

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 10/09/2022 | Bradley Cook | <Brief description of changes in this revision> |

## [Executive Summary](#_sbfa50wo7nsh)

Our client, The Gaming Room is wanting us to develop a web-based game called Draw It or Lose It. They want it to be available on multiple platforms as it currently only runs on Android. It is loosely based on the 1980s TV game Win, Lose, or Draw. A picture is rendered by the application and as it is being rendered, the players must guess what it is before time runs out.

## [Design Constraints](#_2et92p0)

* Multi-platform
* Multiple teams and users
* Mobile platforms

## [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 Player, Team and Game classes all inherit from the Entity class. The GameService class references the Game class. The Game class references the Team class in the teams function and the Team class references the Player class with the players function.

**"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** | Characteristics:  Proprietary, increasingly popular  Advantages:  Stable  Weaknesses:  Can be costly due to cost of licenses. | Characteristics:  Open-source, popular OS for web servers.  Advantages:  Many Linux licenses are free of charge, making this the obvious winner in terms of price. stable  Weaknesses:  Difficult to use | Characteristics:  Proprietary, ease of use.  Advantages:  Ease of use, software compatibility  Weaknesses:  Unstable, can be costly due to cost of licenses. | Characteristics:  Advantages:  Availability, cost-effective  Weaknesses:  Low security |
| **Client Side** | Not a lot of expertise required. Relatively easy to use. Both Windows and Mac will have similar cost constraints. | A lot of expertise required. Cost effective. | Little expertise is required as most people are familiar with Windows. Easy to use. Both Windows and Mac will have similar cost constraints. | Little expertise required as most people have a smart phone and are familiar with interface. Cost effective. |
| **Development Tools** | XCode is a popular IDE for Swift as it is preinstalled on Macs. A popular language used with Mac is Swift. Since this would be a web-based application, HTML, CSS and JavaScript will be used as well as other languages for the backend such as Java and Python. | Visual Studio is popular in both Windows and Linux. Since this would be a web-based application, HTML, CSS and JavaScript will be used as well as other languages for the backend such as Java and Python. | Visual Studio is popular in both Windows and Linux. Since this would be a web-based application, HTML, CSS and JavaScript will be used as well as other languages for the backend such as Java and Python. | For iOS devices, Swift can be used and for Android, Kotlin and Java can be used. Since this would be a web-based application, HTML, CSS and JavaScript will be used as well as other languages for the backend such as Java and Python. |

## 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**: I recommend the use of Linux for the application. I make this recommendation due to Linux being a very widely used platform for servers. This is due to the highly reliable nature of Linux. It requires little to no downtime due to being able to stay on during updates and most maintenance. It is relatively lightweight since it can be run entirely without a graphical user interface. This is especially useful because it will be run in the cloud, and this makes it easier to maintain. Linux is also an open-source platform and requires little to no licensing costs, as opposed to Windows and Mac.
2. **Operating Systems Architectures**: Linux allows for multiple users to access it’s resources at a time. This is ideal for “Draw It or Lose It” because the application requires multiple users to have access at a time.
3. **Storage Management**: I recommend the use of Linux Logical Volume Manager (LVM) for storage management. I make this recommendation because LVM allows for a Logical Volume Group (LVG) to span across multiple drives. It also allows for the administrator to increase the volume sizes on the fly, with the filesystem mounted and online. This is great, because it does not require the administrator to stop applications, shutdown the servers, or have to rebuild the filesystem. It allows for greater availability to the end users.
4. **Memory Management**: Linux utilizes virtual memory. Virtual memory splits an application into pages and gives the virtual memory addresses physical addresses. The application only sees the virtual memory that has been assigned to the application and the Memory Management Unit on the CPU handles the mapping from virtual memory to physical memory. This allows for the memory to be utilized in a more efficient way and for multiple applications to be run on the system. In the case of “Draw It or Lose It”, this will allow multiple instances of the application to be run without running out of memory space due to fragmentation.
5. **Distributed Systems and Networks**: A distributed system consists of independent nodes in the system that perform separate, independent functions. It allows for geographical dispersal of the nodes which decreases latency for end users across the world. Nodes can be placed in different locations around the world, allowing users to connect to the closest node. This also increases reliability, in the case of one node failing, another one can take over until the failed node is brought back online. Distributed systems are also extremely scalable. When the demand grows, other nodes can be added to the system. The distributed system appears as a singular interface to the end user, improving on ease of use and making for a better overall experience for the user. Aside from all the advantages I have talked about, distributed systems also come with disadvantages. With multiple nodes and multiple layers to the system, and with them being spread out geographically, it can be harder to detect faults in the system. When a node goes down, it can failover to another node, appearing as though nothing had failed at all to the end user. It also adds complexity with multiple serves spread out geographically and adds complexity to the network.
6. **Security**: Linux utilities privileges for all its files. Everything on Linux is a file, to include applications, files, devices such as keyboards and printers, and even storage devices. This allows for permissions to be set for each file independently, giving the administrator more access in securing the system. The root user, like the administrator user on other operating systems, has the most permission in the OS. Mos applications and users do not have root access, they only have access to what they require access to complete their tasks. Permissions can also be changed on the fly, allowing permissions to be escalated temporarily to complete a task then set back to the original permissions once the task has been completed. Linux, as I’ve mentioned before, is an open-source OS. This means that contributors all have access to the source code. This leads to more eyes on the source code and more chances for security flaws to be caught and corrected. Once a flaw is corrected, a patch is released. These patches can usually be installed on the fly without requiring a reboot of the system. This greatly increases availability. Lastly, every action in Linux is logged, Administrators can review these logs for security related incidents and respond appropriately, oftentimes before a breach has even occurred.