

“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 4**](#_Toc115077322)

[**System Architecture View 4**](#_Toc115077323)

[**Domain Model 5**](#_Toc115077324)

[**Evaluation 6**](#_Toc115077325)

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

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/13/2023 | Cory Miglis | Modified cover page, executive summary, requirements, design constraints, domain model. |
| 1.1 | 09/28/2023 | Cory Miglis | Added evaluation table. |
| 1.2 | 10/09/2023 | Cory Miglis | Added Recommendations section. |

**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 has engaged Creative Technology Solutions (CTS) to assist in the development of a web-based version of their popular game "Draw It or Lose It." The current Android app version has garnered success, and the client wishes to expand its reach to multiple platforms. The objective is to create a web-based game that allows multiple teams with multiple players to participate. This game involves guessing phrases based on stock drawings and is played in rounds, each lasting one minute. In case of non-guessing, other teams are given a chance to solve the puzzle within a limited timeframe. To address these requirements, CTS will design and develop a robust web-based gaming application. The key software requirements include support for multiple teams and players, unique game and team names, and ensuring only one game instance exists at any given time. This software design document outlines the strategies and constraints for achieving these goals.

## Requirements

In this section, we will identify and summarize the client's business and technical requirements for the web-based version of the "Draw It or Lose It" game application. These requirements serve as the foundation for the design and development process.

Multi-Team and Multi-Player Support:

Business Requirement: The game application must support multiple teams, each comprising multiple players. This supports the collaborative and competitive nature of the game.

Technical Requirement: The system should have the capability to create and manage multiple teams with dynamic numbers of players.

Unique Game and Team Names:

Business Requirement: To prevent confusion and ensure a seamless user experience, game and team names must be unique.

Technical Requirement: The system should enforce uniqueness constraints for both game and team names, providing real-time checks during the creation process.

Single Game Instance:

Business Requirement: Only one instance of the game should exist in memory at any given time to maintain a fair and organized gaming environment.

Technical Requirement: The system should implement mechanisms to ensure that only a single instance of a game can be created and active at any point, even when accessed concurrently by multiple users.

These requirements align with the client's vision for expanding "Draw It or Lose It" to a web-based platform and set the foundation for the software design and development process. By addressing these requirements, we aim to deliver a web-based gaming application that meets the client's expectations and enhances user engagement.

## [Design Constraints](#_2et92p0)

Cross-Platform Compatibility: To cater to a diverse user base, the application must run seamlessly on various operating systems, including Windows, Mac, Linux, and mobile devices. This requires careful consideration of platform-specific nuances and compatibility challenges.

Scalability: As the game aims to accommodate multiple teams and players concurrently, scalability is crucial. The architecture must be designed to handle a potentially large number of simultaneous users, ensuring optimal performance and responsiveness.

Network Connectivity: Given the multiplayer nature of the game, reliable network connectivity is essential. The application must handle potential network interruptions gracefully, preventing data loss and maintaining game state integrity.

Security: Protecting user information and ensuring fair play are paramount. The application must implement robust security measures to safeguard user data, prevent cheating, and ensure a secure gaming environment.

User Interface: Providing an intuitive and engaging user interface across various platforms is vital for user satisfaction. The design must adapt to different screen sizes and input methods while maintaining a consistent and enjoyable user experience.

Resource Management: Efficient utilization of server resources, such as memory and processing power, is crucial to deliver a smooth gaming experience. Effective memory management strategies must be implemented to optimize the application's performance.

Concurrency Control: To prevent conflicts and ensure fair gameplay, concurrent access to game resources by multiple teams and players must be carefully managed. Proper synchronization mechanisms are essential to avoid data corruption and maintain game integrity.

These design constraints will influence the architecture, technology stack, and development approach chosen for the web-based "Draw It or Lose It" game. By addressing these constraints effectively, CTS aims to deliver a high-quality, cross-platform gaming experience that aligns with the client's vision and requirements.

## [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 represents the domain model for the game application. It consists of several classes:

ProgramDriver Class: This class appears to serve as the entry point of the application, containing a main() method. It interacts with other classes, as indicated by the <<uses>> relationship with the SingletonTester class. However, it is not directly connected to other classes in the diagram.

SingletonTester Class: The SingletonTester class includes a testSingleton() method. It is also not directly connected to other classes in the diagram, but it is used by the ProgramDriver class, as indicated by the <<uses>> relationship.

Entity Class: The Entity class is a base class that holds common attributes and behaviors for other classes in the diagram. It includes attributes id (of type long) and name (of type String), as well as a constructor and getter methods for these attributes.

GameService Class: The GameService class represents a service for managing games. It contains attributes games (a list of Game objects) and nextGameId, nextPlayerId, and nextTeamId to manage unique identifiers. It follows the Singleton design pattern, as it includes a getInstance() method to ensure only one instance exists. It has methods for adding and retrieving games and managing player and team IDs.

Game Class: The Game class represents a game instance. It includes a list of Team objects (teams) and methods for adding teams and generating a string representation of the game.

Team Class: The Team class represents a team within a game. It includes a list of Player objects (players) and methods for adding players and generating a string representation.

Player Class: The Player class represents a player. It has attributes id and name, a constructor, and a method to generate a string representation.

Relationships and Object-Oriented Principles:

Inheritance: The Entity class serves as a base class, demonstrating inheritance. Other classes, such as Game, Team, and Player, inherit common attributes and behaviors (id and name) from the Entity class. This promotes code reusability and ensures consistency in attribute handling.

