

Draw it or Lose It

# **CS 230 Project Software Design**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design** 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 4**](#_Toc115077324)

[**Evaluation 4,5,6**](#_Toc115077325)

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/13/2025 | Steve Halliwell | First draft of design |
| 1.1 | 10/01/2025 | Steve Halliwell | Updated Project Two requirements |
| 1.2 | 10/15/2025 | Steve Halliwell | Updated Project Three requirements |

## [Executive Summary](#_sbfa50wo7nsh)

The purpose of this software design document is to outline the updated solution for developing the web-based game application, Draw It or Lose It. The game is now expanded to support multiple platforms. Creative Technology Solutions has asked us to move the game from its current Android-only environment into a multi-platform that works across all various desktops (Windows, Mac, Linux) and mobile devices (iOS, Android).

This new and improved system continues to use object-oriented programming principles and design patterns to ensure compatibility and reliability across systems. The Singleton pattern ensures only one active game instance exists in memory at a time, while the Iterator pattern manages collections of players, teams, and game rounds.

Project Two also looks at how the app works across different devices and platforms at the same time, so it can run smoothly no matter where it’s used. Since players will be playing on different devices at the same time, the game needs a client-server setup to keep everything in sync. It also must be secure and work on multiple platforms, so things like logins, encrypted traffic (HTTPS), and safe storage of game data are important.

This updated design provides Creative Technology Solutions with a roadmap for expanding Draw It or Lose It while maintaining the core integrity of its gameplay, supporting multiple devices, and preparing for a cross-platform deployment.

## Requirements

* This design needs one or more teams to participate in a game. Each team must have multiple players assigned to it.
* The game rounds should have specific time limits, such as one minute per round, with drawings
* progressively revealed until fully complete at the 30-second mark.
* 6. If a team fails to guess the puzzle within the given time, the remaining teams should have a
* chance to offer one guess each within a 15-second time limit to solve the puzzle.
* Team names must be unique so players can easily identify their team.
* The final game should be built from Android-only to a web-based application to one that supports multiple platforms.
* Gameplay needs to follow the original rules of four rounds, one minute each, with drawings revealed over time and guess opportunities for other teams if the timer expires.
* Only one instance of the game can exist in memory at any time.
* Each game, team, and player must have a unique identifier.
* The application must use object-oriented programming principles

## [Design Constraints](#_2et92p0)

* Developing the game application in a web-based distributed environment requires good preparation and attention to details. One constraint to keep in mind is that only one instance of the game can exist in memory at a time. This is done with the use of the Singleton design pattern to ensure a single, controlled instance.
* Also, game and team names must be unique and there must be a way for checking existing names before creating new names.
* Furthermore, the application must support multiple players and platforms, therefore it must be scalable, and adaptable to different operating systems. These constraints impact the games development by requiring careful planning and attention to detail.

## [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 class diagram for “Draw It or Lose It” shows how the main classes in the application will interact with each other. The UML class diagram demonstrates several object-oriented programming principles, including inheritance, encapsulation, and abstraction. For instance, the Entity class is the base superclass and has common attributes like id and name. These are inherited by the Game, Team, and Player classes. A Game contains multiple team objects, and each team contains multiple player objects. The GameService class is the heart of the game instances and uses the singleton pattern to make certain that only one instance exists in memory at a time. Additionally, the ProgramDriver class has a dependency on the SingletonTester class, as illustrated by the “uses” arrow, it is a subclass of that class, and it inherits attributes and behaviors. The ProgramDriver class is the starting point of everything and where the main function resides. The SingletonTester class is used by the ProgramDriver to verify that the singleton pattern is correctly implemented. Through OOP, everything works together and ensures the game application is working as intended, organized, and meets the software requirements.

