

The Gaming Room

# **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 |
| --- | --- | --- | --- |
| 1.2 | 11/16/2024 | Alex Chadburn | Revamped recommendations and filled out sys architecture |

## [Executive Summary](#_sbfa50wo7nsh)

Creative Technology Solutions (CTS) has been engaged by The Gaming Room to develop a web-based version of their existing Android game, Draw It or Lose It. This game, inspired by the 1980s television show Win, Lose or Draw, involves teams guessing images rendered from a library of stock drawings. The new web-based game will support multiple platforms and enhance accessibility. This document outlines the software design, addressing the client's requirements for scalability, uniqueness of game and team names, and ensuring only one instance of the game exists in memory at any given time.

## Requirements

The Gaming Room has specified several key requirements for the game application:

* The game must support multiple teams, each with multiple players.
* Game and team names must be unique.
* Only one instance of the game should exist in memory at any time, which will be managed through unique identifiers for each game, team, and player.
* The game must be a web-based application to support multiple platforms.

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based distributed environment involves several design constraints:

* **Compatibility**: The application must be compatible with various web browsers and devices, ensuring consistent user experience across platforms.
* **Performance Optimization**: The design must optimize response times and resource utilization, ensuring a smooth and responsive user experience even under high load conditions.
* **Secure data**: Ensure that sensitive data is secured through encryption. This includes but is not limited to passwords, emails, and 2fa.
* **User friendly**: The game must be easy to use for end-users and be engaging.

These constraints impact the development process by necessitating the use of specific design patterns and careful planning to ensure the system meets performance, security, and usability standards.

## [System Architecture View](#_ilbxbyevv6b6)

Different operating systems have different architectures, the Windows operating system uses hybrid kernel architecture, which combines elements of both monolithic and microkernel designs. This hybrid approach integrates the performance benefits of a monolithic kernel, where core services like process and memory management run in kernel mode, with the modularity of a microkernel, where some subsystems operate in user mode. This design allows Windows to maintain high performance and efficient communication between services while enhancing stability and security by isolating certain services in user mode. The hybrid kernel's modularity also allows for easier updates and replacements of components without affecting the entire system, balancing performance with reliability.

The Linux kernel, on the other hand, follows a monolithic architecture. In this structure, the entire operating system, including core services such as process management, memory management, file systems, and device drivers, is implemented as a single large process running in kernel mode. This design provides high performance and efficient communication between services because all components can interact directly within the kernel space. However, it also means that a bug in any part of the kernel can potentially crash the entire system. Despite this, Linux maintains stability and security through rigorous testing and a large, active development community.

File systems are essential for organizing and managing data on storage devices, consisting of several key components. Files are the basic units of data storage, containing information such as text, images, or videos. Directories, or folders, organize these files into a hierarchical structure, making data management more efficient. File metadata provides crucial information about each file, including its name, size, and access permissions. File system operations, such as creating, moving, and deleting files, are fundamental for data management. Various file allocation methods, like contiguous, linked, and indexed allocation, determine how space is allocated on storage devices. Security and access control mechanisms ensure that only authorized users can access or modify data. Together, these components form the backbone of a file system, enabling efficient storage, retrieval, and management of data.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram for the game application includes ProgramDriver, GameService, Game, Team, Player, Entity, and SingletonTester. The Entity class serves as a base class for Game, Team, and Player, promoting code reuse through inheritance. GameService is a singleton class that manages game instances, ensuring only one instance exists and using the iterator pattern to handle collections of games and teams. ProgramDriver initializes the application, while SingletonTester verifies the singleton behavior. These classes demonstrate key OOP principles such as encapsulation, inheritance, and the use of design patterns like Singleton and Iterator, ensuring a modular, maintainable, and scalable design.

**"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** | Macs have a high initial cost and are not commonly used for server-side hosting. While macOS Server is available, it is less common for large-scale web applications. | Linux is a popular choice for server-side development due to its stability, security, and flexibility. The primary cost for hosting a server is the cost of the machine, as there are no licensing costs for popular distributions such as Ubuntu. | Comes with  licensing costs  that can range in  price. It can host  web-based  applications and  scale up to  thousands of  players. | Mobile devices typically do not serve as servers but can connect to server-side applications. Licensing costs for mobile development tools vary depending on the specific provider. |
| **Client Side** | Macs are popular among developers for client-side development due to their robust performance, high-quality displays, and Unix-based OS, which supports various development tools. They are particularly favored for iOS development. | Linux is less common for client-side development but offers a powerful environment for developers who prefer open-source tools. It supports a wide range of development environments and is highly customizable. | Windows is widely used for client-side development, offering extensive support for various development tools and environments. It is particularly strong in enterprise settings and for developing Windows applications. | Mobile devices are the primary target for client-side development in mobile app development. Ensuring compatibility across different devices and operating systems (iOS, Android) is crucial. |
| **Development Tools** | Mac supports a wide range of development tools, including Xcode for iOS development, and is compatible with many cross-platform tools like Visual Studio Code, IntelliJ IDEA, and more. | Linux supports a vast array of open-source development tools and environments, such as Eclipse, Visual Studio Code, and various command-line tools such as VIM. It is highly customizable to fit specific development needs. | Windows offers extensive support for development tools, including Visual Studio, It also supports a wide range of other tools like VSCode, Eclipse, and more. | Mobile development tools include Android Studio for Android development and Xcode for iOS development. Cross-platform tools like Flutter and React Native are also popular for developing mobile applications. |

