

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[CS 230 Project Software Design Template](#l6ti7uoag22u) 1

[Table of Contents](#j0zll) 2

[Document Revision History](#grjogdjh5fi8) 2

[Executive Summary](#sbfa50wo7nsh) 3

[Design Constraints](#et92p0) 3

[System Architecture View](#ilbxbyevv6b6) 3

[Domain Model](#h2ehzxfam4o) 3

[Evaluation](#o15spng8stw) 3

[Recommendations](#m8aleynsvzvc) 5

## [Document Revision History](#grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/24/22 | Nick Glidden | This is our first iteration of this document, and our primary focus is to edit all of the content found within the executive summary, and design constraint sections. |
| 2.0 | 12/4/22 | Nick Glidden | This is my second iteration of this document, and my primary focus here will be to build upon the concepts and ideas introduced with version 1.0, by completing the evaluation and recommendations sections. |
| 2.0 | 12/14/22 | Nick Glidden | This is the third iteration of this document, and includes finalized recommendations for development, touching on areas from Operating System Architectures, Security, and much more. |

**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)

Creative Technology Solutions (*CTS*) has set many goals for developing this project for The Gaming Room, which will be outlined below. From a high level, our focus will be to create a program that utilizes the web infrastructure, to be widely available, including all three major desktop operating platforms (macOS, Windows, and Linux) as well as the major mobile operating platforms (iOS and Android) as well. Additionally, within this program, we will seek to provide a secure, reliable, and optimized system to power their popular Game titled “Draw It or Lose It.” There are many features and functionalities that we will need to account for, but some of the most important are: ensuring there is not ever more than one game instance in memory, ensuring each game can have one or more teams, which will also contain one or more players. Additionally, the names of both games and teams, must be unique.

We will work hard to provide a strong solution, using the Agile Methodology to provide rapid development, strong communication, and multiple milestones throughout this project. Once our solution has been successfully developed and reviewed by our client, we will perform a rigorous testing process, and account for any weaknesses we are made aware of.

## [Design Constraints](#et92p0)

When developing a web-based program, that will be accessed and used by many different operating platforms, we will need to first and foremost, understand that our design will be broken into two. A server side, and a client side. Below, I will provide a summary of design constraints for both, but when considering them working together, our design will require a reliable, fast, secure connection that allows for both upload and download in parallel.

*Server Side*

The sever side of our application will be the largest piece of development. This is where the core of the game will be stored. Additionally, this central hub will need to be supplemented by a system to handle and maintain multiple connections at once. This supplementative system can be broken down into two parts. The first will be responsible for communicating via POST/GET API calls to each one of the clients. Because this game is highly time sensitive, a design constraint here will be maintaining a synchronized delivery and acceptance timeline across all the clients (players in the game). The second part of this system will be the brains behind the communicative arm, which will provide instructions, content, and responses to be communicated. Things like directions, when the game is starting, if an answer is correct or not, what the current leaderboard is, etc. will all need to be handled by this part.

*Client Side*

The client side of our application will be responsible for communicating with a single, dedicated port of our communicative arm located on the Draw It or Lose It server, as well as providing an incredible, consistent experience despite the operating platform being used. Many of our design constraints will be found here, because of the multi-operating platform support that’s required. Our code here will need to be lightweight, fast, and reliable. Additionally, because this part of the program will interface with the user, it needs to be extremely robust. Our core development here will need to be in a language such as Java, that is large enough to be supported across all platforms, but there may need to be supportive modules that can be utilized if needed for a specific OS. It’s important here that we do not include any OS-specific logic directly to the main program driver, to avoid unnecessary code from being loaded into memory.

## [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](#h2ehzxfam4o)

This UML Diagram is a great overview of how we will structure this program, and it uses many different OOP principles throughout. The program driver of course will be responsible for running the program, and will leverage a Singleton Class, to ensure not more than one Game Instance is loaded into memory at a time. Basically, only one game can be going at a time. This Game Service class will be responsible for “managing” the game and ensuring its administered properly. The game, team, and player classes are all responsible for defining the properties and methods of each respective object and will all contain everything that is found in the entity class.

