

<Eclipse>

# **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 | <09/24/25> | <Victoria Venutra> | <Brief description of changes in this revision> |

**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)

Draw It or Lose It is currently an Android only mobile app game. The Gaming Room is looking to develop the existing game into a web-based game available on multiple platforms. We need to translate the current mobile app into a program available online for everyone.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

1. Each team/player must have a unique identifier
2. A single instance of the game can exist in memory at any given time
3. Cross-platform development: The game must cooperate with players on different platforms
4. Must support multiple teams per server, and multiple players per team
5. Sufficient data storage that manages the game data and allows updates when needed

## [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)

There are five classes depicted in the diagram: Entity, GameService, Game, Team, and Player. Between these classes lie a generalization line that points from three subclasses (Game, Team, Player) to a superclass (Entity). This means that all three subclasses share the same attributes, which in this case are ID and name. We see this as Inheritance, allowing a subclass to inherit properties and behaviors from a superclass, promoting code reuse. This will help us develop the program without rewriting the same attributes for all three classes and risk making a mistake. Moving on from generalization, we find association running in between GameService-Game, Game-Team, and Team-Player classes. This shows an open-ended relationship, where one can change without determining the other.

Begin with the Player, each player is stored having an ID and name. Those players can then be added to a team, copying over ID and name, and the team can be added to a game, containing players, ID and name. You can add players to a team, but you cannot add players to a game. As well as you cannot add a game to a player. They can take from their associated classes, but it will not affect the class being taken from.

**"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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is well known for its tight integration with apple devices and services, leading to increased infrastructure costs. macOS supports server-based deployment using Apache, but it’s rarely used for large-scale hosting since it lacks server optimization. They offer reliable and user-friendly backup system as well as Mac-friendly control panels for managing services. | Linux is an open-source platform best known for its great security, with community vetted patches, and a higher rate of stability. Offering a lot of opportunities for creativity. Linux has a steeper learning curve as well as manual configuration of services but lacks licensing fees. | Windows supports a wide range of enterprise software, but they require licenses to access many enterprise tools. They have plug-and-play, rich development and ecosystem, and security features. You may see these security features anytime you open your computer. | Mobile devices are primarily app centric. Unlike other operating platforms, mobile devices are always connected using cellular or Wi-Fi. They are not suited for hosting since there is no static IP, limited uptime, and unstable network conditions. There is limited ability to handle large-scale traffic. |
| **Client Side** | On the scale, macOS is more expensive due to the hardware requirements, licensing, and testing subscriptions. (hardware investments, licensing and subscriptions, and testing devices) Most time would be spent cross-platform testing: different screen sizes, OS versions, and input methods. The developer needs to have a clear understanding of macOS’s tools and security model. | Linux is open-source and cost-efficient. It supports Android and excels at web development, but iOS testing requires remote Mac access. To use Linux, you need familiarity with desktop, mobile and web stacks (HTML/CSS/JS). | Windows requires paid subscriptions for Microsoft developer tools and services, plus additional costs for testing devices and iOS emulators. Development does take more time due to separate pipelines for desktops, mobile, and web. It also requires remote Mac access. Developers must be proficient in Windows APIs, .NET, and Visual Studio. | Mac mobile devices require a Mac and Apple Developer Program subscription. Other devices might require cross-platform frameworks, cloud testing services, and device farms. You must design for touch input, varying screen sizes, and orientation. App store submissions introduce wait times and review delays. Developers must understand mobile constraints like battery, network variability, sandboxing, and permissions. |
| **Development Tools** | Swift and Objective-C are the typical development programs, with X-code being the primary IDE. JavaScript and Dart are both great multi-platforms developing tools. Developers will need a good understanding of the apple ecosystem beforehand. A downside is the cost, with the developer program being $99 per year and per user. | Linux usually has C++, Python, Java, JavaScript and Dart with IDEs like Visual Studio Code and Eclipse. These are great tools for web-based deployment, but iOS requires access to a macOS. The best part, Linux has many open-source and free tools. | Windows usually uses JavaScript, .NET, and Dart with Visual Studio as the primary IDE. The cost can be significantly higher than macOS or Linux due to paid subscriptions. Developers would need expertise in Windows APIs and .NET, which may increase team size or require many different roles. | Because mobile devices usually incorporate one of the three operating platforms, costs remain the same. Swift, Kotlin and Java, Dart, and JavaScript can all be used, with the IDEs including Xcode for iOS, and Android Studio for Android. A larger team would be required on this project to better support each platform. |

## 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**: To achieve the software requested by The Gaming Room, a cloud platform would be the best option. Cloud platforms support multi-platform deployment, making it easier to expand “Draw It or Lose It” across various operating systems. They offer scalable infrastructure, centralized updates, and compatibility with containerized environments like Docker, which streamlines development and maintenance across diverse systems.
2. **Operating Systems Architectures**: The cloud favors microservices, containers, and distributed systems, while traditional OS often uses monolithic designs with tight coupling to the host operating systems. The benefits of cloud architecture are scalability, elasticity, fault tolerance, and easier global deployment and updates.
3. **Storage Management**: Google Cloud Storage is a great choice for storage management as it is vastly used within multiple operating systems. It provides object-based storage, scalability, multi-platform access, good security and storage classes.
4. **Memory Management**: The cloud is a great tool when you are not sure how much memory is needed. The platform’s memory can be managed through container orchestration. Having this kind of memory management system can ensure that resources are distributed according to real-time demand. Serverless functions are another tool integrated in the cloud platform, meaning that developers write code without managing any servers. Games will scale efficiently while maintaining performance across multiple environments.
5. **Distributed Systems and Networks**: When looking to communicate across various platforms, distributed architecture should be implemented. RESTful APIs are especially useful to exchange data over the internet in real time. It is much easier to ensure global accessibility, so anyone, anywhere, can access the platform at the same time with minimal buffering. But what if the user’s device loses internet connection, how do we protect their progress? The developers can implement local caching and reconnection strategies to preserve user progress and continue right where they left off when the connection is restored.
6. **Security**: Pairing Google Cloud Storage with a cloud-based operating platform can already add security to various platforms. A cool feature in Google Cloud is its Identity and Access Management support. This provides developers with the ability to restrict users based on roles. A second security measure could be securing RESTful APIs with OAuth 2.0, which creates token-based authorization. Another good way to restrict unauthorized users. Other than restricting user access, we can monitor those unrestricted users by enabling audit logging, firewall rules, and network segmentation. This way, the program can be monitored from the inside, making sure authorized personnel do not use the program maliciously.