

<Draw It or Lose It>

# **CS 230 Project Software Design Template**

Version 3.0

## 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  2.0  3.0 | <07/13/2023>  <07/28/2023>  <08/11/2023> | <Joseph Kawamoto >  <Joseph Kawamoto>  <Joseph Kawamoto> | <Brief description of changes in this revision>  <Filled out the Development Requirements Table for Mac, Linux, Windows, and Mobile Devices.>  <Filled out recommendations section for application.> |

**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 goal of this project is to create a multi-platform game/app called, “Draw It or Lose It.” The game is currently only available on Android devices and was developed natively for that device. To ensure that the application is accessible from multiple platforms, they have proposed a web-based game application to make it available on computers, tablets, and other smartphones.>

## 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)

<**Cross-Platform Capability**: The main objective of developing a web-based game application is to ensure that it works on multiple devices. This ensures that users across all platforms will have the same in-app experience, and that the game can be accessed regardless of the device that the user has (within reason). To incorporate this functionality, differences in hardware need to be accounted for. Screen-size, RAM, and other limitations need to be considered.

**Scalability**: The game instance needs to be able to scale to accommodate up to a certain number of players. Multiple users connecting to the same game instance could be problematic, and we will need to make sure that photo rendering, as well as answer submission all have quick response times to ensure a smooth gameplay experience. This is also a technical constraint, increasing user traffic will make the application run slower than acceptable if proper precautions are not taken. We should incorporate some sort of cache system for quick data retrieval.

**Game Instance Hosting**: Will the game have dedicated game servers, peer-to-peer networking, or maybe a mixture of both? To provide a similar experience across all possible platforms, and a multitude of signal strength, I suggest either dedicated game servers or a mixture of both. This technical constraint is here to ensure a seamless gameplay experience.

**User Information Security:** Because the application is web-based, it needs to be able to handle user authentication and protect users’ privacy. This is a technical constraint. Authentication procedures such as password validation, and possibly data encryption along with routine security checks to ensure user information is safe.

**Network Requirements:** Since the application will be web-based as well as multiplayer, a consistent network connection is required to play the game. The application’s bandwidth requirements need to be lightweight, considering that the application will be designed to work on home computers as well as mobile smart phones, this is a technical constraint. The UI will also need to communicate any network problems or disconnects to the user, so they are not left wondering what is happening. The main idea of this constraint is to provide as smooth a user experience as possible.

**Accessibility Options:** The entire point of this project is to make this game available to as wide an audience as possible. To do so, the game should offer accessibility options to those who may need them. Speech-to-text, UI element customization, etc. should be considered.>

## [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)

<**Program Driver:** The Program driver class depends on the SingletonTester as depicted by the filled arrow and <<uses>>

**Entity**: The Game, Team, and Player class all rely on the entity class.

**GameService**: The GameService class is associated with Game. They share a multiplicative relationship and the line between them indicates that there can be 0 or more than one instance of the Game class.

**Game**: The Game class is associated with Team in the same way that GameService is associated with Game. The 0…\* indicates that each Game class can be associated with 0 or more Team classes. The Game class is associated with the GameService class in the opposite way that it is related to Team. The Game class is also dependent on the Entity class, as depicted by the open arrow.

**Team**: The Team class is associated with the Game, Player, and Entity Classes. The relationship that it share between Game and Player and multiplicative. The Team class is dependent on the Entity class.

**Player**: Is associated to the Team class and is dependent on the Entity class. >

The UML Diagram displays the OOP principals of encapsulation, inheritance, association, dependency, as well as the Singleton Design Pattern. Each class I the diagram encapsulates attributes for its class. The Game, Team, and Player classes all demonstrate inheritance by being dependent on the Entity class. They will “inherit” things from the entity class. Association is shown in the relationship between GameService and Game classes, as well as the Game, Team, and Player classes. This allows for interaction/collaboration between different instances of those classes. Dependency is shows in the ProgramDrivers dependency on the SingletonTester, as well as Game, Team, and Player classes dependency on the Entity class.

