

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 | 07/12/23 | Rafael Canseco | Cover page, document revision history, executive summary, requirements, design constraints, system architecture view, domain model, and recommendation were all modified. |
| 1.1 | 07/30/23 | Rafael Canseco | Evaluate OS for client application “Draw It or Lose It” |
| 1.2 | 09/12/23 | Rafael Canseco | Recommendations added |

**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 has expressed the desire to expand the gaming application to multiple operating system platforms using different software patterns within a distributed environment. To address this design challenge, our team will assess and evaluate the various operating systems under consideration to determine the optimal platform for running the gaming program.

## [Design Constraints](#_2et92p0)

## Game needs to run on multiple platforms.

* Include multiple teams of multiple users.
* Be able to check for unique team/game names must be present.
* Only one instance of the game must be allowed at a time.

The game's inherent requirements encompass these constraints. From a developmental perspective, The Gaming Room aims for the game to be compatible across various platforms. Thus, we will need to adapt the functional Android code to these other platforms, potentially necessitating additional developers proficient in the respective languages to undertake this task.

## [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)

Game, Team, and Player, which together form the building blocks of an Entity unit. These classes are connected through associations, as indicated by the 0…\* symbol. A team unit can have multi-players, each with a unique ID and name. Similarly, a Game unit can have multiple teams added to it. CTS will save each unique set of Game, Teams, and Players as an Entity unit. To organize the information effectively, CTS will use the Game info to transfer the data into the Game service. From there, the program will iterate through each team and okayer, displaying the fame details to the console. **"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 graphical user interface offers a seamless and user-friendly experience, providing ease of use. Mac stands out for its straightforward accessibility and simplified server configuration. | The platform can be challenging to navigate initially. Similar to Mac, Linux features a command shell for straightforward server configuration and accessibility. Moreover, Linux is known for being cost-friendly. | Windows servers, like Mac, can be costly. However, they offer a user-friendly graphical user interface and a wide array of software options for developers. Additionally, Windows provides the convenience of a command prompt for more advanced configuration and tasks. | Mobile device specifications vary widely among users and platforms, making them cost-effective options. Android, iOS, and Windows Phone (WP) are the main platforms. The main challenge lies in developing a game that is compatible with the majority or all mobile platforms, considering their diverse specifications. |
| **Client Side** | The expense would resemble that of a Windows setup, given that these operating systems are not open source. The timeline would be contingent on expertise, with individuals familiar with Mac requiring less time, whereas those less acquainted would necessitate a longer duration. | The expense, if applicable at all, would be minimal with Linux, given its open-source nature. Optimum time and an elevated level of expertise would be essential, as Linux is less commonly utilized. It would be imperative to engage a proficient Linux user and allocate them ample time to address the task, considering that even experienced individuals may find Linux challenging to navigate. | Regarding users, the cost projection would align with that of a Windows setup, given the non-open-source nature of these operating systems. The required time would correlate with the user's expertise: those well-acquainted with Windows would demand less time, whereas individuals with limited familiarity would necessitate a more extensive duration. | The cost factor would be relatively manageable concerning mobile devices. Experience might not pose significant challenges, given that mobile devices are more user-friendly to operate. However, a more extended timeframe would be essential due to the involvement of various operating systems and numerous mobile devices requiring attention. |
| **Development Tools** | Swift stands out as the predominant language employed for crafting Mac applications. Several Integrated Development Environments (IDEs) are available for Swift, including Atom. | Eclipse and Atom find widespread usage as Integrated Development Environments (IDEs) within the Linux environment. While Eclipse is predominantly employed for Java, it also extends its support to other languages, including C+. | Eclipse and Visual Studio emerge as widely embraced Integrated Development Environments (IDEs) within the Windows domain. Visual Studio serves as a versatile platform suitable for HTML, C#, and JavaScript development, among other capabilities. | In the realm of iPhones, the development tools mirror those utilized for Mac systems. iOS applications are commonly crafted using Swift, even though iOS and macOS differ significantly in certain aspects. |

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**: After thoroughly reviewing my research, I am inclined to assert that Windows stands out as the optimal choice for an operating platform. This preference is rooted in its widespread usage, making it the operating system with which most individuals possess functional familiarity. Windows boasts a plethora of compatible Integrated Development Environments (IDEs), and the overall cost associated with its utilization tends to be more budget-friendly.
2. **Operating Systems Architectures**: The architecture of Windows enables applications to harness the capabilities of the platform's kernel processes without causing direct disruption to those processes. In simpler terms, applications can leverage Windows' potential to establish graphical user interfaces (GUIs), access memory, and engage with essential processes that constitute the application, all without inadvertently interfering with the core operations that sustain the functioning of the operating platform.
3. **Storage Management**: Within the Windows ecosystem, inherent functionalities like Disk Management and Storage Sense are integrated directly into the operating platform. Furthermore, a tool known as Disk Cleanup is available for use. Disk Management serves as a Windows system utility primarily intended for more complex storage operations. On the other hand, Disk Cleanup and Storage Sense serve the purpose of assisting in the maintenance of system storage by identifying and eliminating superfluous files that occupy valuable space.
4. **Memory Management**:   
   Memory Management is an inherent component of the Windows system, encompassing a built-in utility. Our project would involve the creation of a database to house all the game's image files, streamlining their accessibility for seamless integration within the application.
5. **Distributed Systems and Networks**: In this scenario, a client-server distribution system will come into play. Each client application will rely on a central server application for our game, allowing customization to align with the strengths of each client's system. A robust server network is imperative as the game's success hinges on facilitating multiple clients connecting to a singular server to partake in a unified gaming experience.
6. **Security**: Windows Defender operates as an innate security feature within the Windows operating system. Our approach would involve employing established encryption techniques to safeguard all data transmitted bidirectionally, ensuring its confidentiality and integrity.