

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 | 03/23/25 | Christopher Barnes | Changes were made to the Executive Summary, Design Constraints and Domain Model |
| 2.0 | 4/6/25 | Christopher Barnes | Changes were made to the Evaluation |
| 3.0 | 4/20/25 | Christopher Barnes | Changes made to the recommendation 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 client is requesting a web-based game that is like a 1980’s TV game show called “Draw it or Lose it”. The primary requirements for this are as follows, multiple teams, each team can have multiple players each with a unique game and team name and unique game ID. Only one instance of the game can exist in memory at any given time. The solution of this will be utilize all four object-oriented principles to design and create this web-based application ensuring that only one game at a time has one team name and different player names within the one team.

## Requirements

* *Multiple Teams*
* *Unique Game Names*
* *Unique Team Names*

## [Design Constraints](#_2et92p0)

* *Technical Constraint – Language selection for application development in a web-based distributed environment. There are multiple languages that use various frameworks to develop their web-based programs. You are forced to think about front end and back-end development languages, like Java/Python for back-end and react/JavaScript for front end.*
* Technical Constraint – With potentially large groups for each game, there is a server-side bandwidth capability that will need to be considered for scalability. The larger numbers will require greater architecture for databases, bandwidth and security.
* (I added this one as I thought it could relate to the application) Business Constraint – given that this is a web-based game with an already previous “show history” there would need to be extensive legal research to ensure there are no licensing issues. Legal can be extremely constraining to have the project be built but then not have the approval or rights to create said project. This becomes more constraining if there is a source of income related to the project that may require users to register for access, for example, or membership fees.
* Technical constraint – given that mobile architecture has web access through applications like Safari or Google Chrome, the application would need to have the ability to be viewed on the mobile architecture devices. This could create additional constraints as more developers/skill sets may be required.

## [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)

First are the classes, and then a summary of their utilization as it pertains to OOP principles.

**"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.**

**ProgramDriver: This is the primary class that contains main().**

**SingletonTester: This class is the test for Singleton pattern.**

**Entity: This is the Parent class with three children, Game, Team, and Player that inherit from Entity.**

**GameService: This class manages most of the game operations. It manages the teams, the games, the players and works in tandem with the Singleton pattern to ensure that no duplicates are being created and gives visibility to them if they are.**

**Game: This class manages a game and can manipulate the teams. (This inherits from Entity)**

**Team: This class is like the Game class; it can store and manipulate players within it. (This inherits from Entity)**

**Player: This class holds the player information. (This inherits from Entity)**

**OOP Principles Summary:**

We have four different principles that are used in Object Oriented Programming. **Inheritance**, this is used in the above when looking at our Parent class “Entity” and three Children Classes “Game, Team, Player”. All three children inherit attributes from their parent class. **Abstraction** works within this program by focusing on defining clear, simple interfaces while hiding or abstracting the internal complex methods. For example, Entity can be generic and shares the attributes but hides them from other classes that could share with them in common such as Game, Team, and Player. **Encapsulation** is also present within this web-based application as noticed with the GameService private and public methods. Instead of having every single method public and accessible, Encapsulation protects and provides security to the attributes that don’t need to be accessed/mutated like making the gameID , playerID or TeamID private and not public so that they aren’t changed. **Polymorphism** which acts like “many forms” is also present like that of having the toString() with method override functions. When the method is overridden in child classes like game, team or player for example, it can behave differently or take on a different form depending on what is being done with the object.

