

<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 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 | 03/24/2024 | Mary Watts | Defining the constraints, create the executive summary, and making the domain model description. |
| 2.0 | 04/07/2024 | Mary Watts | Adding the Evaluation |

**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)

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know in order to proceed with the process you are proposing.>

The problem that is presented is taking our Clients current Android only app “Draw it or Lose it”, loosely based on the hit show *Win,Lose or Draw*, and making it cross- platform and more widely accessible. The game needs to be able to have one or more teams involved with multiple player. Each game and team name must be unique so that the users can check to see if the name is available. The last requirement is that only one instance of the game can exist at a time ( which is achievable by creating unique identifiers for each instance of a the games.

## Requirements

1. Game must have the ability to have one or more teams.
2. Each team must have multiple people.
3. Game/team names must be unique to allow for a check to see if that current name is in use.
4. There can only be one instance of the game in the memory at a time.

## [Design Constraints](#_2et92p0)

This is a currently only available via the app and they would like to be able to be access via a web-based platform as well, this also throws into question the creation of the environment around the site that is created for it. There would need to be works for network communications, security, and most importantly compatibility cross-platform.

Unique names are also one of the constraints as the system would need to enforce the creativity and uniqueness of the game/team/player names, as this prevents the naming conflicts and creates an optimal end user experience during the creating/joining of games.

There is also the matter of maintaining the proper functionality and accommodation of only having one instance of the game service in the memory.

## [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)

This UML Diagram shows the class structure of the game application system itself. At the center of it all is the Entity class that serves as the Parent class and contains all the common attributes such as the game id and the game name, making sure that each name has a unique identifier. Three classes extend the parent class: Game, Team, and Player. They are the key entities of the game application. The ProgramDriver class is the entry point and is where the main function is. It also contains the singleton instance of GameService to make sure that there is only one instance of all key components. The ProgramDriver is solely responsible for adding all key entities using the GameService instance. It also has a dependent relationship with e the SingletonTester so that it can ensure that there is only one iteration of the game/names at a time. Lastly, the GameService class has a composition relationship with the game class meaning that it manages the Game instances and holds all references. Team also has a composite relationship with the Player class and the Team has one with the Game.

The UML class diagram shows instances of inheritance through the relationship between the Entity class and the Game, Team, and Player classes. You can physically see the encapsulation of attributes as well as the abstraction

**"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** | * Stable and secure for web applications. * There is a better environment for developers. * There is a limit to scaling compared to the other OS. | * Completely opened OS for completely customization and flexibility * There are GUI and hardware compatibility/limitation | * There is a wide range of compatibility as it is the most used OS. * There is a large support system for developers using this OS | * Highly portable * Compatible with touch screen and gesture-based * Limited by screensize * Capabilities vary between devices. |
| **Client Side** | Pros:  Wide range of web browsers, and cross-browser testing. Decent development time and deployment time. User-friendly interface  Cons:  Requires expensive Apple products with MacOS | Pros:  Also sports a wide range of browsers. It has a tree file structure making files easier to find. Due to the open-source software it is highly compatible. Free to use/distribute.  Cons:  Steep learning curve | Pros:  Well supported  Easy cross platform testing besides MacOs  Quick development and deployment  Cons:  More difficult to test for Mac OS browser.  Higher costs compared to open-source alternative | Pros:  Expertise with Android app development  Cons:  Difficulty testing with other environments and has longer development times.  Connectivity limitations |
| **Development Tools** | Noe.js and Java are the most commonly used. Can utilize Virtual Machines. Must be reviewed/approved by Apple. 99$/year Apple Dev program  Eclipse for Java | Can utilize Virtual Machines.  Shell prompt and terminal  No license cost  Eclipse for Java | C# and .NET frameworks are the most popular languages for Window-based applications Mainly uses Visual Studio IDE | Kotlin and Objective-C besides Java. There is use of Android Studio as Device emulators and simulators.  99$/year Apple dev program for iOS |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>