

<Name-of-Software-Application>

# **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 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 |
| --- | --- | --- | --- |
| 1.0 | <06/01/2024> | <Eskinder Kassahun> | <Changes where made to the contents > |
| 1.2 | <06/14/2024> | <Eskinder Kassahun> | <Worked on the Evaluation part of the template> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**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 hired Creative Technology Solutions (CTS) to create a web-based version of their Android game, "Draw It or Lose It." This game, inspired by the TV show "Win, Lose or Draw," involves teams guessing images from a drawing library. The game includes four one-minute rounds, with drawings completed by the 30-second mark. If the first team doesn't guess correctly, other teams have 15 seconds to make one guess. The project aims to support multiple platforms, ensure unique game and team names, and maintain a single game instance in memory. The system's design includes a base Entity class, with Game, Team, and Player classes inheriting from it, ensuring structured and maintainable code. The Game Service class uses the singleton pattern to manage games and ensure only one instance exists, with unique identifiers for games, teams, and players.

## Requirements

*The Gaming Room's requirements for the game application include:*

*Ability for one or more teams to be involved in a game.*

*Each team must have multiple players assigned.*

*Unique game and team names to prevent duplicates.*

*Only one instance of the game in memory at any time.*

*Robust validation mechanisms to maintain data integrity.*

*Efficient data handling and real-time updates for multiple simultaneous users.*

## [Design Constraints](#_2et92p0)

-Platform Compatibility: The game must work on desktops, tablets, and mobile devices. This means it needs to be responsive and tested on different devices and browsers.

-Concurrency and Performance: The game must handle many users and teams at the same time without slowing down. Efficient data handling and real-time updates are crucial.

Unique Identifiers: Each game, team, and player must have a unique ID to prevent duplicates and ensure data integrity.

-Single Game Instance: There should be only one instance of the game running at any time. Using the Singleton pattern will help manage this.

-Team Composition: Each team should have multiple players, enhancing collaboration and competition.

Unique Names: Game and team names must be unique, so users can check if a name is already in use when choosing a team name.

## [System Architecture View](#_ilbxbyevv6b6)

The Entity class forms the backbone of our system, with Game, Team, and Player classes inheriting from it. This means they all share common features from Entity. In UML diagrams, this is shown as inheritance. Each Game has multiple Teams, and each Team has multiple Players, which is called a HAS-A relationship. The GameService class manages multiple Game instances, ensuring only one instance of GameService exists at any time, following the Singleton pattern. This structured approach makes the system easy to understand and maintain.

## [Domain Model](#_8h2ehzxfam4o)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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.**

**The GameService class uses the Singleton pattern to restrict the instantiation of the class to one object. This is crucial for managing game state and ensuring that all operations are performed on a single instance of the service.**

**Composition:**

**The composition relationships between Game and Team, and Team and Player, illustrate the "has-a" relationship, where a game is composed of teams and a team is composed of players. This helps in organizing the hierarchy and managing the components efficiently.**

**By adhering to these principles, the design ensures that the software requirements are met efficiently, promoting scalability, maintainability, and robustness in the application.  
  
Domain Model**

**The UML class diagram for the game application includes the following classes:**

**Entity: A base class with common attributes (id and name).**

**Game: Inherits from Entity, represents a game instance.**

**Team: Inherits from Entity, represents a team.**

**Player: Inherits from Entity, represents a player.**

**GameService: Manages game operations, enforces Singleton pattern, and contains methods for adding and retrieving games, teams, and players.**

**Class Relationships:**

**Entity Class: Abstract base class with id and name attributes.**

**Game Class: Contains attributes and methods specific to a game instance.**

**Team Class: Contains a collection of Player objects and methods to manage team operations.**

**Player Class: Represents individual players, linked to specific teams.**

**GameService Class: Singleton class that manages the creation and retrieval of game instances, ensuring unique game, team, and player names using the Iterator pattern.**

**Object-Oriented Principles:**

**Inheritance: The Entity class serves as a base for Game, Team, and Player, promoting code reuse.**

**Singleton Pattern: Ensures only one instance of GameService exists, managing game state effectively.**

**Iterator Pattern: Facilitates iteration through collections of games, teams, and players, ensuring unique names and efficient data handling.**

