and SingletonDriver. Both of these classes are meant for simple testing of the other classes to ensure that everything works. Moving into the other classes, we have the Entity class. This is the parent class of the Game, Team, and Player classes. This uses the Object Oriented Programing (OOP) principle of inheritance. Every class that inherits from Entity uses the id and name parameters in their constructors. Each class also overrides the toString method and can use the getId() and getName() methods which can be related to polymorphism. Game, Team, and Player all relate to each other and can communicate with each other. For example, Team has a list of players using the Player class. For abstraction and encapsulation principles of OOP of the classes, all of the classes have private, public, and protected variables. The private ones are the abstracted variables that don’t need to be touched outside of its own class. The protected variables can help encapsulate implementation of an object, but still allow a child class access. Finally, we have the GameService class which is used to host a Game.

**"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** | Macs are expensive when it comes to getting a license to use them for server creation. They can be used for servers, but you must have a MacBook to do any development. | Linux is a popular choice for web-based servers. It has equipment for doing so and is free in licensing which makes it a cheaper option. You can get Linux on different computers as well. | Windows is also expensive when it comes to getting licenses for use. However, Windows is a secure option. Windows is also generally easier to set up and use with its servers. | Mobile devices could be used for server hosting, but they aren’t made to do it well. They don’t have as much capacity or power needed to host a larger server. |
| **Client Side** | Macs are good for development because they have easy to use kits for them. They aren’t hard to use, but you cannot develop for Mac without having a MacBook. | Software development on Linux tends to take the most amount of time as Linux can be finicky for people who aren’t experienced in it. This could add to cost and expertise as well. | Windows is fairly low in cost for software development since a lot can be done on Windows systems. However, Windows development has to be done in specific ways that expertise would be a high requirement. | Mobile devices are different than the other three systems for development. The way users interact and how things are laid out often need to be in different formats. Costs could get higher because you need experienced developers working on mobile devices and they can take more testing. |
| **Development Tools** | Mac has a few different IDEs that work including Swift Playgrounds and Xcode. Most of the coding on Macs is in Swift without having to do work around for other languages. | A lot of programming on Linux can be done using Python, C or C++. Because of the lack of licensing issues with Linux, there are many IDEs that can be used. | Windows primarily supports Visual Studio Code for making applications on Windows. A lot of this would be done in C++, but Visual Studio Code and Windows can use many languages. | Mobile applications change depending on what kind of device you are developing the app on or the app for. With an iPhone, you have to have a Mac to develop the application using Swift. If it’s an Android, there’s the Android Studio. There is also the possibility of doing the development for an application on a separate machine using C++, but then the application would have to be migrated to an Android or iPhone. |

## 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 would recommend Linux as it is able to be used on different machines, is lower in cost, and has a lot more flexibility in how things are implemented. This also will help with connecting with other operating platforms later in development.
2. **Operating Systems Architectures**: Linux systems use Kernel as a way to communicate between the hardware and the software of the game. It will also help with managing the lower level functions of the system.
3. **Storage Management**: I would recommend cloud-based storage. Right now, the amount of images and code used to run Draw It or Lose It, is smaller. However, it has a lot of capability to grow and will need that growing space in the form of cloud-based storage.
4. **Memory Management**: The memory can be managed in the Primary Memory of the cloud storage with the images being moved into the Cache for use in the game. This can also be helped with using a Watcher to shift memory allocation based on usage.
5. **Distributed Systems and Networks**: With the systems being on the cloud, having the systems have different distributions and nodes helps to ensure that if there is a failing in one, it doesn’t cause the entire program to go down. This also allows for a shifting of servers more easily.
6. **Security**: Security will be best handled by the use of user authentication via passwords and accounts made for each user of the game. The other security practice that should be used is roles-based security which admins, players, games, and other roles getting different amounts of access.