Composition: The UML diagram illustrates a composition relationship between GameService, Game, Team, and Player classes. GameService manages instances of Game, Game contains Team instances, and Team contains Player instances. This composition reflects the hierarchical structure of the game application.

Singleton Pattern: The GameService class implements the Singleton pattern by providing a single point of access (getInstance()) to its instance. This ensures that there is only one GameService instance throughout the application's lifecycle.

Association: Although not explicitly stated in the UML diagram, we can infer associations between classes, such as GameService associating with Game instances and Game associating with Team instances.

Aggregation: Aggregation relationships exist between Game and Team, as well as Team and Player, represented by the "0...\*" multiplicity. A Game can have zero or more Teams, and a Team can have zero or more Players.

This domain model leverages object-oriented programming principles to efficiently represent the game application's structure. Inheritance and composition promote code reuse and maintainability, while the Singleton pattern ensures proper management of the GameService instance. Associations and aggregations reflect the relationships between key entities in the game application, facilitating data management and gameplay logic.

**"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 is known for its user-friendly interface and is based on UNIX, making it a stable and secure choice for hosting web-based software applications. Mac hardware can be expensive, and it has less server-specific software compared to Linux and Windows. | Linux is known for its stability, security, and flexibility. It is highly customizable, offers a wide range of server software options, and is cost effective. It might have a steeper learning curve and not all software or applications are readily available for it. | Windows is known for its ease of use and compatibility with various software applications. The environment is familiar, and it supports a wide range of web technologies. Licensing costs can be high, and windows servers require more frequent updates and security patches compared to Linux. | Mobile devices have limited processing power, screen size, and input capabilities. Limited resources on mobile devices can pose challenges for hosting complex web applications, and optimizing user interfaces can be challenging. |
| **Client Side** | Consideration for supporting multiple types of clients on Mac involve potential development costs associated with macOS-specific software, the time needed to ensure cross-platform compatibility, and the expertise required to develop and maintain applications for Mac users. | Supporting multiple types of clients on Linux requires consideration of development costs related to open-source tools and libraries, the time needed to ensure compatibility across various Linux distributions, and the expertise needed to work with Linux-based systems and software. | When supporting multiple types of clients on Windows, developers need to factor in licensing costs for Windows-specific development tools, the time required to address compatibility issues across different Windows versions, and the expertise needed to navigate the Windows development environment. | Supporting multiple types of clients on mobile devices involves considering development costs associated with different mobile platforms (iOS and Android), the time required to create and maintain separate codebases for each platform, and the expertise needed to design and develop mobile-friendly user interfaces and experiences. |
| **Development Tools** | Relevant programming languages for software deployment on Mac include Swift and Objective-C, with Xcode as the primary Integrated Development Environment (IDE). Other tools may include Homebrew for package management and Git for version control. | To build software for deployment on Linux, developers commonly use programming languages such as C, C++, Python, and JavaScript. IDEs like Visual Studio Code, JetBrains IDEs (e.g., PyCharm), and tools like GCC and Make are commonly used in Linux development. | For software deployment on Windows, programming languages like C#, C++, and .NET are often used. IDEs such as Visual Studio and Visual Studio Code are popular choices, and tools like PowerShell and WiX can be employed for various tasks like scripting and installer creation. | To build software for deployment on mobile devices, developers typically use Swift and Objective-C for iOS development and Java or Kotlin for Android development. IDEs like Xcode for iOS and Android Studio for Android are the standard choices, with additional tools like Flutter for cross-platform development using Dart. |

## 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**: For The Gaming Room's Draw It or Lose It software architecture, we recommend a cross-platform development approach to expand its reach to different computing environments. Utilizing cross-platform frameworks like Unity or Unreal Engine allows the game to run seamlessly on Windows, macOS, Linux, iOS, Android, and more without the need for extensive platform-specific development.
2. **Operating Systems Architectures**: To facilitate communication between various platforms, a client-server architecture with protocols like TCP/IP or UDP can be implemented for real-time communication. When designing distributed systems and networks, it's crucial to address dependencies such as connectivity, outages, and cross-platform compatibility. Robust connection handling, error recovery mechanisms, and platform-independent data serialization formats (e.g., JSON) should be part of the architecture.
3. **Storage Management**: When considering storage management, opting for cloud-based solutions like Amazon S3, Google Cloud Storage, or Microsoft Azure Blob Storage is a wise choice. These services provide scalability and accessibility across platforms, ensuring that game assets and user data are easily accessible and synchronized regardless of the user's device.
4. **Memory Management**: Memory management specifics largely depend on the chosen game engine, with engines like Unity and Unreal Engine handling memory management internally to optimize resource allocation. Adhering to best practices for asset management within the selected engine ensures efficient memory utilization.
5. **Distributed Systems and Networks**: To enable communication between different platforms for Draw It or Lose It, implement a client-server architecture using standard networking protocols like TCP/IP or UDP. Ensure robust connection handling with automatic reconnection for network disruptions, and minimize latency. Handle errors gracefully, set up redundancy for outage resilience, use platform-agnostic data formats, and prioritize security with encryption and authentication mechanisms. This approach ensures seamless cross-platform communication while addressing connectivity and outage challenges.
6. **Security**: Security is paramount, and measures should be in place to protect user information and communication between platforms. This includes implementing HTTPS for secure data transmission, user authentication and authorization mechanisms, encryption for data storage, regular system updates to address security vulnerabilities, and leveraging built-in security features of the chosen operating platform, such as firewalls and access controls.