

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 | 06/01/2025 | Jasmyne Fisher | This revision includes explanations for evaluating Mac, Linux, Windows, and Mobile Devices as platforms for developing and hosting web-based applications. Each section now addresses key areas such as system characteristics, software development considerations, and relevant programming languages and tools. |

**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 Gaming Room currently has a guessing game called **Draw It or Lose It**, and now they want to make a version that you can play on any device such as a computer, phone, or tablet. The game is simple; teams look at a group of stock photos and try to guess what the puzzle is. Each game has four rounds and if one team can’t guess the answer in time, the other teams get a chance to guess. When it comes to the technical side, to build this game for the web we need a plan that keeps everything flowing in an easy and user-friendly manner. We need to let people be able to make teams, choose topics, make sure there are enough players and there are no duplicate player names. This document explains how we’ll set everything up using efficient coding methods that help keep the game fair, organized, and fun to play online but also easy to navigate.

## Requirements

* The game should allow **one or more teams to play.**
* Each **team should be able to add multiple players.**
* Every individual player and team name must be different to keep from duplicates.
* Only **one main game system** should be running in memory at once.
* Each **game, team, and player** should get a **special ID number** so they don’t get mixed up.
* The game should work in a **web browser** so people can play on phones, tablets, and computers.
* The system should be designed in a way that makes it easy to expand or update later if needed.

## [Design Constraints](#_2et92p0)

* **Web-Based Only:** The game must run in a browser as well as the phone.
* **Devices:** The game should work on phones, tablets, and computers. So the design has to adjust to different operating systems and desktop/mobile views.
* **Unique Names/IDs:** Each game, team, and player needs to have a unique name and ID number. This prevents confusion and keeps the data clean and easy to follow.
* **Online Play:** Since people will play from different locations using the internet, we need to make sure the game can handle slow connections, timeouts, and data moving between all types of devices and the server.

## [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 UML diagram for *The Gaming Room* shows how different parts of a game program work together, there’s a main class called Entity, and other classes like Game, Team, and Player all get their basic features from it. A process like this helps save time and avoid repeating code. The GameService class keeps track of all the games, teams, and players, it uses the singleton pattern, so everything stays organized and in one place. The diagram also shows that each game can have many teams, and each team can have many players. The ProgramDriver starts the game program and uses SingletonTester to make sure the GameService is working correctly. This design uses smart programming ideas like encapsulation, inheritance, polymorphism, and abstraction. These ideas help make the program easier to build, fix, and grow.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is a stable and secure system because it's built on Unix, which helps protect it from crashes and viruses. It has great tools like Xcode that make it easier for developers to build and test web apps. However, Mac is not commonly used for hosting web-based applications, so there’s less support and fewer resources available. It can also be expensive and may not work well with some server tools used by big companies. | Linux is a popular choice for hosting web-based software because it's **free**, **secure**, and **very stable**. It works well on many types of computers and is trusted by big companies to run websites and apps. One big advantage is that it's open-source, meaning developers can change it to fit their needs. However, Linux can be harder to learn, especially for people who aren’t used to using command-line tools instead of a regular desktop. | Windows is a common choice for hosting web-based software, especially in businesses that use Microsoft tools. It supports programs like **.NET** and has a user-friendly interface with good technical support. One advantage is that it's compatible with many types of software and hardware. However, Windows can be more expensive and is more likely to face security issues like viruses if not managed carefully. | Mobile devices are not usually used to host web-based software because they have limited power, storage, and battery life. They are designed more for using apps than running them for others. While they can run small local servers or connect to cloud systems, they’re not strong enough for full-time hosting. Their biggest advantage is mobility, but they aren’t reliable or secure enough for hosting big web apps. |
| **Client Side** | Making apps for Mac can cost more because Apple computers and tools are expensive. It also takes more time since you have to follow Apple’s rules and make sure the app works on things like iPhones and iPads. You need someone who knows how to use Apple’s coding languages, like Swift, to build the app the right way. | Developing software on Linux is usually **low-cost** because the system and many of the tools are free to use. It can save time for experienced developers, but may take longer if the team is not familiar with Linux or command-line tools. Developers need to have some technical knowledge, especially since Linux works differently depending on the version (called a "distribution") being used. Supports all modern browsers and its responsive design ensures it looks good on any desktop or mobile browser. | Developing software on Windows can be more expensive because of licensing fees for the system and tools like Visual Studio. It may take more time if you need to test for different versions of Windows or connect to other systems. Developers also need experience with Microsoft technologies like **C#**, **.NET**, and the Windows environment to build and support apps well. | Developing for mobile devices can be expensive and time-consuming because you often need to build separate versions for **iOS** and **Android**. It takes extra work to make sure the app runs smoothly on many different screen sizes and devices. Developers need experience with tools like **Swift** for iOS, **Kotlin** or **Java** for Android, or cross-platform tools like **Flutter** and **React Native**. |
| **Development Tools** | To build software for Mac, The main tool they use is **Xcode**, which is Apple’s official app-building program (IDE). Other helpful tools include **Terminal** for running commands and **Homebrew** for installing extra software. These tools help developers create, test, and fix apps made for Mac computers. | To build software for Linux, developers often use programming languages like **Python**, **Java**, **C++**, and **JavaScript**. Common tools include **Visual Studio Code**, **Eclipse**, and **NetBeans** for writing and testing code. Developers also use the **Linux terminal** and package managers like **APT** or **YUM** to install and manage software easily. | To build software for Windows, developers often use programming languages like **C#**, **C++**, and **JavaScript**. The main tool used is **Visual Studio**, which is a powerful IDE for creating and testing apps. Other helpful tools include **PowerShell** for running commands and **.NET Framework** for building web and desktop applications. | To build apps for mobile devices, developers use **Swift** for iOS and **Kotlin** or **Java** for Android. The main tools are **Xcode** for iOS development and **Android Studio** for Android. For building apps that work on both platforms, developers often use tools like **Flutter** or **React Native** to save time and effort. |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Recommendations

