

**Draw It or Lose IT**

# **CS 230 Project Software Design**

Version 2.0

## 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 | 12/10/23 | Luis Valdez | Based on their present game, Draw It or Lose It, which is only accessible as an Android app, we will create a web-based game that supports various platforms. |

**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)

"Draw It or Lose It" so far has built an Android version of the game and is now set for multi-platform expansion. The goal is to grow the application's other operating systems. For example, the OS systems they would need are Linux, Mac, and Windows, and they also can work on mobile platforms. One of the most important features of the game is to have prime performance. Next, the security of the game must protect the user from any vulnerabilities, and the user experience must be prestigious so that users enjoy playing regardless of the type of system they are on.

## Requirements

*Business Requirements:*

1. Move the Android App to a web-based platform to reach more users.
2. The development will include the ability to access desktops, tablets, and mobile phones.
3. Make sure to comply with user data protection and privacy laws.
4. The game needs to be based on the "Draw It or Lose It" design.
5. If a team fails to figure out the answer before the time runs out, another team will be given a chance to guess with a time limit.

*Technical Requirement:*

1. Each team must have several players.
2. Multiple teams need to participate in each game session.
3. Games and teams need unique names.
4. The program will provide system checks to make sure the team’s names are 1 of 1.
5. The Entity base class needs to be shared with the Player, Team, and Game classes.
6. The base class named Entity should be created.

There are many design constraints that can affect the SDLC. To begin, I believe that one of the key design constraints for developing the game app in a web-based distributed environment is knowing how to make the game function across several platforms and browsers. Another big constraint is knowing how to manage any network delays if they ever occur. The game flow of the app must be great if not users will be disappointed with the final product. Keeping the game state consistent plays an essential role throughout the life of the game for the long term. Also, over time as the game grows and more users join, the database needs to be created with organization being a high priority. Having the game database organized will help with data retrieval and storage. Lastly, I believe that knowing how to prevent potential security breaches and protect user privacy is a constraint that should not be dealt with lightly.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram shows the framework of “The Gaming Room application.” In the UML diagram below there are classes that have relationships, and object-oriented principles.

The “Entity” class is the base class in this diagram as it connects to the “GameService”, “Game”, “Team”, and “Player” classes.

The singleton pattern makes sure that each of the game-related steps is controlled via one instance of the "GameService" which also manages a list of Game objects/retrieves IDs for games, teams, and players.

The "Game" class acquires its properties from the base "Entity" class. It has a list of “Team” objects along with contractors on the bottom.

The “Team” Class also is an extension of the base “Entity” class that includes a list of Player objects with relationships. Constructors are included below.

Lastly, the “Player” class extends the “Entity” class. It comes with a constructor and a to String() function which does not have extra features.

"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 Server  Pro: It has strong security, an easy-to-use UI, and that is convenient for Apple users.  Con: potential compatibility issues with macOS software design.  Deployment: "Built-in web application hosting capabilities"  Licensing: Restricted to Apple hardware and it can be expensive. | Linux Server  Pro: One of the securities, and performance as a server platform. Also, operates a significant percentage of the planet's web servers.  Con: Need to have the right expertise to manage because of the system complexities.  Deployment: Many server platforms that can host web applications  Licensing: Open-source web hosting platforms for free | Windows Server  Pro: ease of use, works well with other Microsoft products, and works well with other platforms.  Con: expensive licenses to support.  Deployment:  support web application hosting.  Licensing:  Subscription based and can be costly, especially for enterprise-level solutions. | Mobile Servers  Pro: User Friendly, can be responsive with touch capabilities  Cons: Need to have the right expertise to manage because of the complexities. Also, can cost the most because of cross-platform development needs.  Deployment: Not used as server platforms.  Listening: N/A - not servers. |
| **Client Side** | Cost: More expensive expenses are caused by the need for specific Apple hardware and macOS software.  Time Expertise: May take longer because Apple's ecosystem is unique but, it is consistent with its environment. | Cost: Open-source programming tools help provide a client-side engineering platform at an affordable cost.  Time Expertise: Need to have the right expertise to manage because of the system complexities. | Cost: Thanks to free resources and developers with a Windows background, the user base makes it less expensive.  Time: Different Windows versions could boost testing and development time. | Cost: Costs may rise owing to the need for multiple platforms and tools/expertise to develop.  Time: Development time can be long, also checking throughout a lot of devices and OS versions will take a lot of time. |
| **Development Tools** | Programming Languages: C++/Java.  IDE’s: Eclipse, XCode & Visual Studio  Impact on Team: Macs require devs with expertise | C++, Python, Java.  IDE’s: Eclipse, NetBeans, Atom, Visual Studio  Impact on Team: Need developers that have exp. with Linux environment. | Python, C#/C++, Java/JavaScript.  IDE’s: VB.NET, Sublime text, Visual Studio, Eclipse, Atom, NetBeans.  Impact on Team: Since it is most common finding devs is more common than any other OS. | Programming Languages: Python, C#/C++, Java/JavaScript  IDE’s:  iOS: XCode, Swift for iOS apps  Android: React Native/Flutter, Android Studio, Visual Studio, and Eclipse.  Impact on team: Need devs that can use multiple cross-platform framework. |

**Client-Side Evaluation**

My compatibility requirements are to develop a user-friendly experience and ensure functionality across web and mobile browsers. My development consideration is to have devs that have experience with HTML5, CSS3, and JavaScript. Cost Implications will include the licenses for cross-platform development tools. To ensure the application is compatible with all web browser platforms and mobile devices I would need to guarantee the merger changes into the production codebase and deployment pipelines. I can also include the use of RESTful APIs because they provide a way for the client-side application to communicate with the server. Including an accessibility feature would help those who are disabled to be able to use the game app. Lastly, using HTTPS would be able to safely transfer data across all devices. I believe using HTTPS is essential because HTTPS encryption protects sensitive information. The likelihood is that the application will function seamlessly across different browsers and mobile devices, providing a consistent user experience.

