-

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 01/27/24 | Benjamin Cleary | Initial release |
| 1.1 | 02/10/24 | Benjamin Cleary | Update to evaluation |
| 1.2 | 02/24/24 | Benjamin Cleary | Update to recommendations |

**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 proposed mobile style game between multiple unique teams where the application will draw a picture over 30 seconds with one team guessing what the picture is as a clue and if they fail to correctly guess the opposing team is granted 15 seconds to guess what the clue is. This application will be required to support a minimum of two unique teams, with any number of unique players, to complete in a unique game instance. At any given time, there will be only one game stored in memory of the client system at any given time. Players must have the ability to access the application over a web-based environment across multiple different platforms. Additionally, players must be able to verify whether a team name or game name are already in use.

## Requirements

The specific requirements for the software are as follows:

* Games must have one or more teams in play
* Teams will have multiple players assigned to those teams
* Game and teams will have unique names for players to search and join
* One instance of the application is loaded into memory at any given time
* Must be accessible through a web-based system involving client side applications and server side hosts across different client operating systems communication over network protocols.

## [Design Constraints](#_2et92p0)

Major design constraints on the project are:

* Interoperability between disparate client operating systems and host server systems
* Network architecture design to allow for high availability and reliability
* Limiting application instances on client systems to one
* Maintaining unique game and team instances for player reference
* Securing server-side host code to maintain integrity of games and teams

## [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 describes the software architecture of the gaming room package. The Program Driver class initializes the application. As part of this operation, it initializes the Game Service class and also initializes the Singleton Tester to verify that only one instance of the Game service is running in memory at any given time. The Game Service then runs and can call instances of Games, Team, and Player objects as children of the Entity parent class. The Entity class provides for storing Id and name variables for each sub-class as well as the ability to get those values for comparison and identity check of the object instance referenced. The Game object holds a list of unique teams for any specific game and the add function allows for checking against that teams list to unsure that each team is unique and adding new instances of the team object. The Team class maintains a list of unique players assigned to that team object and has the function to add a new player object to that team object and checks for uniqueness of that player against the player list. The player object defines the player object with a unique name and id for reference.

**"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 based server systems are not optimized for operations outside of the Mac ecosystem. While implementation across the Mac ecosystem would function quite well their support to systems outside of them would require additional effort to implement very specific translations to support a non-Mac environment. While Mac does have options for enterprise support, they do offer support with their licenses, albeit the cost of licensing is greater up front that other providers. | Linux implementation of server-side operations would be the most cost effective and adaptable option for server side. While it is similar to the Unix standard that Mac is also a child of there is a robust library to support translation to Windows based system. Additionally, most mobile devices use a Linux based operating system already and would require minimal additional work to implement. While many versions of Linux are free, very few will have any enterprise support without a subscription as, for example, RHEL. | While Windows is the most adopted OS model and it’s API is very robust, there is still a large enough component of market share does not use this implementation. The library supporting translation to Linux and mobile users is robust but a Windows server-side will still have hurdles to crossing over to a Mac environment. Additionally, Microsoft has transitioned to a SaaS model which will incur some monthly subscription. | While mobile server-side implementations exist, their use case for a mobile game are not ideal. The limited capacity for networking and the continual management of those style of networks in regard to the proposed software will be cost prohibitive and not nearly robust in the metrics of reliability and availability. |
| **Client Side** | Mac OS systems will have similar cost and time to Windows to develop due to the very controlled and structured way that they are implemented across their domain. Expertise will be the biggest cost as there is not nearly as many people that develop in a Mac environment as compared to Windows, mobile, or even Linux. While there is some cross over from Linux as both are UNIX children there will still be some unique aspects to consider. | Linux based OS systems are quite varied and due to the nature of their custom ability, will provide the largest cost and time for development. While it is open source so getting an environment for test is relatively cheap, the possible interactions with various unique systems will be difficult to control for outside of base OS releases such as Fedora, Ubuntu, etc. | Windows OS has a very robust API and is the most adopted traditional computing platform. This robust environment will enable relatively fast development time and mid-range development cost. Expertise is widely available for Java implementations in Windows but will require extra personnel for ensuring security. | Mobile devices present the lowest cost to value proposition for the program because of the diverse and robust market for mobile development. Expertise that will be required to bring this to market focus on mobile architecture and data portability. This will also be the shortest window of development as the somewhat spartan design architecture removes some functions or streamlines them to enable faster implementation. |
| **Development Tools** | The biggest language used will be Java and with it some IDE to compile and edit code with be required. Eclipse is ideal as it is available on Mac, Windows, and Linux. It is also a powerful IDE and using one IDE will help with team communication and use. Xcode is a tool that will help with Mac specific development tools. | As stated with Mac, Java as a language and Eclipse as the IDE are the most cost-effective solution. Additional tools such as Git and Redis Desktop Manager can provide additional tools that are Linux specific and for the most part is open source and cheap to acquire. | As stated with Mac, Java as a language and Eclipse as the IDE are the most cost-effective solution. Microsoft Visual Studio is the premier development tool for Windows, is cheap to acquire, and can handle Window specific challenges and provides a wealth of tools to address a wealth of issues. | As stated with Mac, Java as a language and Eclipse as the IDE are the most cost-effective solution. An android specific IDE like Android Studio would be more effective for a pure Android system development and it also enjoys robust support from Google, is free, and has a very active community support as well. |

