

Draw It or Lose it

# **CS 230 Project Software Design Template**

Version 1.3

## 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 |
| --- | --- | --- | --- |
| 1.3 | 2/16/25 | Victor Ngetich | Updating Recommendations |

**[Executive Summary](#_sbfa50wo7nsh)**

The Gaming Room has enlisted CTS, represented by me, to develop a web-based version of their existing Android-exclusive game, “**Draw It or Lose It”**. This new version will be designed to run across multiple platforms. The document outlines the business and operational constraints affecting product delivery, along with solutions to ensure the widest possible audience reach, thereby expanding the company’s market presence.

**Key Objectives**

1. **Develop a Web-Based Game**: Transition the existing Android-only game to a web-based platform, ensuring compatibility with multiple operating systems and hardware environments.
2. **Address Development Environment Concerns**: Establish a reliable and efficient development environment that ensures seamless implementation, testing, and deployment.
3. **Ensure Smooth Performance**: Select and display images from an extensive library of stock drawings at a steady and reliable rate, maintaining consistent game performance.

**Technical and Application-Specific Requirements**

* **Multi-Platform Compatibility:** The game must function seamlessly across multiple platforms, including desktops and mobile devices (Iversen & Eierman, 2014; Smyth, 2021; Haibeh, Yagoub, & Jarray, 2022).
* **Team-Based Gameplay:** Each game session must support one or more teams, with multiple players per team.
* **Unique Identifiers:** Unique IDs and names must be assigned to all games, teams, and players. A name validation feature must be implemented to check for duplicate entries (Avnet Technology Solutions, 2012).
* **Single Instance Enforcement:** Only one instance of the game should be active in memory at any given time to ensure consistency and prevent duplication (Tanenbaum, Herder, & Bos, 2006; Stallings, 2007; Silberschatz, Galvin, & Gagne, 2000).
* **Cybersecurity Considerations:** The game must follow industry security standards to protect user data, incorporating best practices based on ISO/IEC 27001 and NIST controls (Angelo Edu, Alexis, & Lenis, 2023).

**Technical Implementation**

To achieve these goals, the following design patterns and technologies will be utilized:

**1.** **Singleton Pattern:** Ensures that only one instance of the game exists at any time, maintaining consistency and avoiding duplication (Tanenbaum, Herder, & Bos, 2006; Stallings, 2007).

**2.** **Iterator Pattern:** Allows efficient management of teams and players, ensuring smooth traversal and organization without conflicts.

**3.Cloud-Based Hosting:** Provides scalable infrastructure to handle high demand and ensure reliability. This will support rendering images at a steady rate and managing game data efficiently (Avnet Technology Solutions, 2012).

**4.Hardware Compatibility:** The game will be designed to run on a range of devices, utilizing optimized server architecture to minimize hardware constraints (Silberschatz, Galvin, & Gagne, 2000; Stallings, 2007; Tanenbaum, Herder, & Bos, 2006).

**5.Operating System Considerations:** Ensuring smooth integration with macOS (Krill, 2016), Android (Smyth, 2021; Iversen & Eierman, 2014), and Windows (Microsoft, 2021) will be a key aspect of the development process.

CTS is committed to delivering a scalable and reliable solution that meets The Gaming Room's needs. By addressing their concerns about platform compatibility, development setup, and performance, we will ensure the game functions seamlessly across multiple platforms, providing an engaging and consistent user experience.

## [Design Constraints](#_2et92p0)

**1. Business and Project Constraints**

**1.1 Project Schedule, Milestones, and Budget**

* Development phases will strictly adhere to the project schedule and remain within budget expectations.
* Change requests will be evaluated for their impact on budget, timeline, and development effort before approval.
* Source: Project Scope Documentation.

**1.2 Team Knowledge of Web-Based App Development and Client Domain**

* The development team must have expertise in web-based applications to ensure optimal user experience (UX), maintainability, and usability (Meiert, 2019).
* The project team should include skilled web developers, software engineers, and subject matter experts (SMEs) in the client’s domain.

**1.3 Software Licensing, Legal, and Compliance Considerations**

* No known software licensing requirements impact development.
* No legal, regulatory, or compliance constraints are currently identified.

**1.4 Vendor Solutions and Cloud Hosting**

* The game will be remotely hosted via a third-party cloud provider, eliminating the need for customized vendor integration unless necessary.
* Source: (Avnet Technology Solutions, 2012).