## Recommendations

1. **Operating Platform**: Linux is a versatile and widely used operating system known for its stability, security, and flexibility. It is an open-source platform, which means it is free to use and can be customized to meet specific needs. Linux is popular in both server and desktop environments, making it a strong choice for a variety of applications. Its robust performance makes it an ideal platform for developers and businesses alike. Additionally, Linux supports a wide range of hardware and software, providing a flexible and scalable solution for different use cases. Whether for server-side hosting or client-side development, Linux offers a reliable and cost-effective platform.
2. **Operating Systems Architectures**: The architecture of the Linux operating system is a layered structure that includes several key components. At the base is the hardware layer, which consists of physical components such as the CPU, RAM, and input/output devices. Above this is the kernel, the core of the operating system, responsible for managing hardware, system calls, memory, and processes. Linux uses a monolithic kernel, which means that all the core functions of the operating system run in a single large block of code in a single address space. The shell, which can be either a command-line interface (CLI) or a graphical user interface (GUI), sits above the kernel and provides an interface for users to interact with the system. Finally, the application layer includes user-level programs such as web browsers, media players, and office suites that run on top of the operating system.

**Storage Management**: Effective storage management involves a combination of cloud storage solutions and database optimization. Cloud storage services like Amazon S3 offer scalability and accessibility, allowing users to access their data from multiple devices. Amazon S3 is reliable, has low latency, and is cost-effective for hosting images. Using data compression can help reduce the storage footprint, and regular maintenance, like cleaning up unused files and optimizing database performance, will keep storage management in check.

1. **Memory Management**: Linux memory management includes several key features designed to optimize the use of system memory. Virtual memory allows the system to use more memory than is physically available by using disk space as an extension of RAM. This is complemented by demand paging, which loads pages into memory only when they are needed, reducing the overall memory footprint. Memory allocation in Linux is managed for both kernel structures and user space programs, ensuring efficient use of resources. The page cache stores frequently accessed disk data in RAM to improve performance, while swapping moves inactive pages from RAM to disk to free up memory for active processes. These mechanisms work together to ensure that the system runs smoothly and efficiently, even under heavy load.
2. **Functions of memory and storage management**: Memory and storage management are crucial functions in computer systems, ensuring efficient use of resources and optimal performance. Without sufficient local storage, we would need to rely on alternatives that could potentially stall and make the game unplayable. Similarly, if we do not effectively utilize RAM, caching, and other memory management techniques, we would be wasting resources by repeatedly requesting data from the cloud that is already stored locally. Efficient memory and storage management help prevent these issues, ensuring smooth and reliable system performance.
3. **Memory management techniques**: These are essential for optimizing the use of system memory and ensuring efficient performance. Key techniques include **single contiguous allocation**, where all memory except a small portion reserved for the OS is available for one application, and **partitioned allocation**, which divides memory into contiguous partitions for different tasks. **Paged memory management** splits memory into fixed-size units called pages, while **segmented memory management** divides memory into segments that can vary in size. **Swapping** temporarily moves processes between main memory and secondary storage to free up memory for active processes. These techniques help manage memory allocation, reduce fragmentation, and ensure that processes run smoothly and efficiently
4. **Distributed Systems and Networks**: Linux supports distributed systems and networks through a variety of features and protocols. Network File Systems (NFS) allow files to be shared across a network as if they were on a local disk, facilitating easy access to data across multiple machines. For distributed computing, Linux supports frameworks like Hadoop and Spark, which enable the processing of large datasets across clusters of computers. Communication protocols such as TCP/IP ensure reliable communication between distributed systems, while load balancing distributes workloads across multiple servers to ensure efficient resource use and high availability. These capabilities make Linux a robust platform for building and managing distributed systems.
5. **Security**: Security is paramount, and employing strong encryption protocols, robust authentication mechanisms, regular updates, and secure coding practices will protect user information across platforms. Encryption protocols like SSL/TLS protect data in transit, while multi-factor authentication secures user accounts. Regular updates and patching protect against known vulnerabilities, and secure coding practices prevent common vulnerabilities like SQL injection and XSS. These measures ensure that user data is protected both on and between various platforms.