

# Win, Lose or Draw

# **CS 230 Project Software Design**

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 |
| --- | --- | --- | --- |
| 3.0 | 09/30/2023  10/14/2023 | Cruz, Matthew | * 9/30/2023 * Operating systems explained and compared * 10/24023 * Recommended a cross-platform cloud-based approach for "Draw It or Lose It" to ensure compatibility across various computing environments. * Advised the use of cloud-based storage solutions for flexible data management. * Explained the memory management approach for web-based applications. * Suggested the utilization of APIs and web services for inter-platform communication. * Emphasized security measures, including encryption and regular updates, to protect user information. |

## [Executive Summary](#_sbfa50wo7nsh)

Our client has presented us with a software design challenge that involves creating a system to efficiently manage games, teams, and players within their organization. To meet this challenge, we propose a comprehensive solution that addresses both business and technical requirements.

On the business side, the client requires a streamlined system with a user-friendly interface for managing game-related data. Data security is a top priority, and the software must prevent unauthorized access. Additionally, the client anticipates future growth, making scalability and performance essential.

Technically, we will implement the system using object-oriented principles, ensuring maintainability and extensibility. The Singleton pattern will be applied to the GameService class to provide centralized control. The software will support one-to-many relationships between entities and include robust data validation and error handling. Finally, an authentication and authorization system will be integrated to protect data and functionalities. Our proposed solution, which employs UML class diagrams for visualization, aligns perfectly with the client's objectives, setting the stage for an efficient and secure software system to manage games, teams, and players.

## 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)

Developing a web-based distributed game application comes with a set of crucial design constraints. Network latency and bandwidth limitations can impact real-time interactions, necessitating data compression and asynchronous communication. Scalability is vital, requiring load balancing and distributed databases to accommodate increasing user numbers. Security measures, including encryption and authentication, must be robust to protect user data and gameplay integrity. Ensuring compatibility across various platforms and browsers is challenging but necessary for a consistent user experience. Effective data synchronization, latency compensation, and data backup strategies are critical for multiplayer games. User authentication, third-party integrations, regulatory compliance, and cost management also play pivotal roles in delivering a successful and secure gaming application. Adhering to these constraints is essential to meet user expectations and thrive in the competitive gaming industry.

## [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 provided UML class diagram depicts a system for managing games, teams, and players. It comprises several classes, including Program Driver and Singleton Tester at the top level. Program Driver likely serves as the program's entry point, while Singleton Tester may be utilized for Singleton testing. The Entity class represents generic entities with attributes such as 'id' and 'name.' Below Entity, there are GameService, Game, Team, and Player classes. GameService acts as a central manager, tracking games, players, and teams, with a Singleton pattern to ensure a single instance. Relationships are established between GameService and Game (one-to-many), Game and Team (one-to-many), and Team and Player (one-to-many). While the open arrows from Game, Team, and Player to Entity suggest inheritance or usage relationships, these are not explicitly defined. Object-oriented principles like encapsulation (private attributes, public getters), Singleton pattern, association, aggregation (composition), and possible inheritance are evident in the diagram, facilitating efficient software development for managing games, teams, and players.