**2. Design Constraints**

**2.1 Application Architecture**

* The transition from a mobile-based architecture to a client-server model requires establishing three distinct environments: development, testing, and production.
* Each environment must be equipped with the necessary hardware and operating systems to support the application’s needs.
* The infrastructure must support scalable, distributed computing to handle concurrent users efficiently (Avnet Technology Solutions, 2012; Haibeh, Yagoub, & Jarray, 2022).
* The server infrastructure must ensure high availability, fault tolerance, and redundancy to minimize downtime.

**2.2 Storage and Memory Management**

* Transitioning to a client-server model increases storage and memory demands due to centralizing data storage.
* The server infrastructure must support high-speed data retrieval and caching mechanisms to maintain performance.
* Adequate storage capacity and RAM must be provisioned to support large-scale game data storage and retrieval (Silberschatz, Galvin, & Gagne, 2000).
* The server OS must efficiently manage multi-threading and concurrency to optimize resource utilization (Stallings, 2007; Tanenbaum, Herder, & Bos, 2006).

**2.3 Security and System Constraints**

* The transition to a client-server model introduces new security and infrastructure complexities.
* The operating system must support secure communication protocols to safeguard data transmission.
* The server OS must facilitate scalable session management to handle multiple users securely.
* The infrastructure must support access control mechanisms at the OS level to enforce security policies (Angelo Edu, Alexis, & Lenis, 2023).

By addressing these constraints application architecture, storage and memory management, and security and complexity the transition will ensure that “**Draw It or Lose It”** is scalable, efficient, and secure, providing a seamless and enjoyable gaming experience for users across all supported platforms.

## [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 design of “**Draw It or Lose It”** leverages Object-Oriented Programming (OOP) principles and design patterns to address the client’s application requirements and ensure scalability, reliability, and efficiency. The game must support multiple teams and players, assign unique identifiers and names to each game, team, and player, and ensure only one instance of the game exists in memory at any given time. These requirements are fulfilled using a combination of OOP principles such as inheritance, polymorphism, encapsulation, and abstraction, along with design patterns like the singleton and iterator.

The **Entity** class serves as the base class for **Game, Team,** and **Player**, providing shared attributes like **id** and **name** to ensure every object in the system has a unique identifier and name. This approach satisfies the requirement for uniqueness while promoting code reuse and consistency across the system. Polymorphism allows the system to interact with these objects uniformly, supporting scalability and simplifying future enhancements. Encapsulation ensures that object creation is controlled through the **GameService** class, preventing direct instantiation and maintaining the integrity of the system. Abstraction further simplifies the user interface by allowing users to interact with methods like **addGame(), addTeam(),** and **addPlayer()** without needing to understand the internal complexities of the object creation process.

The singleton pattern is implemented in the **GameService** class to ensure that only one instance of the game service exists in memory at any given time. This is achieved through a private constructor and a **getInstance()** method, which checks if an instance already exists and creates one if it does not. This pattern satisfies the requirement for a single game instance while centralizing the management of games, teams, and players. The iterator pattern is used within the **addGame(), addTeam(),** and **addPlayer()** methods to prevent the creation of duplicate objects and to ensure proper relationships between games, teams, and players. For example, the **addTeam()** method checks for duplicate team names before adding a new team to a game, and the **addPlayer()** method performs similar checks when adding players to teams.

Additional considerations include the system’s scalability and future enhancements, such as the potential for an administrative interface to manage users, teams, and games. This interface will leverage the same OOP principles and design patterns to maintain efficiency and consistency. While the **ProgramDriver** and **SingletonTester** classes are not part of the game’s core objects, they play a vital role in testing and verifying the functionality of the domain model during development. Together, these principles and patterns provide a robust, scalable, and efficient solution that fully satisfies the client’s requirements, ensuring a seamless gaming experience and reliable system architecture.