## 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**: Recommend a server-side Linux based architecture for core functions with Mac and Window specific gateway servers for platform specific communication and addressing. The Linux back end can be specifically tuned to handle a large number of individual threads and partition out and process large volumes of computations as far as the flow of the game code server-side. The Mac and Windows servers just outside the main core can be tuned to handle specific protocol and service handling and translations to support base code operations at the Linux core. Additionally, physical server acquisition can track platform user numbers so that there are not unused servers because a lack of user traffic. This also supports specific security risks that might have platform specific cross-domain issues that can be addressed between the edges and the core versus between the core and every user.
2. **Operating Systems Architectures**: While most environments provide multithreading and parallel processing, Linux distributions have too must leeway when it comes to handling this feature. With a proposed architecture of a Linux core inside of platform specific servers, the core can be tuned to effectively handle the pure computation load without having to assign resources for threads outside of running the program such as loading graphics, user authentication, etc. The edge servers can assume these loads and will be faster to implement as the tools and OS native protocols and services have already been implemented on the OS level reducing resources required on the server if it were consolidated onto one specific operating platform.
3. **Storage Management**: While all platforms use some form of network attached storage as well as native storage, most of the native storage requirements would come from the Linux based core. The read-write requirements of running a static program would be minimal as a lot of unique data would need to be loaded into memory once and if necessary, forked or cloned, reducing the amount of reading from slower storage providing faster access to the processor for runtime. For the edge platform servers, there is more necessity for a network-based storage solution as there is no guarantee that a user will connect to the same server every time and that user specific information will need to be available to support authentication, graphics, and much of the user specific interface data calls. Additionally, many of the resources for the game will be common among users and have platform specific formats and protocols for use and can be loaded as needed for the game without duplication across multiple servers when additional resources are implemented.
4. **Memory Management**: The Linux core servers can be tuned for efficient memory usage to support faster access time to CPU runtime through the use of paging and swap space form faster storage mediums such as solid-state drives and fast disk speed hard drives. This allows backend computation to handle higher volume at any given time. The edge servers will use paging blocks to efficiently manage memory usage and provide a standardized capacity monitoring for analytics and can maintain authentication and other resources loaded into memory to speed up other services required outside core program functions.
5. **Distributed Systems and Networks**: The network component of the project must be deliberate in its implementation. The user will have a client application loaded onto their device that will handle translation of user and profile imports into a standardized format which can then be passed through the network via TCP/IP protocols to the back-end platform specific servers. IP protocols are robust in their routing capabilities, such as OSPF, and can be configured to quickly route network traffic based on a number of requirements and limits. IT is also platform neutral and is standardized across operating platforms. The backend network between the core and edge servers is controllable by the company and must be designed to enable high bandwidth and low latency links between the individual servers to provide seamless processing to the end user. Things like redundant links, ports, and paths should be considered and implemented so that when something like a network interface card fails on a server there is another NIC or server available to assume the load.
6. **Security**: Security is a very serious consideration and is vital to protect user and company information. Having an account with a password to access the program is an obvious first step but not the last. Ensuring that connections between the user and servers are encrypted, using a sufficient encryption algorithm, after authentication and password data is stored hashed with a salt are basic security measures that can greatly reduce most unsophisticated attacks. Drive encryption on the back end and comparing hashed file values with expected when downloading and updating software is another crucial step. The security posture of the system should be appropriate to the value of data and no user data should be transmitted or stored plain text or unencrypted.