

<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 | 01/21/2024 | Quinlin MacKenzie | Breakdown of OS considerations with detailed report on recommendations |

**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 aims to develop a web-based game application titled "Draw It or Lose It". The game will involve teams competing to guess a puzzle depicted by rendered images from a stock drawing library. Each game will consist of four rounds, with teams attempting to guess the puzzle within a one-minute time limit. If unsuccessful, other teams will have a chance to guess within a 15-second time limit. To proceed with the development process, we must address critical software design considerations and constraints.

## Requirements

Team and Player Management:

* The game must support multiple teams, each comprising multiple players.
* Unique identifiers are required for each game, team, and player instance to maintain data integrity.

Unique Names:

* Game and team names must be unique to prevent conflicts and confusion among users.

Memory Management:

* Only one instance of the game can exist in memory at any given time to ensure proper functioning and data consistency.

## [Design Constraints](#_2et92p0)

* Application must be able to handle multiple concurrent users
* Data synchronization is needed to maintain consistency across the systems
* Network latency and bandwidth constraints need to be carefully considered to optimize user experience, especially in terms of real time gameplay

## [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)

* ProgramDriver: The main entry point of the program.
* SingletonTester: Tests the implementation of the Singleton pattern.
* Entity: Represents an abstract entity with common attributes and methods.
* GameService: Manages game-related operations and implements the Singleton pattern.
* Game: Represents a game and contains a list of teams.
* Team: Represents a team within a game and contains a list of players.
* Player: Represents a player within a team.

Relationships:

* ProgramDriver uses SingletonTester for testing.
* GameService manages multiple Game instances.
* Game contains multiple Team instances.
* Team contains multiple Player instances.

**"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** | The main benefit is the stability and security of the Unix-based system, and the pre-installed support tools. Drawbacks are the limited hardware options and a general higher development cost. | Offers customizability and flexibility from its open-source system, but it offers a steeper learning curve for beginners and people not familiar with the systems. | Widely-used, which means broad compatability is its strongest asset, but it comes with higher licensing costs and security concerns. | Portability, touch interface, and a focus on user experience are limited by performance constraints and a limitation on resources. |
| **Client Side** | Higher initial cost is offset by faster development, but a fair bit of knowledge of the macOS enviornment is needed. | Lower cost and open-source, but offers a steep learning curve for beginners.  Requires knowledge of Linux environment | Cost of liscencing fees as well as additional add-ons depending on needs. Familiarity also expedites development.  Familiarity with Windows ecosystem required. | Cost varies by device  Familiarity expedites development  Knowledge of mobile development platforms |
| **Development Tools** | Objective-C, Swift, Xcode | C/C++, Python, Visual Studio Code | C/C++, .NET, Visual Studio | Java, Kotlin, Swift, Android Studio, Xcode |

Recommendations

1. **Operating Platform**: Linux is recommended for expanding Draw It or Lose It due to its open-source nature, customizability, and broad compatibility across different hardware architectures. With lower costs compared to proprietary systems like Windows or macOS, Linux offers a cost-effective solution for reaching a broader audience. Its developer-friendly environment, supported by a vibrant community, facilitates adaptation and expansion efforts, providing a familiar and flexible platform for game development.
2. **Operating Systems Architectures**: Linux operates on a modular architecture centered around its Unix-like kernel, managing hardware resources, facilitating communication, and providing essential system services. It relies on system libraries like the GNU C Library (glibc) and features a hierarchical filesystem structure. Linux supports various graphical user interfaces (GUIs) and utilizes package management systems like APT and YUM. It offers robust networking capabilities and prioritizes security.
3. **Storage Management**: The Ext4 filesystem is recommended for Linux due to its reliability, performance, and scalability. It supports large file sizes and offers efficient file allocation and management.
4. **Memory Management**: Linux employs virtual memory, demand paging, and memory allocation mechanisms like malloc and free to optimize memory usage. It also uses caching mechanisms to enhance performance.
5. **Distributed Systems and Networks**: Implementing a client-server architecture enables communication between platforms for Draw It or Lose It. Dependencies include ensuring robust network connectivity, reliability of communication, and security of data transmission. Strategies such as redundant servers, failover mechanisms, encryption, and load balancing should be employed.
6. **Security**: Utilizing encryption techniques, strong authentication mechanisms, and access control lists (ACLs) can protect user information on and between platforms. Regular security audits and timely software updates are crucial for maintaining security.