**"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 must 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, built on a stable Unix-based framework, is recognized for its reliability and security, though it prioritizes Mac integration and restricts third-party compatibility (Tanenbaum, Herder, & Bos, 2006). While macOS supports virtualization through Parallels and VMware, it lacks the flexibility of Linux for server environments. Its LDAP support is limited, and it is not a common choice for cloud hosting. | Linux is widely used for hosting, offering high uptime and strong security, though it has a steep learning curve (Stallings, 2007). It is an open-source, POSIX-compliant operating system extensively utilized in cloud and server environments. Linux excels in lightweight containerization (Docker, Kubernetes) and provides robust support for LDAP authentication. | Windows offers moderate performance and integrates with Linux but lacks the stability of Linux for hosting (Silberschatz, Galvin, & Gagne, 2000). It supports Active Directory and group policies but is more susceptible to malware attacks. While Windows allows virtualization and can host Linux instances via WSL, it is generally less efficient than native Linux servers. | Mobile devices require minimal space but have limited resources for hosting (Iversen & Eierman, 2014). They are not designed for server use due to hardware limitations and the lack of enterprise-grade security and access controls (Smyth, 2021). |
| **Client Side** | macOS relies on Safari with limited third-party integration. It offers a stable ecosystem for Apple applications but is restrictive for cross-platform compatibility. (Krill, 2016). | Development in Linux is straightforward since commonly used programming languages like Java, C/C++, and Python are well-supported. Moreover, the GNU/Linux platform offers robust multi-user support (Stallings, 2007). While Linux is less common for consumer desktop use, it is widely adopted in enterprise, server, and cloud environments (Tanenbaum, Herder, & Bos, 2006). | Windows is widely used, offering strong third-party integration and a user-friendly interface. It supports various browsers and development tools, making it an ideal choice for diverse applications (Silberschatz, Galvin, & Gagne, 2000). | Mobile devices are cost-effective and intuitive but have limited processing power. They dominate user engagement but are constrained by hardware capabilities for high-end computing tasks (Iversen & Eierman, 2014; Smyth, 2021). |
| **Development Tools** | Swift and Objective-C are the primary languages for iOS development. Essential tools include Xcode, CocoaPods, and Homebrew. macOS is required for iOS development as it mandates Xcode and a macOS-based development environment (Krill, 2016). | Python, C++, Java, and JavaScript are widely used for Linux development. Common tools include GCC, Clang, Vim, Emacs, Docker, and Kubernetes. Linux is preferred for backend development, cloud applications, and DevOps tools due to its flexibility and stability (Stallings, 2007; Tanenbaum, Herder, & Bos, 2006). | C#, JavaScript, and Python are commonly used for Windows development. Key tools include Visual Studio, .NET Framework, PowerShell, and Windows Subsystem for Linux (WSL). Windows supports a wide range of development tools, making it ideal for enterprise and .NET applications (Silberschatz, Galvin, & Gagne, 2000). | Kotlin (Android), Swift (iOS), and Dart (Flutter) are commonly used for mobile development. Key tools include Android Studio, Xcode, React Native, and Flutter. Mobile development is typically done on macOS, Windows, or Linux environments rather than directly on mobile devices (Iversen & Eierman, 2014; Smyth, 2021). |

**Recommendations**

**1.Operating Platform** Windows OS serves as an appropriate server platform for expanding "Draw It or Lose It" due to its extensive user base and developer support (Microsoft, 2021). Its widespread adoption provides access to a large pool of resources and expertise, making it a practical choice for game development. Additionally, Windows OS supports cross-platform game development frameworks such as Unity and Unreal Engine, facilitating the expansion of the game to various computing environments. Furthermore, its compatibility with third-party software and tools enhances development efficiency and deployment capabilities.

**2.Operating System Architectures** Windows OS consists of several integral components, including the kernel, hardware abstraction layer (HAL), executive, and drivers (Silberschatz, Galvin, & Gagne, 2000). These components collectively manage system resources, ensure hardware compatibility, and optimize system performance. The modular nature of Windows OS architecture allows efficient process management and seamless interaction between hardware and software. The application of this architecture in server management and enterprise-level gaming platforms ensures resource allocation optimization, which is critical for performance-driven applications like online gaming.

**3.Storage Management** A cloud-based storage management system such as Amazon Web Services (AWS) is recommended for supporting "Draw It or Lose It" due to its scalability and reliability (Avnet Technology Solutions, 2012). AWS provides a variety of storage solutions, including object storage, block storage, and database services, ensuring high availability and redundancy for game assets and user data. Cloud-based storage enhances security through encryption, access control mechanisms, and integrated backup and disaster recovery solutions, ensuring data integrity and continuous accessibility while minimizing performance lags.