## [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** | **Characteristics**: Mac includes built-in Apache web server commands and a robust suite of command line tools such as Python. Highly intuitive Mac client creation policies compared to Windows. **Advantages**: Seamless integration with macOS and other Apple products. Reliable performance and security. **Weaknesses**: Runs on limited hardware configurations and higher costs. **Deployment Method**: Limited options for server-based deployment. **Licensing Costs**: High due to proprietary nature. | **Characteristics**: Linux has a powerful command system and web server, with the benefit of being cost-effective. **Advantages**: Excellent performance, flexibility, and security. Stable and reliable for long periods. **Weaknesses**: Requires advanced expertise and has a steeper learning curve. **Deployment Method**: Excellent support with numerous distributions (e.g., Ubuntu Server, CentOS). **Licensing Costs**: None for the OS; potential costs for enterprise support. | **Characteristics**: Windows has wide software support and ease of use. **Advantages**: Familiarity, extensive support, and compatibility. Strong integration with Microsoft tools. **Weaknesses**: Higher licensing costs and potential security vulnerabilities. **Deployment Method**: Strong support with Windows Server editions.  **Licensing Costs**: High, especially for enterprise editions. | **Characteristics**: Mobile devices are not typically used for server-side applications. **Advantages**: Provides flexibility in testing and can be used for lightweight server applications, useful for development and small-scale deployment scenarios  **Weaknesses**: Limited resources and performance capabilities. **Deployment Method**: Not suitable for large-scale server-side hosting. **Licensing Costs**: Generally none for development, potential costs for app stores. |
| **Client Side** | **Characteristics**: Mac development is costlier but offers a smooth user experience. **Cost**: Higher due to hardware.  **Time**: Requires expertise in macOS and iOS development. **Expertise**: Swift/Objective-C for iOS and macOS. **Development Process**: Consider hiring an independent React-native web developer for under $5k for a hybrid app. | **Characteristics**: Ideal for developers due to cost-effectiveness and open-source programs.  **Cost**: Lower with no licensing fees for the OS.  **Time**: Requires maximum time to learn compared to Mac and Windows. **Expertise**: Requires familiarity with Linux environments and tools. **Development Process**: React-native web can support iOS, Android, and Web platforms with a single codebase. | **Characteristics**: Familiar environment with many unique tools.  **Cost**: High due to licensing fees. **Time**: Moderate development time.  **Expertise**: C#, .NET, and Windows Server management. **Development Process**: React-native for web codebase can be developed using Visual Studio Code. | **Characteristics**: Flexibility for clients to have the app anywhere, anytime. **Cost**: Costs associated with cross-platform tools and app stores. **Time**: Requires adjustments for screen real estate differences. **Expertise**: Requires knowledge of React and responsive design. **Development Process**: Utilizing React for responsive media adjustments for iOS, Android, and web**.** |
| **Development Tools** | **Tools**: VSCode, Homebrew, Xcode IDE, iTerm2, Tower git client, Dash API browser. **Languages**: JavaScript, HTML, CSS, React, React-native, React-native-web. **Impact on Team**: Requires expertise in Apple-specific tools. **Licensing Costs**: Xcode is free; Apple Developer Program $99/year. | **Tools**: Visual Studio Code, Atom, Vim, bash, git, node, Flask. **Languages**: JavaScript, HTML, CSS, React, React-native, React-native-web, yarn. **Impact on Team**: Open-source tools reduce costs, but require expertise. **Licensing Costs**: Generally minimal to none.**.** | **Tools**: Visual Studio, git for Windows, Git bash, Node, yarn. **Languages**: JavaScript, HTML, CSS, React, React-native, React-native-web. **Impact on Team**: Strong integration with Microsoft tools. **Licensing Costs**: Visual Studio has free and paid versions; Windows Server licensing can be costly. | **Tools**: Browsers like Firefox, Opera, Chrome, Metro. **Languages**: JavaScript, HTML, CSS, React, React-native, React-native-web.  **Impact on Team**: Requires knowledge of mobile-specific testing and deployment. **Licensing Costs**: Costs associated with app store deployment. |

## Recommendations

1. Operating Platform:

For the server-side, I recommend using Linux because it's known for its great performance, flexibility, and being cost-effective. On the client-side, it's best to use a mix of macOS, Windows, and mobile platforms to make sure the game is accessible to a broad audience. Linux is stable and efficient for running server applications, and since it's open-source, it can save a lot on costs while offering plenty of customization options. For the client-side, using frameworks like React Native or Flutter will help create a seamless experience across different platforms.

2. Operating Systems Architectures:

We should go with a distributed system that has a central Linux server managing the game logic and data storage. Linux's modular architecture, with its monolithic kernel, is perfect for this. It provides all the essential services like process and memory management, file systems, and device drivers in a single, efficient package. The kernel’s ability to dynamically load and unload modules gives us the flexibility we need. For the clients, we can run applications on macOS, Windows, and mobile devices, all communicating with our server via a RESTful API. This setup will ensure that players can access the game smoothly from any platform.

3. Storage Management:

For managing game, team, and player data, I recommend using a relational database management system (RDBMS) like PostgreSQL or MySQL. These systems are reliable and scalable, and they handle data integrity and efficient querying very well. This will help us manage the large amounts of data generated by the game and ensure everything runs smoothly and securely.

4. Memory Management:

Linux is great at handling memory, which is crucial for our game’s real-time data needs. It uses techniques like virtual memory, paging, and swapping. Virtual memory helps extend physical memory using disk storage, allowing us to run large applications and handle multiple processes efficiently. Paging loads data into memory only when it's needed, improving performance. Swapping moves inactive pages to disk, freeing up RAM for active processes. These techniques ensure that our game can handle the dynamic and intensive data processing needed for a smooth gaming experience.

5. Distributed Systems and Networks:

To make sure "Draw It or Lose It" works seamlessly across different platforms, we should use a microservices-based distributed architecture with a central Linux server. By setting up a RESTful API, we enable different client applications (on macOS, Windows, and mobile) to interact efficiently with the server. We’ll use TCP/IP protocols for reliable communication and implement load balancing to distribute network traffic across multiple servers, ensuring high performance and reliability. Redundancy measures will minimize downtime during outages, keeping the game available and running smoothly for all users.  
  
6. Security:

Security is essential to protect our users’ information. We need to implement a secure login system with usernames and passwords, and ideally, two-factor authentication (2FA) for added security. Regular updates are crucial to fix any security vulnerabilities and enhance overall security. During development, we should follow best practices and secure coding guidelines, including thorough code reviews to catch any potential issues. Regular security audits and assessments, like penetration testing and vulnerability scanning, will help us stay compliant with industry standards and regulations. For data transmission, we’ll use SSL/TLS encryption to keep everything secure. Implementing OAuth2 for user authentication and role-based access control (RBAC) will help protect sensitive data, ensuring only authorized users have access to specific features and information. These steps will create a robust security framework, providing a safe and enjoyable gaming experience for all players.