In terms of object-oriented programming principles, inheritance is the first I would mention. All three of the object classes, including “*Game*”, “*Team*”, and “*Player*” all inherit from the Entity class. In layman’s terms, this makes a lot of sense, because these objects are entities. Additionally, another principle that is found in this UML diagram is aggregation. Aggregation can be spotted between the: game service class and the game class, the game class and the team class, and the team class and the player class. This is true because the player class will be used as an object within the team class, and likewise between the game class and team class, and game service class and the game class.

UML Diagrams, when fully understood, can convey quite a lot of information when done correctly. For me, this diagram is going to help me develop this project much faster.

**"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](#o15spng8stw)

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.

*\*I know this is not in the grid format, obviously. I made a judgement call to change it, because my answers are lengthy, and the formatting started to get really odd. Hopefully this isn’t a problem.\**

**Server Side**

**Mac**

The Mac operating system is not a popular candidate for server-side development. While it still exists, it primarily has benefits relating to environments that are 100% Apple based. It’s been noted that the Apple operating system is limited to smaller sized environments, and can’t be scaled quickly, easily, or inexpensively. The Mac operating system Is the least compatibly as a server-side OS, and the shallowest. When I say shallow, I am referring to the level of tools, flexibility, and power you are provided. Much of the tooling is represented via a UI, which is certainly user friendly, but extremely limiting. While there are a lot of reasons why Mac is not a great choice for server-side development, two reasons it would be a good choice are: (1) the server will interact with Apple devices exclusively, or (2) the server is being deployed to a small, fixed sized team, that has very basic, general needs.

**Windows**

The Windows operating system has been around for a long period of time and is often used by developers that need servers for various networking functionalities. Since Windows is maintained by Microsoft, the OS performs incredibly well for servers managed by teams, schools, and corporations. Microsoft has state of the art administration tooling, and the Windows operating system is tied deeply into those tools. A prime example of this, is the PowerShell language. A downside of the windows OS working so well for business customers, is its cost. You can expect to pay $50 - $150 when deploying Windows on your server, but those prices can jump quickly.

**Linux**

Again, Linux arguably the best choice here. The Linux OS has countless distributions that are built entirely with server developers in mind, by supporting the key infrastructure, optimizing performance, and maintaining strong security. Linus is widely recognized as one of the most secure and stable OSs’to exist, and because of that feature, is a shining candidate for powering and being used by servers. Additionally, due to the inherent free and open-source nature of Linux, developers can expect a very little to none cost throughout the OS. While much of this description has been advantages of Linux, there is one disadvantage to be aware of, and that is its user friendliness. Linux’s flexibility provides a lot of power, but complicates the UI and system as a whole, increasing the entry barrier and toughening the learning curve.

**Mobile**

Mobile devices, even more than the Mac operating system, are arguably never used to host web applications or provide server-side development. Mobile devices, their operating systems, and their hardware are all optimized to perform as a consuming, client-side device. Using one of these devices to provide hosting or development would certainly not be the best option, unless convince and portability were essential.

**Client Side**

**Mac**

The Mac operating system is Among the least compatible and should be expected to require a large amount of work to ensure things work properly. Especially now that Apple devices are split into two categories, intel-based, and apple silicon devices. Both devices have a different infrastructure, and often require two different approaches to interact with. Additionally, the Mac OS is a famously closed system, so interacting with a device as a “untrusted developer” may also be tricky. However, if you can support the macOS, chances are you will be able to support all apple devices, as iOS, watchOS, and tvOS are all directly supported.

**Windows**

The Windows operating system should be the easiest to work with. Windows is absolutely the most widely supported, and the most open to “outside” developers working with it. This is an obvious advantage for many reasons, but disadvantages include a lack of security. Due to the operating system having a more “open” architecture, virus’s, malware, and other malicious software is much more prevalent in the Windows OS, than anywhere else.

**Linux**

The Linux operating system will work well the largest range of servers and other pieces of hardware and will provide the user with much more customization than most other operating systems. It will be the most secure, as well as the least expensive, mostly costing nothing to work with.

**Mobile Applications**

Mobile applications (previously defined as Android and iOS), typically require a fair amount of work to support. This is because of the lack of hardware on the client side. While there are always hardware and spec improvements happening, mobile devices are still behind desktop devices in terms of capability. While this doesn’t necessarily apply to the Android OS, on an iOS system, file management, downloads, and installs are extremely difficult to interact with.

