

<Name-of-Software-Application>

# **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 | 05/22/2025 | Jose Tellez | <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)

The client wants to grow their current game, *Draw It or Lose It*, which is now only on Android, into a web-based version that works on different platforms. They want to build the new version using the existing Android app as a starting point and need help setting up the tools for development and making the process more efficient. The game must be created using tools that allow it to run on many types of devices. Each game and team must have a unique name, and only one game session should be active in memory at a time. Each team will have multiple players, and the game should support matches with one or more teams.

## Requirements

To match what the client needs, the game must allow one or more teams in each session, and each team must have several players. It's very important that only one version of the game is running at any given time. This will be managed by giving each game, team, and player a unique ID. Also, users must be able to check if a team name is already in use when they create a team, so every game and team name must be unique.

## [Design Constraints](#_2et92p0)

One of the main challenges in this project is building the game to work on more than one platform. Since the client’s team might not have much experience with cross-platform development, they may need different teams to work on parts for different platforms. Also, the finished game must meet all the client’s needs on every platform it runs on.

## [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 system uses an object-oriented design with a base class called Entity, which is extended by the Game, Team, and Player classes. Each class inherits common features from Entity and also has its own unique properties. The GameService class handles core rules, such as ensuring only one game instance runs at a time and that all names are unique. The ProgramDriver contains the main code and uses SingletonTester to enforce the one-game rule. A Game holds a list of Team objects, and each Team holds a list of Player objects. Players are linked to teams but do not directly reference their team or game, keeping the structure simple and flexible.

**"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** | A big plus of macOS is that it can run apps made for macOS, Windows, and Linux all at once. This makes it very flexible. It's also known for being consistent over time, so people used to macOS find it easy to use. The downsides are its higher cost and limited hardware options compared to Windows or Linux. | Linux is popular because it’s free, open-source, and has many versions to choose from. It’s very customizable, which makes it great for servers and special systems. Linux also has strong built-in security. However, it doesn’t have as many ready-to-use devices, and it may have issues with some file types and software. | Windows works well in business settings because it connects easily to corporate networks and tools without needing extra software. On the downside, it's not the best for mobile app development, and its weaker security can make it more likely to be hit by viruses or hacking unless extra protection is added. | Running a web app on a mobile device can work well if the app is simple and doesn’t have many users. Android devices, especially, have free or low-cost server apps. But mobile hosting often uses the cloud, which can be less secure. Phones and tablets also tend to be more open to security risks than computers. |
| **Client Side** | macOS is easy to use once you're familiar with how it works. But it only runs on Apple devices, which can limit who can use it. Developers trained to build macOS apps may be stuck if they don’t have access to a Mac. | Linux is free and gives developers a lot of control. It’s easier and cheaper to manage since it’s open-source. But because it doesn’t come with built-in support, users often have to handle their own security, unlike with Windows or macOS. | Windows is widely available and comes in different price levels to match your needs. It also includes built-in support and strong security tools. The downside is that you may need someone skilled in Windows, and some features may cost extra. | Mobile devices are cheap and easy to use, with lots of apps available. But they don’t offer all the tools that desktop systems do for development. Also, each device usually runs a different operating system, which can make it harder to create apps that work across all platforms. |
| **Development Tools** | macOS development mainly uses the Swift programming language. Most developers use Xcode or Xcode Cloud, which are tools made by Apple. Xcode Cloud helps teams build, test, and launch apps faster and more easily, especially when working together. | Linux has many development tools, giving developers lots of flexibility. One popular tool is Docker, which helps create a stable environment, makes it easier to build apps for different systems, and speeds up deployment. Docker Hub also offers ready-to-use setups so developers can start working right away. | Windows was built using the C programming language, with some parts in assembly. A common tool for Windows development is Microsoft Visual Studio. It works as an all-in-one code editor, development platform, and version control tool. Although it can take some time to get used to, many developers now use it regularly for school and work projects. | Java is a top choice for mobile app development, especially on Android, because it's object-oriented and reliable. Other languages like Python and C++ are also used, mainly for mobile games. Tools like Visual Studio Code, IntelliJ IDEA, and Eclipse are common, though I personally prefer Visual Studio with Xamarin for building apps that work on multiple platforms. I find Eclipse harder to use due to its cluttered layout. |

## 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**: Windows is the best choice because it works well with Android tools, making it easy to connect with the existing app. Since about 90% of users use Windows, there are more developers and tools available for cross-platform work.
2. **Operating Systems Architectures**: Windows 10+ uses Universal Windows Platform (UWP), which helps developers build apps that run on many devices using common tools. This makes UWP a good fit for this project.
3. **Storage Management**: It’s best to use a central server to store game data. This setup improves reliability by having backup systems and allows automatic data saving. It also helps the system run faster and keeps the game working smoothly for many users at once.
4. **Memory Management**: Windows manages memory well with both physical and virtual options. Cloud services like Azure and OneDrive help store data and manage app versions, which is useful for handling many users.
5. **Distributed Systems and Networks**: Using a cross-platform tool like Develop4 can make it easier to create apps for different systems without needing many specialists. To keep the app running well, servers should be built to handle the expected number of users, so the game stays available even during busy times or if there are connection problems.
6. **Security**: Keeping user data safe is very important. Basic protections from the operating system aren’t enough, so extra security is needed. Aura is a good option because it protects Windows, macOS, Android, and iOS devices. It offers strong data security, identity checks, and 24/7 customer support based in the U.S., helping to keep users safe and the app running smoothly.