

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 | 05/25/2025 | Sarah Warden | Executive summary, software requirements, and design constraints all filled out. Domain model and evaluation have been completed. Final recommendations were made. |
| 1.0 | 06/12/2025 | Sarah Warden | Updated evaluation table section. |
| 1.0 | 6/22/2025 | Sarah Warden | Addressed final recommendations for The Gaming Room to begin building their product |

## [Executive Summary](#_sbfa50wo7nsh)

Our client, The Gaming Room, would like to make their game, *Draw It or Lose It*, compatible with multiple platforms, as it currently only runs as an Android application. The app will draw renderings from stock images, each being completed at the 30-second mark, with the current team able to guess throughout the process. Once the drawing is finished, if no one on the current team has guessed correctly, the other teams have 15 seconds each to solve the puzzle. The game will consist of four rounds, each lasting one minute, and the puzzle will be a phrase, title, or thing.

## Requirements

* Games must be able to have one or more teams involved
* Each team will have multiple players assigned to it
* Each game and team name must be unique
  + Allows users to check whether a name is in use when choosing a team name
* Only one instance of the game can exist in memory at any given time
* The game must run on multiple platforms as a web-based design
* Games need to have time limits for rounds
* If a puzzle is not solved, the game needs to allow the other teams to guess within a new timed section

## [Design Constraints](#_2et92p0)

* Only one instance of the game can exist at a time – unique names need to be enforced and checked so that this holds true.
* Multiple players on each team – allows for solo team play for a less competitive version.
* 1+ team per game – game cannot run if no one is playing.
* Compatible with multiple platforms (Mac, Android, Windows, etc.) – game needs to be created as a web-based platform process with adequate security and network reliability.

## [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 Entity class is the parent class for the program, with the Game, Team, and Player classes inheriting the private id and name attributes from it. There is an optional zero to many multiplicity composition relationship from Game to Team and Team to Player, meaning that there can be zero, one, or many players on each team, and zero, one, or many teams in each game. The GameService class has a composition relationship with the Game class of zero to many as well.

The ProgramDriver contains the main program and uses the SingletonTester to check whether a game, team, or player name already exists.

**"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)

I will be evaluating the validity of using the main four platforms for development (Mac, Windows, Linux, and mobile devices). Below are the strengths and weaknesses of each platform choice.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Stable and well prepared for lots of hits while running. Great for Mac clients with high-level graphics. Most expensive hardware and pricey to upkeep.  While fine for the server side, it would be very expensive for the clients to find host sites if they don’t have a mac, therefore this would not be the best choice.  Need commercial license from GameMaker - $99.99  Some other games will have additional costs | Open source so cheaper relative to other platforms. Known for stability and great security. Highest processing power. Does not import well from Windows.  As the only opensource option, it is the cheapest for use and maintain as there are no licenses that need to be purchased, unlike Mac and Windows. There re also more options for Linux when utilizing cloud based providers. | Proprietary, making it a pricier option because of the licensing needed. Most compatible with other servers and add-on hardware that the client might use. Maintaining the server is relatively simple.  Most business computers are Windows based so there would be much less learning curve and no purchases necessary to acquire the products at the start. However, it is a proprietary server platform so there are annual costs.  Costs vary greatly, but include other platforms such as PlayStation | Supported by iOS and Android, however much more difficult to implement and is generally not utilized by users.  This would be the most difficult as the servers are not built to handle such games and traffic to the sites. An app that could be downloaded from the Apple or Android store would be a much better option if going the mobile device route. |
| **Client Side** | Most expensive hardware and software for clients. Generally, user-friendly operation. Not compatible with other servers without the use of a remote desktop environment.  Mac hardware and software necessary, must be purchased if not already used by the client. Languages used for coding are less known and will need lots of front-end time to learn. | Free to get for the client. Known to be the most difficult and least user-friendly platform, so a large learning curve and is time-consuming for the client.  Uses most common programming languages (e.g., Python, C++, Java). This is the least-used platform; however, multiple user support is available for a slightly quicker learning process. | Easiest to use as the server uses an environment with many similarities to a ‘home computer’. Mid-range for price and maintenance. The least time needed to learn the system.  Uses common languages as well as the most commonly used server on the market. Generally speaking, most businesses and homes either strictly use, or have access to Windows hardware. Meaning there would not need to be hardware purchased by the client. | Most diversity in platforms and the ability to play from anywhere. However, with laptops, this is generally true with the other platforms as well.  The software for mobile devices isn’t conducive to multiple user games. You also have the same issue of Mac vs Windows and that you would only be usable on one; however this market is much more greatly in Macs realm. |
| **Development Tools** | Languages: Python, C, C++  IDE: Xcode (apps), Visual Studio Code (Web)  Only uses C and SWIFT so getting new developers, or teaching new knowledge may be necessary. Also, there is a ~$100 annual license fee | Languages: C, C++, Python, Java, C#, JavaScript, PHP  IDE : Ruby, Code ::Blocks, JetBrains CLion  Since all of the most common languages can be used, there would be no need for new developers. Also, Eclipse can be a useful (and free) IDE for all languages, meaning no extra costs. | Languages: C, C++, JavaScript  IDE: Eclipse, Command Prompt, PyCharm  C is the most common language used (although other common ones are supported). Visual Studio Code is available and can run all of them; however it is not free. The base level will cost around $50 and can go run as high as $250 annually. | Languages: C++, JavaScript  IDE: Visual Studio Code  Depending on which mobile device you are going for (Apple or Android) the same benefits and deficits of Mac and Windows would apply. i.e.: $100 annually for Apple and more developers or knowledge, or $50 – 250 but no additional training needed. |

