Operating Platforms Module 7

Joshua Williamson

Southern New Hampshire University

CS-230-11538-M01 Operating Platforms C-4

Brian Enochson

August 16th, 2024



Win, Lose, or Draw

# **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 | 07/19/2024 | Joshua Williamson | Executive Summary, Design Constraints, Domain Model |
| 1.1 | 08/01/2024 | Joshua Williamson | Evaluation |
| 1.2 | 08/16/2024 | Joshua Williamson | 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)

The Gaming Room would like to develop a web-based client that can be deployable on multiple platforms. The client “Draw It or Lose It” is currently on the Android Platform. The game consists of several teams, and they must guess the images that are produced. This client must be capable of supporting multiple teams that have several players on the team. The team and game names must be unique when created. One game instance must exist in the game’s memory at a time.

## 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)

* Must consist of web-based technologies.
* Ability to have multiple teams and names must be unique.
* Ability to have unique game names and to allow system check if the name is in use.
* One game instance must exist in the games memory at a time.
* Mobile users must be able to access the web page.
  + Possible call to detect if mobile client is being used to render games for mobile devices.
  + This will allow easier accessibility for mobile devices.
  + Porting to mobile devices will require

## [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)

Reviewing the UML design below. The Entity Class holds the common attributes and behaviors. The Entity class is a parent class to the Game, Team, and Player class. Based on the open arrow this shows the examples of an Inheritance. The classes Game, Game Service, Player, and Team share the same multiplicity. They contain zero or many objects. The Game Service class also holds the majority of the methods that make up the game. This will allow the ability to add multiple players to each team as well as adding multiple teams to the game.

There are also two separate classes, the ProgramDriver and the SingletonTester. The ProgramDriver has the main method and will call the SingletonTester to test the running code. The SingletonTester will also check to see if there already is a running instance of the game. This will meet the requirements requested by the client to check if the game is already running in the memory. This is a good example of Abstraction when testing the features of the program.

**"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 Servers are available through third party hosts. Purchasing your own hardware can be costly. Some of the downsides to using macOS would be its limited hardware configurations compared to other platforms. | Linux is open source and cheaper compared to Windows and macOS offerings. Due to this, there would be a lot more offerings and configuration options compared to macOS. | Most office environments run on Window based clients and a Windows server would be easier to maintain for those coming from years of using Windows. There would be a lot of configuration options however, licensing costs for the different Windows service may add up. | Mobile devices are usually lower powered. As servers are resource dependent hosting the software on a mobile device would not be beneficial. It is possible to host web services on Android devices. However, additional development and coding would be needed to port over other services to the Android platform. This would take time and cost which could be better suited for already existing platforms. |
| **Client Side** | To develop for macOS platforms you would need Xcode. As this is only found on macOS clients the developers would need to purchase macOS desktop clients.  As of February 2024 Windows has roughly 72% market share as macOS has 15% market share. Cost and development time may not be beneficial due to a lower market share.  However, if there is a demand from our userbase as of June 7th 2023, Apple has teamed up with Crossover to allow seamless porting over of Windows games to macOS. This may save time and development costs.  https://developer.apple.com/videos/play/wwdc2023/10123/ | Development would make use of Java, C, C++, or Proton.  However, due to low market share on Linux and the complexity of different Linux environments it may not be recommended to develop a client for Linux.  Due to the success of Values portable gaming device the Steam Deck development of Proton has made playing Windows games in Linux simple and easy to deploy. Verifying that our game can easily be playable through Proton would cut down the need to develop for Linux based clients.  https://www.protondb.com/ | Windows make use of C# and .NET which are both common in Windows games and applications. Both do not have a cost when developing on Windows.  As of February 2024 Windows has roughly a 72% market share. As the majority of users would be using Windows it would be required to develop a Windows client.  As Windows applications are mostly supported for older and current Windows revisions little support is needed for clients who may be using older Windows versions. | Mobile development would come down to two platforms. Android and iOS. Developing on mobile devices makes it easy to deploy for different screen sizes and different versions.  Android makes use of the Android SDK. This is Java based and can be developed on both Windows and Linux platforms. Android developer fee is 25$ one time charge. The low cost alone and the market share of Android would make it a good investment for developing the client for Android.  <https://support.google.com/googleplay/android-developer/answer/6112435?hl=en>  iOS makes use of Xcode and Swift. A macOS client is needed to develop. An iOS developer license is also needed which is 99$ per year and 299$ for enterprise-based development. The cost may be higher than Android. However, the market share for iOS would make it a requirement to develop an iOS app.  https://developer.apple.com/support/enrollment/ |
| **Development Tools** | macOS uses Objective-C and SWIFT for application development.  Xcode is the development environment and is listed as 99$ per year and 299$ for enterprise-based development. | Linux development can be done with C, C++, Python, or Java.  For Python, developers can use PyCharm.  For C, C++, and Java developers can use Eclipse.  All of the IDE options are free. | Windows development is mostly used with C# and .NET.  Developers can use Microsoft Visual Studio. This offers many tools and plugins for automation.  Visual studios have two prices that are subscription based. 45$ a month for professionals and 250$ a month for enterprise. | Mobile development would come down to two platforms. Android and iOS.  Android makes use of the Android SDK. This is Java based and can be developed on both Windows and Linux platforms. Android developer fee is 25$ one time charge.  <https://support.google.com/googleplay/android-developer/answer/6112435?hl=en>  iOS makes use of Xcode and Swift. A macOS client is needed to develop. An iOS developer license is also needed which is 99$ per year and 299$ for enterprise-based development.  https://developer.apple.com/support/enrollment/ |