## 

## [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 OS has great options for web server-side development like the Apache Http server. This server-side option is open source which comes at a great cost for development fees. Using an open-source project usually has a strong community to help trouble shoot known/unknown errors that may not be easily solved on their own.  *Advantages*:  Mac is commonly known for its secure environment, which is important for setting up authentication and data protection for our users.  *Weaknesses*:  One major downside to using Mac is its potential scalability challenges. Mac is rumored to not be optimized for handling large-scale web server environments which for us if we want to scale our web-based game for thousands of players we may eventually run into limitations for Mac. | *Characteristics*:  Linux OS makes great opportunities for development due to being open source which focuses on being cost-effective for growth. Being free and open source, it also offers scalability and extensive community/documentation to help the development process along the way.  *Advantages*:  Given that Linux started and continues to be open source it offers extreme flexibility to customize our application to suit just about any server-side requirement.  *Weaknesses*:  One major drawback to utilizing development to Linux is that it has a stronger learning curve for beginners. Although great in documentation, the Command Line Interface is known for its intimidation to newer developers which could lead to more bugs/errors for our users. | *Characteristics*:  Windows is one of the most common server-side hosting opportunities. Windows servers are a go to option due to tools like IIS (Internet Information Services) for web server management.  *Advantages*:  Multiple common integrations exist for server hosting through windows like its SQL Server or .net frameworks and database management. This will help create more uniformity for our users and their data.  *Weaknesses*:  Windows, unlike its cousins, tends to have higher costs associated which can prohibit our scalability of the game in the form of licensing. The servers are also usually limited in customization, which forces us to tailor our development towards hardware more than users. | *Characteristics*:  Most mobile ecosystems are limited to two OS, Apple iOS and Android OS. They offer great options for server-side development due to frameworks that typically run lightweight servers like Python based servers. We could consider more cloud-based architecture for server development.  *Advantages*:  Mobile devices don’t put as much strain on data management due to their limited hardware capabilities.  *Weaknesses*:  When working with mobile devices, our API architecture is limited and sometimes restricted when needing to run server-side applications. This can also be a cause for concern for our users due to possible data breaches. |
| **Client Side** | *Cost:*  Safari would be our biggest client aspect as it is the native browser for Mac. Although rapidly growing, the number of developers that know framework compatibility and design for Mac are still limited, which may require further courses/training for development.  *Time:*  Time to invest, which drastically impacts our development process due to insufficient knowledge and training courses needed for understanding.  *Expertise:*  We may need to lean on developers that already have this kind of knowledge base and can increase our costs as we consider compatibility for innate OS Browser but also portability through other browsers on the OS like Chrome/Firefox. | *Cost:*  Given the open-source nature of Linux, we can use highly compatible and cost effective open-source frameworks like React or Angular. This can drastically reduce our costs and given that all is open source it tends to have a free price tag.  *Time:*  Most open-source projects tend to have generic frameworks and open to various languages. The time sink would be to become familiar with Linux browsers and setting them up.  *Expertise:*  This could be a viable option for portability as we won’t be restricted to one specific language due to Linux being open source. Setting up browsers for distribution and testing like Ubuntu would be an essential knowledge base. | *Cost:*  Most developers use windows, and this helps keep our cost low for testing. We can still utilize cross-platform frameworks like React and Angular as well for compatibility between windows browsers.  *Time:*  Time sinks would consider How Edge stays performant as we utilize various cross-platform frameworks.  *Expertise:*  Expertise would be to know how windows work and browser optimization tools/familiarity with Edge. | *Cost:*  Costs drastically go up mobile development. Purchasing the hardware to test on requires multiple devices and forces hardware limitations based on price.  *Time:*  Time becomes extensive as you are testing for multiple devices compatibility.  *Expertise:*  Once peripherals are removed, development becomes much harder as now you are leaning into specialization skills for touch-friendly interfaces. This may require additional expertise and could also force our costs to go up for development. |
| **Development Tools** | *Programming Language:*  Swift is the most common programming language to use for Apple devices.  *IDE/Tools:*  Apple’s official IDE is Xcode but there are various plugins that can be used through Visual Studio Code. There will be additional fees to join the Apple Developer Program and or Enterprise program upwards of $300 per year. | *Programming Language:*  C++|Python is the most common programming languages for Linux web-based app development.  *IDE/Tools:*  You can use most free IDEs like vs code or eclipse for development. Selenium will also work for testing tools. | *Programming Language:*  Python |Java |C++ for programming languages  *IDE/Tools:*  VS code, frameworks, and Selenium will also work with windows. We can use mostly the free version of VS code but as we scale our game we would want to consider paying for the Enterprise Versions of Visual Studio via a subscription plan. | *Programming Language:*  Swift for Apple iOS development and Kotlin for Android. Given that we are working two separate OS for mobile devices we may need to consider running two separate dev teams.  *IDE/Tools:*  Android Studio and Xcode all work great for IDEs to use for development. For iOS we can use their IDE Xcode. There will be additional fees to join the Apple Developer Program and or Enterprise program upwards of $300 per year. |

## 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**:
   1. We would recommend utilizing Windows as the primary operating platform to allow the expansion of The Gaming Room, Draw It or Lose it. Although windows may have higher costs sometimes compared to other platforms, it is also more common amongst developers and has great documentation and scalability.
2. **Operating Systems Architectures**:
   1. Windows primarily supports the Model-View-Controller (MVC) architecture, which is crucial for the development of our user interface. The MVC pattern divides the application into three interconnected components: the model, which manages the logic for storing and retrieving data; the view, which delivers the user interface for interaction; and the controller, which acts as an intermediary to facilitate communication across the application. This structure ensures modularity and enhances the application's maintainability and scalability.
3. **Storage Management**:
   1. Azure Storage is an ideal choice for cloud storage due to its cost-effectiveness. It offers a pay-as-you-go model, eliminating the need for purchasing hardware that requires regular provisioning, maintenance, and upgrades. Additionally, SQL Server has been selected as the database type, leveraging its open-source nature and extensive support history to ensure reliability and scalability.
4. **Memory Management**:
   1. Windows utilizes x86 architecture, enabling higher memory allocation and larger memory sizes. This is particularly important given the need to manage memory efficiently for concurrent gameplay across multiple games. Windows offers robust memory allocation and deallocation techniques to optimize performance. To ensure smooth and efficient operation of our multi-game environment, a minimum of 32GB of RAM is recommended for handling the required memory demands.
5. **Distributed Systems and Networks**:
   1. To facilitate connectivity with our database and storage solutions, we will implement APIs using REST API architecture. This approach ensures seamless integration across multiple platforms, accommodating users who may not be on Windows. Additionally, we will leverage the .NET framework architecture provided by Windows to execute the program effectively, ensuring compatibility and robustness.
6. **Security**:
   1. Windows offers multiple security layers, including the use of user profiles as guest profiles with User Access Control (UAC), which helps safeguard sensitive user information. In terms of database security, it’s essential to account for threats such as DDoS attacks and malware that could compromise client data. Azure seamlessly integrates with Windows to strengthen security measures, providing DDoS protection, enhanced firewalls, and other advanced tools to ensure robust defense mechanisms.