## 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**:

In my opinion, I recommend using Linux because of its reliability, and scalability. The Gaming Room is trying to utilize a cross-platform approach therefore, Linux provides an efficient route to success.

Some examples of operating (server) platforms that will allow The Gaming Room to expand Draw It or Lose It to other computing environments would be Amazon Web Services (AWS), and Google Cloud Platform because these are examples of cloud-based computing services. Cloud solutions are great for online games. There are many reasons, but I believe that from a developer standpoint, they provide us with the ability to have scalability and reliability of the game. and savings on any type of hardware management cost.

1. **Operating Systems Architectures**:

Remotely in a cloud setting, Linux OS architecture design brings flexibility, security, and trustworthiness. However, multi-platform architecture will be visible through internet browsers the OS default browser. For example, on my MacOS I have Safari, on my Dell for work, I have Chrome for Linux, and Edge on Windows (IDE) for university.

The details of my chosen OS which is Linux are that I will be able to have compatibility with web-based applications and access to a large community of developers. This convenience will give me the benefit of having access to other developers who are like-minded and know what they’re doing on Linux.

1. **Storage Management**:

From a personal outlook, I prefer to use Google Cloud for storage management as it is flexible and free to use for a decent amount of storage space. Although upgrading the plan may be needed in the future, we can start to leverage it via Google Cloud for free in the beginning. This is very helpful when operating on a budget.

Google Cloud for storage management, focusing on its scalability is an added benefit that comes with the ecosystem. Furthermore, Google is one of the top companies on the planet, so getting support from the best wouldn’t be a bad idea when it comes to learning. Lastly, Cloud SQL can be used for "Draw It or Lose It".

1. **Memory Management**:

As far as memory management, I think that using disk space as an extension of main memory will be beneficial to save on costs down the road. Web browsers can also contribute to memory management throughout the process. Garbage collection can be utilized to help automatically reclaim memory that is not used by the app.

Linux uses memory management techniques like using more memory than is physically available on the server. This allows the gameplay to run smoothly. Even over time as the demands on the system increase. The system would be able to withstand the changes. Memory management and the garbage collector in the Linux work together. This would automatically release unnecessary memory, which makes the software better.

1. **Distributed Systems and Networks**:

A distributed software system makes it possible for the "Draw It or Lose It" software to communicate between multiple platforms. Also, the distributed system allows the ability to increase the number of app users, inquiries, and data. To consider the dependencies between components within the distributed systems and networks is the fault tolerance. I like the fact that I can log in on different devices and still access the same files regardless. These are the outage failures that happen on the network. Distributed software and the network the app connects to are extremely important because problems such as loss of vital information, conflicts, and accessibility could happen.

I believe that having cross-platform framework capabilities would be the key to this aspect. Utilizing frameworks that support cross-platform compatibility, such as React Native would be great for achieving success in distributed software and the network that connects the devices. Two components I would say these dependencies rely on are RESTful APIs and Security. I will begin with security because of the components within the distributed systems and network. For example, for security purposes, encryption is critical for protecting data. Data travels across different networks all the time, specifically because gaming platforms could have sensitive customer data. RESTful APIs allow client-server communication which helps different game components to give and receive data normally. Lastly, servers can run without requiring remembering the game's state.

1. **Security**:

At Creative Technology Solutions, security is a main priority as the influx of breaches has increased dramatically over the years. The procedures to defend The Game information across platforms I believe are essential first, would be who has authorization to access client data. The second is encryption because the data will be in an unreadable form that uses a secret key, and it can prevent the data from being stolen. Lastly, adding another layer of security such as passwords or Google/Microsoft authenticators will boost safety.

Using network security is foundational for the SDLC to succeed. I believe that monitoring and managing security using firewalls would be a great step to keep track. This can help in the prevention of hackers finding an illegal way to access a private network. Next, I believe that running penetration testing on a regular basis to detect security vulnerabilities before they may be exploited could be a massive for the app to be around for the long term. I am firm that finding out more way to stay protected because they will pay off later as technology grows. That is why I believe that in the case of a security breach, it is critical to have a plan in place to combat any damages quick.

**Summary**

I would recommend "The Gaming Room's" multi-platform OS to be a Linux-based server. I recommend Linux because of its commonality across different platforms. Also, from my experience thus far the reliability of Linux is great. The scalability aspect is also an advantage since there are more Linux developers as opposed to MacOS, which could be much more expensive down the line. I believe that free-to-use tools like cloud-based storage solutions, like Google Cloud, will need to be implemented. Google Cloud is flexible and easy to use when sharing information among team members. Security protocols will assist with the encryption, and multi-factor authentication to protect user data comprehensively. In the end, developers will be able to assist in giving the users a secure experience across all web browsers and mobile devices.