

Creative Technology Solutions

# **CS 230 Project Software Design Template**

Aaron Bias

Version 1.2

## 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 | 09/17/2023 | Aaron Bias | Executive Summary, Requirements, Design Constraints, and Domain Model updated. |
| 1.1 | 10/01/2023 | Aaron Bias | Server Side, Client Side, Development Tools updated. |
| 1.2 | 10/15/2023 | Aaron Bias | Recommendations Updated. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants would like its game “Draw It or Lose It” that is currently available only on Android, to be ported to a web-based game. To ensure the core gameplay remains true to the original, a comprehensive design review is included below.

## Requirements

* Multi-Team Support: A game will have the ability to have one or more teams involved.
* Team Management: Each team will have multiple players assigned to it.
* Unique Names: Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* One Instance: Only one instance of the game can exist in memory at any given time.
* Render: Must render images from a large library of stock drawings
* Web-Based: Must be ported to a web-based environment

## [Design Constraints](#_2et92p0)

* Budget: Cost can grow over the budget if the timeline is not met, or additional resources are needed to implement key features.
* Scalability: The game must be able to scale with a growing user base.
* Web-Based Browers: The game should be playable on a wide range of browsers to ensure accessibility.
* Performance: The game should perform well under load.
* Schedule: Meeting deadlines could be impacted by the above constraints.

## [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 Entity Class is the super class in which Game, Team, and the Player class inherit its attributes. The UML diagram shows what variables and methods will be used in our development. The main program is within the class ProgramDriver and uses the SingletonTester to test the game. GameService is the “backbone” of the game and the lines connecting each class with the 0...\* tells us they are associated. The 0...\* tells us it can have zero to many associates. This UML diagram shows a few of the OOP principles of Inheritance, Composition, and Encapulation.

**"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 a secure operating environment and is built on a Unix foundation. High quality hardware at a high cost.  Limited Scalability.  Licensing cost will be a factor. | Linux is open source and stable. Cost effective as it is open source. Security in Linux is access control which is crucial for hosting web applications and has support for a wide range of web technologies. The wide range of distros makes compatibility challenging. | Windows is user friendly with its easy GUI. Server side it supports the .NET framework which is a great choice for hosting web applications. Licensing costs can be expensive compared to Linux | Poor choice for hosting as mobile device is Network dependent and many carriers have limited coverage. Making an app as a Progressive Web App and available through the app stores will strengthen user adoption. |
| **Client Side** | Cost: Expensive for development as Mac hardware will be required for testing.  Time: Extensive. Development and testing on different macOS versions and browsers will require time.  Expertise: macOS technologies (Swift, Objective-C) | Cost: Cost effective as Linux is open source. Few distributions may require some licenses.  Time: Extensive. Development compatibility for a wide range of distros could be time intensive.  Expertise: Knowledge in Python, C/C++, Java | Cost: Moderate for development as securing hardware is cheaper than MAC. Windows license will add to the expenses.  Time: Moderate. Windows and Browers versions compatibility will increase development time.  Expertise: C#/C/C++ and Windows APIs | Cost: Expensive for development as a wide range of mobile hardware will be required for testing.  Time: Extensive as you are developing for two platforms (iOS and Android). App store review process could extend development time.  Expertise: App store review process, good UI designer and Swift/Java for platform specific development. |
| **Development Tools** | Languages: Swift & Objective C.  WEB Languages: HTML, CSS, JavaScript.  Tools: Xcode is the IDE from Apple.  Other: Homebrew with the various libraries and tools for macOS. | Languages: Python, C/C++, Java.  WEB Languages: HTML, CSS, JavaScript.  Tools: Various IDE based on preference. Eclipse, Visual Studio, IntelliJ  Other: Git, Package manager (APT), GNU Make, GDB (Debugging). | Languages: C#, C/C++, Java, VB.NET  WEB Languages: HTML, CSS, JavaScript.  Tools: Various IDE based on preference. Visual Studio, Eclipse, IntelliJ.  Other: Electron for developing applications that work on other platforms. | Languages: Swift, Objective-C, Java, Kotlin.  WEB Languages: HTML, CSS, JavaScript.  Tools: Xcode for Apple products. Android Studio for Android devices.  Other: Single Codebase for both platforms (cost effective) would be Flutter (open-source), React Native (Facebook), Xamarin (Microsoft). |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

* **Operating Platform**: The most widely used and appropriate platform to use to expand Draw It or Lose It to other computer environments would be the Windows Operating System platform. The upfront cost would be low as most hardware used today already has the OS on it and the widely available IDEs for various languages makes development efficient to port the game to other platforms. Compatibility with most popular web browsers ensures the game will grow its user base.
* **Operating Systems Architectures**: Windows provides a robust set of tools that makes the architecture versatile and able to meet the goals of our client.
* Windows Server Architecture: A group of operating systems designed by Microsoft that supports enterprise-level management, data storage, applications, and communications.
* .NET Framework: Great platform for hosting web applications
* Compatibility: Window's compatibility with most software and hardware is unmatched, making porting of the game more efficient.
* IDE: Windows has a host of Integrated Development Environments that make development more efficient and reduces development time (saving costs).
* **Storage Management**: The Window’s architecture (Server architecture and/or .NET framework) is compatible with cloud-based storage solutions from Amazon, Microsoft and Google. The benefits of using a cloud-based storage solution are the combination of scalability, redundancy, and flexibility, making it a great choice for data storage and management.
* **Memory Management**: Windows provides various techniques in managing memory such as:
  + Virtual Memory Management: This tool allows Windows to efficiently manage memory between both RAM and disk space, allowing the use of disk space as an extension of memory.
  + .NET Framework: .NET framework offers a range of memory allocation and deallocation functions. These functions ensure the game can dynamically allocate memory as needed and release it when not in use. .NET also includes a garbage collection tool that reclaims memory occupied by objects no longer in use.
  + Pagefile Management: Windows allows the customization of pagefile (virtual memory) settings, which can enhance the performance of Draw It or Lose It.
* **Distributed Systems and Networks**: To communicate seamlessly across different devices and platforms and to utilize the cloud based storage, a Microservices architecture would be beneficial for the game Draw It or Lose IT. Breaking up the application into smaller components that is served through either Google, Amazon, or Microsoft (Google has a host of tools to implement microservices) would mitigate several challenges and risks such as:
  + Scalability Issues
  + Flexibility
  + Fault Isolation
  + Development Speed
  + Resource Efficiency
  + Cross-Platform Compatibility
* **Security**: To ensure the user’s data is secured on the various platforms, the following steps are crucial:
  + Data Encryption: End-to-End encryption will ensure that data is safeguarded when in transit between devices/platforms.
  + Authentication: 2FA would be a great added security protocol to implement as it would ensure only authorized users could access the game.
  + Cloud and Network Security: When using the cloud for our architecture needs, we should follow their best practices to secure data and help prevent unauthorized access.