

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.1

## 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 | 01/26/25 | Jeff Leazenby | Additional pages, all pages updated, project defined more clearly with recommendations and evaluation |
| 1.0 | 02/9/25 | Jeff Leazenby | Information updated for Evaluation |

## [Executive Summary](#_sbfa50wo7nsh)

The goal of the project is to create an application named Draw It or Lose It. This application will resemble a previous iteration of the game that was on the television in the 1980s. By using a library of stock images, the user will have to attempt to figure out the image by constant progression of the image and clues provided by the application.

## Requirements

* The game can have one or more teams playing
* Each team has multiple users
* Game and team names are unique and duplicates are not allowed
* Only one instance of the game can be functional at any given time

## [Design Constraints](#_2et92p0)

* Must be available on common platforms
* Image must be complete after 30 seconds
* After 30 seconds have elapsed, other teams have 15 seconds to attempt to guess
* As it is web based, browser compatibility must exists for common browsers

## [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 uses SingletonTester for running the main application and using the public testSingleton function of SingletonTester.

The class Entity has two private attributes of id and name and a private method Entity. The other methods are public, and the class Entity is a relationship directly to the classes Game, Team, and Player. It has inheritance that is passed down to the sub-classes below.

The class Game has a private attribute of teams and public methods for creating a game. It uses GameService, which has private attributes to help identify the next game information and methods to create the game/lobby. This class also inherits information from the Game class.

The class Team has a private attribute of players and public methods to create a team and and add players to the team. It also has zero to many relationships between Game and Player.

The class Player contains public methods for establishing players.

Overall Entity is a superclass that uses its relationship between all other classes to facilitate the application.

**"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 a server solution named macOS server. It is extremely costly when compared to other options. | Linux has very stable and secure server options. Certain methods of file transfer and other server functions are much easier to do because of its configurability. | Very commonly used. A strong contender when it comes to simple and standard applications, but not as configurable as Linux. | There are different options for servers and backend services for mobile devices even though they are not commonly used. Lower capabilities exist because of the rarity of using this option over more robust computers. |
| **Client Side** | Client-side operations are very user friendly. The main advantage is how easy it is to use with strong security. | Low mainstream compatibility with different common products. A variety of open-source libraries and tools exist for a large level of configurability. | Most used due to its variety of tools and high options for compatibility. Since it is the most common, it also tends to be a prime victim for malware and viruses. | It has a very strong presence in the market due to the wide adaptation of mobile devices in everyday life. Depending on the OS, different levels of configurability and security exist. |
| **Development Tools** | Does support cross-platform tools and has its own development tools. Lack of support for other windows platforms-based tools. | Extremely wide range and versatile set of development tools. Most common languages and tools are supported. | Standardized and common development tools. High integration and compatibility options for standard engines and applications. | Tools tend to be mobile-device specific as they have different operating systems. iOS and Android, for example, have their own suite of tools and libraries. |

**Summary of Evaluation:**

Licensing Costs:

* macOS Server tends to be high when compared to other options. We may have some difficulties when scaling as we try to expand the user bases which may require negotiations to arrive at the final price and may change over time.
* Linux does not have any licensing fees there will be cost associated with the physical components involved in hosting and support.
* Windows will have high costs as well as their pricing require payment based on the quality of the amount of hardware. Scaling will continuously raise the cost as the user base develops.
* Mobile will also have a high cost based on the scaling of the user base. One thing to note is that pricing is typically based on usage.

Conclusion: There is no low-cost, high scalability option. The choice for the lowest cost is likely going to be Linux but will require additional investment in, essentially, housing our own physical hardware for the data. Among the higher cost options, we recommend Windows Server for its standard applications for businesses and ability to scale.

**Windows Costs:**

* There is different pricing structures based on size, but there are options to pay for hardware and the number of users accessing the server. Another option is to use cloud hosting via Azure where the likely costs would be:
* $11,000 per million users monthly
* This would allow for scaling as app usage grows.

**Development Tools Concerns:**

There will be a handful of programming languages used in the execution of this project. C#, Java, C++, and Python will all likely be used for different purposes. Some will be for back-end development and others will be for performance and front-end development. Visual Studio will likely be the standard development tool used. Additionally, when it comes to security and username/data, among other data used for the app, we will likely need to utilize SQL Server as another development tool. While there are free versions of some of these development tools, we will need to license them. The reason is for enhanced safety, reliability, and productivity. It is unlikely that every single developer will use every IDE for developing this software. We can likely break down the groups into front end, back end, and server development. This will require the different groups to have access to the appropriate IDE and resources.

**Development Tool Costs:**

* There will be a balance between using two versions of visual studio, professional and enterprise. While professional is $1200 annually, the enterprise edition is $6000 annually. These costs are similar when looking at different versions and availabilities of SQL Server. Some decision making is likely needed based on what licenses will be procured for individual developers.

**Software Development Considerations:**

When the topic of compatibility is discussed, there are a few things that need to be identified. The first is what platforms are going to be supported. From the client feedback we are looking at supporting both iOS and Android alongside desktop usage. There are really two approaches to this: Use native tools for each platform or focus on cross-compatibility. There are options available, like React Native and Flutter, that allow for a single code base to be used on both. Some adaptation will need to be accounted for based on the differences of the two operating systems but this is likely to be the best path. It will cut down on development time and prevent the need to completely “redo” all the programming for a different OS.

When looking at the development of the app we are going to need to consider what our “standard case” is for the age and quality of the device. We don’t, for example, want the application to only be consistent on the newest generation of devices. There needs to be a certain degree of backward compatibility so that older devices, within reason, still have the minimum hardware requirements to effectively use our service. Significant testing is going to be required for the Android OS. This is because there is a much larger variety of hardware in that ecosystem when compared to iOS. The app will need to be tested on a variety of devices with different ages, resources, and screen size. This will still need to be carried out for iOS devices, but not to the same extent due to the limited nature of the products.

For testing there are resources that can be utilized beyond simply purchasing all the potential variants of setups. Detox, for example, can be used for mobile testing while Cypress can be used for browser compatibility on different computers and operating systems. The focus will need to be placed on promoting a strong user experience regardless of the platform being used.

## 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**: Our recommendation is to use Windows as the Operating Platform due to its ability to extend to other environments and its wide adoption and development options.
2. **Operating Systems Architectures**: Windows is a C++ based Operating System that presents a graphical user interface to users with a large library of maintained and regulated features and options.
3. **Storage Management**: Every iteration of Windows develops upon its ability to manage data and space. Modern versions of windows allow for streamlined management and redundancy alongside the flexibility of a variety of optimization options.
4. **Memory Management**: Windows has different memory management options to allocate memory differently depending on what the user needs to do. Windows 11 is particularly strong in this area due to its ability to automatically provide memory based on what software is taking precedence.
5. **Distributed Systems and Networks**: There is a trade-off with this aspect when it comes to Windows. While there can be bottleneck issues and some custom setup required for advanced functions, it is easy to use with simple tools. This ease of use is also extended to integration applications due to Windows being such a popular platform.
6. **Security**: Windows has a feature called “Active Directory” that manages users, permissions, etc. With regards to security, it is a benchmark in the industry with good solutions (Defender, Antivirus options, etc.) for common users. Depending on the user, there are also more advanced third-party options that integrate seamlessly with Windows.