**"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 servers are characterized by their reliability and Unix-based foundation, making them suitable for hosting web-based software applications. Their advantages include user friendly, strong security features, and integration with Apple's ecosystem. However, they may have limitations in terms of scalability and software compatibility, and they are less prevalent in enterprise settings compared to Linux and Windows servers. | Linux is characterized by its stability, security, and versatility, making it an ideal choice for hosting web-based software. It's cost effective, highly scalable, and suitable for various environments. However, it may require a learning curve for some users, and compatibility issues can arise in specific cases. | Windows servers offer a user-friendly environment with extensive software support and integration with Microsoft technologies. They are suitable for client’s needing compatibility with Windows-based applications but come with higher licensing costs and potential stability concerns. | Mobile devices as servers provide portability and accessibility for remote management. They excel in scenarios requiring real-time notifications but have limited hardware resources and may not offer the same reliability and scalability as traditional servers. Security considerations also apply. |
| **Client Side** | Supporting Mac clients involves considerations for development cost, slight time increases due to Mac-specific testing, and the need for Mac development expertise. The investment may be higher, but it's essential for reaching a significant Mac user base. | For Linux clients, cost is usually lower thanks to open-source tools, but development time can vary due to distribution diversity. Linux development expertise is crucial to address these variations and ensure compatibility. | Windows client support offers a broad user base with manageable development considerations. Costs are moderate, development time is relatively shorter, and expertise in Windows development is essential for optimization. | Supporting mobile clients can be moderately costly, with longer development times due to platform-specific testing and optimization. Proficiency in mobile app development and adherence to platform-specific design guidelines are vital for success. |
| **Development Tools** | For software development on Mac, relevant programming languages often include Swift and Objective-C, with Xcode being the primary integrated development environment (IDE). Xcode offers a comprehensive suite of tools, including a code editor, interface builder, and performance analysis instruments. Additionally, developers may use third-party tools like JetBrains' AppCode for alternative development experiences. | Linux software development supports a wide array of programming languages, including C, C++, Python, Java, and more. Developers often use popular integrated development environments (IDEs) such as Visual Studio Code, Eclipse, or JetBrains' IntelliJ IDEA, tailored to specific language preferences. Compilers, build tools like Make, and debugging utilities are commonly employed to build software for deployment on Linux. | Building software for Windows involves languages like C++, C#, and Python. Microsoft's Visual Studio is the primary IDE, offering a comprehensive suite for Windows application development. Visual Studio Code is a popular choice for cross-platform development. Additionally, the Windows Software Development Kit (SDK) and various libraries aid in creating Windows applications. | Mobile app development requires platform-specific languages and tools. For iOS (Apple), Swift and Objective-C are used with Xcode as the IDE. On Android devices, Java and Kotlin are employed with Android Studio as the preferred IDE. Cross-platform development tools like Flutter (Dart language) and React Native (JavaScript) allow developers to build apps for both iOS and Android simultaneously, streamlining development across mobile platforms. |

## 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**: To expand "Draw It or Lose It" to other computing environments and ensure cross-platform compatibility, I recommend the use of a cloud-based server platform. Specifically, I suggest the utilization of a cloud infrastructure provider such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP). These platforms offer scalability, reliability, and support for various operating systems. The cross-platform compatibility and global reach of these platforms help ensure a seamless gaming experience for users on various operating systems and devices.
2. **Operating Systems Architectures**: A web-based application operates on a client-server architecture. Clients on different operating systems are connected to a centralized server hosting the application. This architecture facilitates compatibility across diverse platforms, ensuring a consistent user experience. It retains all the benefits of the client-server model while leveraging the scalability, reliability, and cost-efficiency of cloud infrastructure. This architecture is well-suited for ensuring cross-platform compatibility, consistent user experience, and efficient management of online games in the cloud.
3. **Storage Management**: A web-based application relies on the client's web browser and the server's resources. The client browser manages memory usage for the web application interface, while the server efficiently handles data processing and memory allocation. This architecture ensures optimal memory usage for "Draw It or Lose It" across different platforms.
4. **Memory Management**: A web-based application relies on the client's web browser and the server's resources. The client browser manages memory usage for the web application interface, while the server efficiently handles data processing and memory allocation. This architecture ensures optimal memory usage for "Draw It or Lose It" across different platforms.
5. **Distributed Systems and Networks**: Achieving communication between various platforms can be accomplished through APIs and web services. The client-side components can communicate with a centralized server via RESTful APIs or WebSocket connections. Dependencies within distributed systems and networks are managed by robust error-handling mechanisms to address connectivity issues, ensuring uninterrupted gameplay and synchronization of data.
6. **Security**: Security is a top priority. A web-based application can leverage security features inherent in modern web frameworks, including data encryption, user authentication, and access controls. Cross-platform user protection is achieved through secure authentication systems. Regular security audits and updates are essential to address evolving threats and vulnerabilities.