**4.Memory Management** Windows OS employs sophisticated memory management techniques that enhance game performance. These include large memory support, copy-on-write functionality, memory-mapped files, and integration with the cache manager (Tanenbaum, Herder, & Bos, 2006). The dynamic allocation and deallocation of memory minimize latency, improving the gaming experience. Caching mechanisms reduce load times, while pre-fetching strategies optimize data retrieval, ensuring real-time gaming efficiency. Such optimizations are crucial for ensuring seamless gameplay, reducing delays, and enhancing overall system responsiveness.

**5.Distributed Systems and Networks** To enable "Draw It or Lose It" to function across multiple platforms, a distributed system architecture should be employed. This involves leveraging multiple servers and dynamically routing network packets based on demand (Stallings, 2007). Load balancing techniques distribute traffic evenly, mitigating congestion and optimizing resource utilization. Edge computing enhances responsiveness and reduces latency, critical for real-time multiplayer interactions (Lina A. Haibeh, Yagoub, & Jarray, 2022). Ensuring robust connectivity and failover mechanisms will maintain game stability during network outages, improving the overall user experience.

**6.Security** is a paramount concern for "Draw It or Lose It." Windows OS provides comprehensive security features such as user authentication, role-based access control (RBAC), and encryption protocols to safeguard sensitive game data (Angelo Edu, Alexis, & Lenis, 2023). Additionally, intrusion detection systems (IDS), multi-factor authentication (MFA), and AI-driven threat detection mechanisms further strengthen security against cyber threats. Secure data transmission protocols, such as TLS/SSL, ensure encrypted communication between client and server, protecting user information across platforms and mitigating potential security breaches.

**Conclusion:** Based on the analysis of system architecture and operational requirements, Windows OS emerges as a highly suitable platform for The Gaming Room. With robust security measures, efficient storage and memory management, and scalable networking capabilities, Windows provides a comprehensive solution for game development, deployment, and maintenance. Futureproofing through cloud integration, distributed networking, and advanced cybersecurity implementations ensure long-term sustainability for the gaming platform.

**References**

Angelo Edu, M. L., Alexis, G. P., & Lenis, W. P. (2023). Cybersecurity framework for SMEs in Peru based on ISO/IEC 27001 and CSF NIST controls. 2023 18th Iberian Conference on Information Systems and Technologies (CISTI), Information Systems and Technologies (CISTI), 2023 18th Iberian Conference On, 1–7.

Applied operating system concepts: First edition. By Avi Silberschatz, Peter Galvin and Greg Gagne. John Wiley & Sons, New York. (2000). Computers and Mathematics with Applications, 40(2), 420

Avnet Technology Solutions Introduces Cloud Computing Offerings Powered by Amazon Web Services. (2012).

Fontana, J. (2009, August 19). Red Hat, IBM, Novell major contributors to exploding Linux kernel development; Linus Torvalds drops in Linux kernel changes ranking, but still key contributor overall. Network World.

Hochmuth, P. (2006, June 12). Linux server uptime is worse than Windows; Unix, says Yankee Group study; \* Yankee Group blames outages on lack of Linux, open- source documentation. *Network World*.

Iversen, J., & Eierman, M. (2014). Learning mobile app development: A hands-on guide to building apps with iOS and Android. Pearson Education.

Krill, P. (2016, June 15). Apple’s Swift 3 developer preview arrives for MacOS and Linux. InfoWorld.Com.

Lía Hebe Molinari. (2007). Operating systems: Internals and design principles, 5th ed. William Stallings. Journal of Computer Science and Technology, 7(03), 267.

Lina A. Haibeh, Mustapha C. E. Yagoub, & Abdallah Jarray. (2022). A Survey on Mobile Edge Computing Infrastructure: Design, Resource Management, and Optimization Approaches. IEEE Access, 10, 27591–27610.

Neil Smyth. (2021). Android Studio 4.2 Development Essentials - Kotlin Edition: Developing Android Applications Using Android Studio 4.2, Kotlin, and Android Jetpack. Packt Publishing.

Tanenbaum, A. S., Herder, J. N., & Bos, H. (2006). Can we make operating systems reliable and secure? Computer, 39(5), 44–51. https://doi.org/10.1109/MC.2006.156