## 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**: Windows or Linux is recommended for the frontend software development. Windows computers are most common in businesses, so if that is what is already available, it would be best to proceed with the hardware already owned. However, Linux is not a proprietary brand and is therefore much cheaper in costs, both upfront and annually, so it would be the best option if Windows products are not already owned and used within the company.

Using Linux also gives the option to code in both languages, so that the program will run best on each platform. The use of Eclipse for an IDE would be beneficial as it is free and can handle most programming languages.

1. **Operating Systems Architectures**: Linux has four layers: hardware, kernel, shell, and application. The hardware handles the memory and storage with CPU, RAM, and input-output devices. The Linux CPU can work with all other structures of CPU, making it ideal when users may have different systems on their personal devices.

The kernel is the main part of the operating system. It is essentially the foundation layer. This layer manages the devices and processes the CPU, creating a bridge between all the resources and processes used. This section also handles the performance optimization, balancing all the memory, and maintaining system efficiency.

The shell is the part of Linux that interacts with humans, on both ends of the cycle. Essentially, it will take the commands written by the user and interpret them to the kernel in the operating system.

1. **Storage Management**: It is recommended that The Gaming Room uses a virtual server to cut down on costs, with the added benefit of more storage availability and greater security for users. This will also create the best option for both growth or scaling down in size in the future.
2. **Memory Management**: If we use the virtual server that is recommended, then Linux uses a combination of physical RAM and swap space for its memory management system. The virtual memory essentially creates an illusion that there is a much larger space to process, causing speeds to be faster and digital quality to be sharper. When there needs to be more room than the physical RAM can provide, then the swap space comes into play. The system will move the pages used less often to a swap space, so there is more physical RAM available for the necessary processing happening that moment.
3. **Distributed Systems and Networks**: In order for the game to run best on every system and server, cloud-based programs are being used most often. That is what I would recommend for The Gaming Room as well. A cloud-based system will allow for additional security for the user information on top of making sure that the game will run when outages occur. The way cloud systems work is that multiple servers are used at any given time, so if one goes out, the entire game does not collapse and cease to work. Given that this game will ideally be being played all over the world, we need it to continue to operate 24/7. Additionally, the front and backends will utilize RESTful APIs to communicate with each other. This will allow different servers and systems to work with each other from both client and server sides, meaning that Windows can talk to Mac, iOS, etc., and vice versa.
4. **Security**: The cloud-based server will lend a certain amount of security for users with the added benefit of it being handled by a third party and will not be the concern of The Gaming Room. Additionally, each type of user will have different levels of access to the information given to and stored by the game.

The users playing the game will have the lowest level of security clearance, meaning they will have access to only their own information. They will be able to enter player and team names, create a new game, and edit this information at any time.

The only people with ADMIN access to the information should be the specific employees at The Gaming Room.