

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 | 11/10/23 | Dennis Ward | Changed Executive Summary, Requirements, Design Constraints, and Domain Model. |
| 1.1 | 11/23/23 | Dennis Ward | Completed Evaluation |
| 1.2 | 12/6/2023 | Dennis Ward | Completed Recomendations |

**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 a gaming application inspired by the 1980s television game Win, Lose or Draw, employing a digital approach where stock drawings serve as clues instead of manual sketches. The game comprises four one-minute rounds, with drawings completed at the 30-second mark. If a team fails to guess within the allotted time, other teams get a 15-second chance for a single guess. The client, The Gaming Room, seeks assistance in setting up the development environment for the web-based version. The software design document should address the ability for one or more teams, multiple players per team, and enforce uniqueness for game and team names. To ensure memory efficiency, unique identifiers for each game, team, and player instance are required. Hardware requirements will be determined later based on software decisions.

## Requirements

* Set up basic functionality for Draw It or Lose It game.
* Application must support Multiple players within multiple teams. Games and Team must have Unique Names, while Games, Teams, and Players must have a unique identifier.
* This is a web-based app.

## [Design Constraints](#_2et92p0)

* A web-based app needs to be usable on multiple different platforms (Desktop, smartphone, tablet) and usable with many different web browsers.
* A web-based app needs to be able to handle a large user base.
* Security of our user’s data is a top priority in a web-based application.
* Application must support Multiple players within multiple teams. Games and Team must have Unique Names, while Games, Teams, and Players must have a unique identifier.

## [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 below UML diagram clearly shows inheritance with Game, Team, and Player all inheriting from the Entity class. Many of the attributes are private, which shows a good example of encapsulation. The toString method is overridden throughout the subclasses which is created in the super class which is a good example of polymorphism. The program takes advantage of the singleton pattern in the GameService class to make sure only one instance of them is running at any given time. Encapsulation is shown through the instantiation of the Object to be hidden and controlled within its own class rather than being instantiated elsewhere. Abstraction is shown in the singleton pattern when we use the getInstance method to obtain a new or already running object of GameService. The builder class is also implemented within this diagram. The Entity class is set up to build the Game, Team, or Player this is a good example of encapsulation as the actual Entity class runs the high-level functions of the Game, Team, Player class. The builder pattern is specifically made to create subclasses that will specify what we are building (Game, Team, or Player) this is a good example of inheritance and Abstraction. Polymorphism takes a large role in the builder class as well as each subclass will need to override some methods and change the functionality to specify what we are building.

**"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** | macOS is not very common for web hosting so expertise in this area would cost in development time, to add to the cost there are licensing fees associated with server hosting with mac. The advantages are the easy-to-use terminal commands and graphical interface. Apache can be configured on a macOS system. It should be noted macOS provides a great deal of security to server hosting. It also provides great hardware. | Linux is the most used OS for server hosting due to its inexpensive nature. Linux is open source therefore there are no licensing fees, and it can run a very broad selection of programing languages. Linux OS can change from system to system which is the pro and con making it not work well with a large assortment of software but the ability to customize the system the way it will run best for your server. Specifically for server hosting there is a large assortment of software you can choose from to host Apache, Nginx, and others. | Windows has consistency issues, there is often bugs, blue screens, frequent updates, tends to run slower than other os’s and costs more than Linux. Easy deployment, easy graphical interfaces, backwards compatibility. Can run ASP, .NET, and many other helpful tools for deployment and support. | There are a lot of issues with hosting a web based game on a mobile device as this is not there intended purpose. The cons would be Hardware, software, and compatibility the list can go on. The easiest way to do this is with NodeJS which does not allow for an scalability. Not to say this could not be done but for a large and populated game across multiple platforms it is not ideal. |
| **Client Side** | The development time would be somewhat more expensive than other systems because expertise in these fields would be needed leading to longer development times, mac supports all major browsers. | This is a similar setup and installation to Windows many mainstream browsers work with Linux. This is a difficult operating system to use without experience. Development will be much the same as windows, except for testing and quality control as Linux can run differently from system to system so ensuring a quality experience will need additional testing and expertise to ensure the same experience as windows or mac on this platform this often leads to longer development times and cost. | Windows could be considered the most supported platform, with the largest number of gamers. This should be considered as when it is often the most used client platform it becomes the cheapest to develop as it has the most support for the running client and well-known guidelines for implementation. | For a mobile environment there are two main platforms iOS and android. Each platform will have a different deployment entirely. The interface for other platforms would be built with mobile interface in mind. Development of a mobile app would be somewhat specific to the targeted OS, so development cost on this portion would be higher. You would need developers with expertise in this field which adds to the cost. |
| **Development Tools** | Developing on mac is by far the choice for most developers. You have access to all prominent IDE’s(Visual Studios, Eclipse etc..) including Visual Studios and eclipse. The ability to run duel operating systems is a plus so that you can do your development across multiple OS’s from the one OS. You can use java, JavaScript, and C languages along with many others to deploy on this system. | Often IDE’s used on this platform have to deal with bugs and issues longer than other platforms due to the transformable nature of Linux. If you are running a common up to date version of Linux Visual Studios and Eclipse become a viable option to use on Linux. As far as languages that are often used and deployed within this environment, java, JavaScript, and C languages are often used. | You can use almost any IDE with windows as its one of the most used platforms for personal use and work. MacOS is close second to windows. Visual Studios and Eclipse are IDE’s you can consider. Languages are Java, JavaScript, and C languages to list a few languages that are commonly used for deployment in this area. | As far as IDE’s that run on a mobile device there are none, IDE’s often used for deployment are Android Studio and Xcode. Android runs on Java or Kotlin as of lately. Apple runs with Objective-C and Swift code. Other languages can be used for deployment on mobile devices but you would need someone experiences in the mobile game world to achieve this. |