**"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** | Web-based application hosting is possible, but uncommon on Mac platforms. Mac is more commonly used for client-side development. Although Mac does offer web-server hosting and can do so. It lacks the optimization to handle heavy server-side traffic and scalability like the other platforms offer, there may also be additional configurations and things to learn since Mac is proprietary. | Linux is a good choice and is very popular for server-hosting. Linux is known to be secure, stable, and scalable for thousands of potential players. The distributions are designed with server applications in mind. Also, since Linux is Open Source, the community support (while less formal and official) is massive. Linux is also more cost effective than Mac and Windows because it is Open Source. | Windows is another proprietary OS and is owned by Microsoft. Windows is the most popular operating system in the world, it dominates the desktop computer market. The price for licensing will inevitably have an impact on the overall budget of the project. Windows is also more resource intensive than Linux. | Mobile devices should not be considered for server hosting. But can be used to access the web-based application itself via mobile web browsers. When developing to optimize for Mobile Devices, varying screen sizes should be accounted for as well as touch screen interaction. |
| **Client Side** | To make sure that the application is compatible with all internet browsers and platforms, developers should run tests using these popular applications. For Apple devices, there should be special care taken to ensure that the application works well with trackpads and touchscreen as well as the Retina Display. Additional testing should be done to ensure that the user experience is consistent across all Apple products using Safari. Team members should include people that are familiar with Xcode and Swift development, the native coding language for Apple application development. | Developers should ensure that the application works consistently across all popular web browsers and ensure responsiveness on multiple desktop environments. Mobile development should prioritize android devices, that is who is more aligned with the Linux userbase due to both using Open Source. | Common web browsers on Windows include Chrome, Firefox, Edge, and Internet Explorer. Tests should be run to ensure that the application experience is consistent across all popular web browsers on the platform. The application should work seamlessly across all versions of windows and should work with a multitude of screen sizes and resolutions. | The largest market will be the mobile device market. To achieve compatibility across all mobile devices, we should prioritize design that adapts to different screen sizes and screen orientations. Tests need to be run on all major smart phone operating systems to ensure that the application runs on both iOS as well as Android. Heavy testing on touch screen responsiveness across all platforms should be done as well. The application should work just as well with both Safari and Chrome, the two most dominant browsers on iOS and Android. |
| **Development Tools** | Teams will need to consist of people that are familiar with HTML, CSS, JavaScript, and perhaps Swift. HTML, CSS, and JavaScript individually help to create a web-based application. Multiple teams with members from different backgrounds will be required. HTML, CSS, and JavaScript are standard and free to use. For Native app development on Mac, teams should include members who are familiar with Xcode and Swift development to create applications native to Mac and iOS. | For the web-based application portion. Development tools and languages should remain consistent. This means using HTML, CSS, and JavaScript for Linux development as well. Publishing applications on Google Play Store as well as apple store may include fees. | Suggested Languages/Tools for Development: HTML, CSS, JavaScript, Swift. Consistent with all the other web-based app development tools listed in this row so far. Licensing fee for developing applications native to Windows should also be considered. | For app development can be marketed on the app store. The client should consider adding team members that are familiar with Xcode, Swift, Kotlin, and JavaScript to port the current Android only application to iOS as well as traditional desktop Operating Systems. Fees associated with both marketplaces should be considered. For example, Apple charges a yearly subscription fee for application publication. |