## 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**: Recommendation is to use Linux for server hosting. This will reduce license cost compared to Windows or macOS. As Linux is open source it will offer many tools available as well as robust security solutions. The front end can be developed in different languages to suit the client and platform needs. **Examples: Java, .Net, SWIFT, C.**
2. **Operating Systems Architectures**: Given that "Draw It or Lose It" gameplay does not rely heavily on real-time reactions, low latency between the frontend and backend is not critical, allowing for asynchronous communication between them. Implementing a backend that makes use of VMs, and containers would enhance scalability. Cloud providers would be needed to deploy the game. The cloud platforms would offer proprietary tools that would make the setup and deployment process smoother.

Deploying a client-side of the game would shift the heavy lifting and processing tasks away from the server. This can reduce operation costs and data usage on the server. Client-side can also enhance gameplay by ensuring framerate remains consistent as the client can cache data and images ahead of the gameplay.

1. **Storage Management**: For the game client both HDD and SSD would meet the game’s needs. Recommending SSD for cache and rendering of the game. Standard HDDs offer larger capacity but slow read and write times. It is recommended to host larger files on the HDD. This will help offload rendering and loading and reduce the demand for server communication from a single client.

On the server-side SSDs will offer faster performance and hosting the VMs and or container on SSD would be the best solution. With the SSD this would offer faster read and write times as well as better reliability compared to standard HDDs. HDDs should be hosted on an alternate storage device in a RAID 5 or 6 configurations. Backups of the game environment on the server should be stored on a NAS. This is ideal in the event of a need to recover the game server in the event of a disaster recovery scenario.

1. **Memory Management**: In Android it would utilize paging and memory-mapping for memory management. As a result, any memory that an application modifies—whether through allocating new objects or interacting with mapped pages—remains resident in RAM and is not subject to paging out.

iOS uses manual memory management, requiring developers to retain and release objects explicitly. In later iOS revisions, memory management is now automated at compile time by Xcode, eliminating the need for manual intervention.

Windows, memory management is handled through a combination of demand paging and the page file system. Windows uses a page file on disk to store pages of memory that are not currently being used, allowing for more efficient use of physical RAM. The memory manager in Windows relies on utilizing the Least Recently Used algorithm to manage the paging process.

For macOS, memory management uses a combination of virtual memory and advanced caching mechanisms. macOS uses a memory compression technique, which allows inactive memory pages to be compressed and stored in RAM rather than being paged out to disk, thereby optimizing memory usage and performance.

Linux employs the concept of page cache for managing data stored in main memory, utilizing virtual memory to allocate pages. It uses demand paging, which optimizes memory usage by only loading pages into memory when they are actively needed. Page replacement is managed through the Least Recently Used algorithm.

For server-side operations, minimum RAM requirements can be expected with client-side rendering. In a modern environment that utilizes containers and other services, costs will scale with the number of users. On the client side, RAM usage should also remain minimal, as only one to two images need to be stored in memory at any given time, along with the RAM required to support the client application, such as a web browser.

1. **Distributed Systems and Networks**: Hosting the server onto multiple servers would be the best solution for optimal uptime and reliability. Hosting the servers on third party hosting services such as AWS would allow us to distribute the hosting across different regions. This would allow more responsive communication as our clients could connect to closer server nodes. This would also help reduce any widespread outages as clients can connect to another server node.

The server can communicate through RESTful APIs and this would allow seamless client/server communication to client front ends such as PC clients and mobile devices.

1. **Security**: It is recommended for security to have role-based authentication. Security will be managed through role-based authorization. This requires the development of an entitlements interface to enable efficient administration of roles and accounts.

The principle of least privilege should be applied, restricting users' access to specific game-related functions, such as game creation, team name creation, and team enrollment. If necessary, user permissions can be expanded within a team hierarchy to allow team captains or members to edit teams or add/remove players.

Passwords should be stored encrypted and not in plain text. This will help prevent any possible tampering in the event of a database breach. For network security a business-class firewall would be recommended. Restricting and locking down any ports that are not in use would prevent outside brute force attempts when scanning for open ports on the outside.