**"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)

Linux feels like the best choice for hosting the server because it’s stable, secure, and free for the most part. It is capable of handling lots of players without too much trouble, but we’ll still need to test it under heavy load.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers, built on Unix, provide a stable and secure environment for web apps and can be used for server-based deployment. They come with developer-friendly tools like Xcode, but scalability is more limited compared to Linux and Windows. | Mainly open-source and is highly customizable, Linux servers are known for their stability, security, and scalability. Linux is widely used in the industry for hosting web apps, though they rely more on command-line management and have some GUI limitations. | Windows servers have broad software compatibility and a strong developer ecosystem, with lots of documentation available. They are easier for some developers to manage but tend to have slightly more known security vulnerabilities than Linux or Mac. | Mobile devices themselves don’t host servers. They act as clients connecting to the main server, with limited resources and performance depending on the device. Which means we really need to be careful with the client-server side of things. |
| **Client Side** | Mac interfaces are intuitive and user-friendly, reducing the learning curve for users. Developing for multiple clients may require more effort and diverse expertise. | Supporting multiple clients on Linux is cost-effective and stable but requires developers familiar with Linux commands and system management. | Windows clients are straightforward to develop and test, but licensing costs can be higher than open-source alternatives. Developers must ensure compatibility with Linux and Mac clients. | Mobile apps require responsive design for various screen sizes and must account for different operating systems (iOS and Android). Leveraging native features like cameras, GPS, and push notifications is important. |
| **Development Tools** | Node.js, JavaScript, and HTML/CSS are commonly used. IDEs like Visual Studio Code and Xcode are standard for development. | C, C++, Python, and Java are common languages. Tools like VS Code, Sublime Text, and command-line package managers help with development. Docker or virtual machines are often used for cross-platform testing. | C#, .NET, and Java are commonly used for Windows-based web apps. IDEs like Visual Studio or JetBrains are popular, and Docker/VMs help with testing across platforms. | Java/Kotlin for Android, Swift/Objective-C for iOS, and HTML/CSS/JavaScript for web-based clients. Emulators and simulators are used for testing across devices. |

## Recommendations

1. **Operating Platform**: For hosting the main server of *Draw It or Lose It*, Linux is a good choice. It’s stable, secure, and works well for web-based applications. It’s mostly free, which is nice for the budget, and it should handle lots of players without too much trouble. We’ll still need to test it under heavier loads to make sure performance stays smooth. Linux is stable, secure, and free, and it’s basically the standard in the industry for web servers (What Is a Linux Server and Why Does Your Business Need One?, 2025). Plus, as someone who’s built PCs from scratch, I appreciate that Linux doesn’t try to lock you into hardware.
2. **Operating Systems Architectures**: Using a 64-bit Linux system is recommended. This should help the server manage more memory and run smoother when lots of players connect at the same time. We think 64-bit is safer than 32-bit for this kind of application, but we’ll have to confirm during testing. A 64-bit Linux system makes sense here. It can handle more memory, which is key if a ton of players are on at once. In a gaming environment, smoother memory handling equates to less lag, which is a huge plus in gaming environments (Silberschatz et al., 2023).
3. **Storage Management**: A reliable database like PostgreSQL seems like a good fit for storing player info, team data, game history, and puzzles. It’s important that the database is easy to access from multiple devices at once. One could also consider MySQL depending on performance testing. Cloud storage could be considered too if we need scalability, but the cost comparison (AWS vs. Azure vs. Google, 2025) shows Linux-hosted solutions are super cost-effective for small to medium traffic.
4. **Memory Management**: Java’s JVM comes with garbage collection, which is a lifesaver for memory management (Silberschatz et al., 2023). Memory management is handled by the JVM, which does garbage collection automatically. Hopefully this keeps everything running smoothly, though we might run into problems if too many players are active at once.
5. **Distributed Systems and Networks**: Since players will be on different devices, the game should use a client-server model. REST APIs or WebSockets can handle real-time updates, so everyone sees drawings and guesses immediately. Microservices architecture could also help if we ever want to scale features separately from the main game (Microservices vs. SOA, 2025).
6. **Security**: Players’ accounts and game data need protection. Secure logins, HTTPS traffic, and proper database permissions are a must. Linux supports all of this, and the right setup can make sure nobody hijacks a game session or leaks team info. Think of it as the firewall between the gaming rig and any sketchy Wi-Fi network that it could connect to (What Is Serverless Architecture? What Are its Pros and Cons?, 2025; Silberschatz et al., 2023).

**References**

Cloud Storage Cost Comparison: AWS vs. Azure vs. Google. (2025). *TechTarget*. https://www.techtarget.com/searchcloudcomputing/feature/Cloud-storage-cost-comparison-AWS-vs-Azure-vs-Google

Microservices vs. SOA — Is There Any Difference at All? (2025). *TechTarget*. https://www.techtarget.com/searchapparchitecture/feature/Microservices-vs-SOA-Is-there-any-difference-at-all

Silberschatz, A., Galvin, P. B., & Gagne, G. (2023). *Operating system concepts* (10th ed.). Wiley.

What Is a Linux Server and Why Does Your Business Need One? (2025). *TechTarget*. https://www.techtarget.com/searchdatacenter/definition/Linux-server

What Is Serverless Architecture? What Are Its Pros and Cons? (2025). *TechTarget*. https://www.techtarget.com/searchcloudcomputing/definition/serverless