Analyze the characteristics and techniques specific to various systems architectures and make recommendations to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: Linux is a great choice for expanding Draw It or Lose It because it’s free, secure, and works on many devices like servers, desktops, and even phones. It supports tools that make it easier to build apps for Windows, Mac, Android, and iOS. Since it’s open-source, developers can customize it to fit the game’s needs.
2. **Operating Systems Architectures**: Linux is built in parts, which makes it flexible and easy to adjust for different devices. Its core, called the kernel, handles memory, programs, and hardware. This setup keeps the game running smoothly and allows quick fixes without restarting the system.
3. **Storage Management**: Linux works well with EXT4, a fast and safe file system that keeps data organized. It helps the game load quickly, save progress, and avoid losing files. It also supports backups and limits who can access certain data for extra safety.
4. **Memory Management**: Linux uses virtual memory to keep things running if the system runs low on space it breaks memory into smaller pieces and loads only what’s needed, helping the game stay fast. It also keeps programs separated so one doesn’t crash another, they are no depented on one another
5. **Distributed Systems and Networks**: *Draw It or Lose It* can run on many devices by using a distributed system, where different parts of the game work on different computers. They stay connected over the internet, and if one part goes down, the others keep working. Tools like cloud servers and backups help everything stay online and smooth.
6. **Security**: Linux keeps user info safe using firewalls, permissions, and regular updates. It protects data with encryption and secure connections like HTTPS. Only approved users and programs can see or change files, keeping everything private, in one place and secure.

References

Lama. (2024, December 1). *Windows vs. macos vs. linux: The ultimate operating system showdown*. Nahil. <https://nahil.com.sa/windows-vs-macos-vs-linux-the-ultimate-operating-system-showdown/>

*What is linux?*. Red Hat - We make open source technologies for the enterprise. (n.d.). <https://www.redhat.com/en/topics/linux/what-is-linux>