## Recommendations

1. **Operating Platform**:

* **Recommendation:** Linux Operating System (Ubuntu 20.04 LTS).
* **Rationale:** Ubuntu 20.04 LTS, a long-term support release of Linux, is recommended for its stability and extensive support. Leveraging the ext4 file system ensures efficient file management, while the Linux kernel (version 5.4) provides essential features for server-side operations. The use of systemd enhances process management and control.

1. **Operating Systems Architectures**:

* **Details of Chosen Architectures:** Microservices-Oriented Client-Server Architecture with Docker Containers.
* **Rationale:** Adopting Docker containers facilitates seamless deployment across diverse environments. Microservices, orchestrated with Kubernetes, enhance scalability and fault tolerance. Linux namespaces and cgroups ensure process isolation and resource allocation, contributing to the efficiency of the chosen architecture.

1. **Storage Management**:
   * **Recommendation:** Integration with Amazon S3 Object Storage.(Cloud Storage)
   * **Rationale:** Utilizing the AWS SDK for Java ensures native integration with Amazon S3. Leverage Linux kernel support for the FUSE (Filesystem in Userspace) framework to enhance compatibility. Employing the S3 Transfer Acceleration feature optimizes data transfer speeds, and the use of server-side encryption (SSE) adds an additional layer of data security.
2. **Memory Management**:
   * **Explanation of Techniques:** Optimize Virtual Memory and Java Garbage Collection (G1 Collector).
   * **Rationale:** Tuning Linux's virtual memory subsystem involves adjusting parameters like swappiness to optimize the use of RAM and swap space. Utilizing the G1 Garbage Collector in the OpenJ9 Java Virtual Machine (JVM) ensures efficient memory management. Implementing memory-mapped I/O for certain data structures further enhances performance.
3. **Distributed Systems and Networks**:
   * **Recommendation:** Microservices Communication via RESTful APIs (HTTP/2) with NGINX Load Balancing.
   * **Explanation:** Utilize NGINX as a reverse proxy for load balancing and as an HTTP/2 endpoint. Implement gRPC for inter-microservice communication where low-latency and efficient data serialization are crucial. Leverage Linux's netfilter framework to manage packet filtering and implement iptables rules for network security.
4. **Security**:
   * **Explanation of Security Measures:** SELinux Mandatory Access Controls, iptables Firewall Rules, and Cloud-Specific Security Measures (AWS Key Management Service).
   * **Rationale:** Configure SELinux to enforce mandatory access controls, restricting processes to the minimum required privileges. Employ iptables rules for granular control over network traffic. Implement AWS Key Management Service (KMS) for server-side encryption key management, ensuring data security in transit and at rest.