## 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 believe that the best platform to expand the “Draw it or Lose it” game application would be Linux-based server platform. I believe that Linux will be the best fit for us because the platform is known for its affordability, scalability, security, and flexibility. It also is the most compatible platform for a multi-platform expansion like the one we are planning. Linux is a reliable platform that can operate under heavy loads, this will be important for the online aspect of our game. There will potentially be thousands of players logging in and playing at the same time. We need to make sure that whatever platform we host the application on can handle the amount of information that will be flowing between the server and the player’s game clients. The open-source nature of Linux allows developers to customize the platform to tailor it to the application that is being built and for the multitude of computing environments that the game will be played on. Linux is known for having robust security features, this is important for our web-based application to protect the user data.
2. **Operating Systems Architectures**: Linux operating systems use something called a monolithic kernel design. A kernel is the engine of an operating system, and its main function is to communicate to each piece of hardware in a computer system. A monolithic kernel has all these important functions and instructions packed away into one box. What this means is that all the necessary and essential components of the system are all interwoven into a single executable model that runs in kernel mode. Because of this implementation, in monolithic kernel, all the functionalities are integrated into a single piece of software. All the kernel components exist within a single address space and can access each other’s functions. This allows for quick communication between each of the components, which will be perfect for us to keep the user gameplay experience fast paced. These types of systems are known for their efficiency in system call performance.
3. **Storage Management**: To store game and player data, I suggest using a combination of HDFS and a database. HDFS stands for Hadoop Distributed File System, and it is a storage system that can handle large files. HDFS is designed to work across multiple computers. We can use a relational database to store uniform information like player ID, name, email address, and other login information. And we can implement a NoSQL database to house information that is not uniform. This can include messages between players, notes, etc. Things that may vary in length and size and do not fit neatly into any pre-determined boxes. By combining these three things, we can keep big image files and graphics secure using the HDFS, while also organizing player information and game progress by using databases.
4. **Memory Management**: Proper memory management will ensure that the game runs smoothly on all devices. To help better manage the limited resources that certain user platforms will have (phones especially) we need to implement proper instance management, resource allocation, image caching, and memory organization. Each instance of the game will be unique as will the ID for each team and each player on each team. We can use naming conventions to group teams and team members together when a new game instance is created. This will help us keep track of how many active game sessions, teams, and players are using the app. Since each entity will have a unique ID, this limits the likelihood of conflicts in memory. Each time a new game instance is created, the system sets aside a block of memory. This will ensure that each session has enough memory to run effectively. After a session is completed, the allocated memory will be reclaimed by the system, ensuring that memory is not being wasted in an empty game lobby. I also recommend that we use image caching for images that are used frequently (i.e., logos or even the most popular stock images from the library) should be kept in cache for quick retrieval. This will reduce the need to load the same images repeatedly, increasing the smoothness of the user’s gameplay experience.
5. **Distributed Systems and Networks**: A distributed system and network can be thought of as a group of computers working together rather than one main computer handling all tasks. Each task is broken down and handled by smaller computers on a linked systems network. When a user plays the game, their device becomes one of the “computers” in the system that handles processes. Because the goal is to ensure that the game is available to multiple platforms, the different platforms need to be able to communicate with each other. This is where a distributed systems and network (DSM) will come in handy as it acts as a translator between the various platforms. It establishes a common language that every device can understand. The shared data between devices will be in the form of player actions and moves within the game (making a guess, sending a message, etc.) When the user makes an input, this input is sent to the central computer system which then communicates that information to the devices of the other players in the same game instance. The DSM will ensure that each device connected to the game instance is in the same “game state.” Meaning that whenever a player makes a move or and input, the rest of the player’s screens are updated with that information in real time. This will enhance the user experience by giving them a smoother feeling game. Because the system is distributed across multiple computers, if one goes down, it does not take the entire system and game network down with it.
6. **Security**: Linux is known for strong security features, that is why I believe it to be the best option for us since the game also needs to be multi-platform. Here are some of the built in Linux features that we can implement. Linux implements end-to-end encryption which will scramble to user information as it is transferred between the multiple computers in the distributed systems network (DSM). This way, even if someone were to intercept the information somehow as it was transferred between computers, they wouldn’t be able to tell what the information contained without the decryption key. Similarly, Linux also provides this level of protection and encryption to user data that is stored on user accounts. When a user logs in for the first time, they are given unique credentials that will serve as their “key” to enter the game and access their account information. Linux also provides ways for us to control who has access to the game as well. Linux also allows us to monitor the application, and should anything happen that is deemed suspicious, an alert will give administrators the opportunity to respond.