**Development Tools**

**Mac**

Mac is arguably the “friendliest” choice, due to its beautiful UI and attention to detail when it comes to the user experience. Mac has grown in popularity over the last decade, so compatibility issues are becoming less likely, but still something to be concerned about. Mac is considered a closed box system, and that often benefits developers that work entirely in the Apple ecosystem, but if you ever need to develop outside of it, you may run into challenges. A popular tool that is available for Mac developers is XCode, which is an apple created program, that streamlines development for software, especially for software that will exist in the Apple ecosystem.

**Windows**

Window is arguably the “most popular” choice, given the sheer size of Microsoft, and its wide support across many commercial products including Lenovo, Asus, HP, Dell, etc. The Windows operating system does a great job at utilizing its popularity, both as Windows, and as a product of Microsoft. The operating system is home to one of the most popular IDE’s, Visual Studio, and Visual Studio Code. That community is something not found elsewhere. Additionally, Windows is often the first language to make sure hardware is compatible with, so as a developer, compatibility issues are much less likely.

**Linux**

Linux is arguably the “best” choice, excluding user preference. Due to the distribution model that Linux is based on, developers often experience speed and performance enhancements. This unique model also is responsible for creating an unmatched level of flexibility, allowing a wide range of languages and tools to be available on the Linux operating system. Not only do these languages and tools exist, but by utilizing a specific Linux distribution, you can optimize and customize the Linux OS to fit your needs best. While all languages and tools can be considered supported, it is worth mentioning that the C family of languages has been thought to be the most popular across Linux developers. This can be explained because Ubuntu, one of the original Linux distributions, was written in C. Many of the other distributions that exist, also directly use, or are mirrored like C.

**Mobile Applications**

Mobile applications today typically consist of two operating systems, iOS (by Apple), and Android (by Google). There is worth noting that while it is not popular by any means, Microsoft does produce a mobile version of Windows as well. If you are a mobile developer, which operating system you are working on matters much more than any other type of software. For iOS devices, all mobile applications must be developed via macOS, approved by Apple, and published to its App Store. Alike, Windows devices require that their developers use the Windows operating system. Android developers, however, have more of a choice. Android Studio, the primary application for android developers, is available on all major operating systems. Like tooling, the languages that mobile developers are forced to use are much more to what operating system is being used. Most iOS applications are written in Swift or Objective-C, while most Android applications are written in Java.

## 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 would recommend the Linux Operating system, due to its strong sense of security and stability, and wide level of support through its unique distribution model. Leveraging specific distributions will allow for customized features and performance. Additionally, the cost of running Linux is free, so there would be a substantial, recurring cost savings compared to other platforms.
2. **Operating Systems Architectures**: Linux has an architecture that is comprised of three parts, the kernel, which is at its core and responsible for all the heavy lifting. Next, working outward, is the shell. The shell is a communicative layer and is responsible for providing commands to the kernel. The shell can be used directly, through a command window, or can be used to power applications behind the scenes. Applications matter of fact, are third, and the outer most part of the Linux operating system. Most of our work will be done via the shell.
3. **Storage Management**: I believe that a cloud storage management system would be the best, rather than an on-premises system, because it is so wildly available. For one, cloud storage is much cheaper. Secondly, cloud storage can be paired with other cloud computing resources, such as load balancers, edge locations, and dynamic (or elastic) scaling to increase performance, stability, and reliability. As for how to format that data within the cloud, I would suggest a relational database, such as RDS offered by AWS.
4. **Memory Management**: For this game, memory management is important due the large file size of images, and animation (the drawing part) files. In Linux, all memory is managed virtually. This is beneficial for us, because we can make quick and easy changes to our virtual memory, that will through a low-level language, quickly and efficiently translate into changes on real memory changes.
5. **Distributed Systems and Networks**: Since Draw it or Lose it wants to have compatibility across all operating systems and mobile devices, our network will need to be extra robust. Mobile devices are unique because they operate sometimes with poor connectivity, and that can seriously damage our system, especially in a collaborative environment. I believe it’s important to use popular network techniques to maintain a durable connection, that can overcome connectivity issues by reconnecting itself seamlessly. Additionally, speed is important, given that the game will be played in real-time. Using a proper CDN network, and load balancing software, a “real-time” experience should be made possible. Additionally, we could use governing, and dynamic rules to ensure that all connections are occurring at the same speed, by limiting all the connections, to whatever the slowest connection is supporting.
6. **Security**: A strong reason Linux was the recommended operating platform was its increased level of security compared to Windows or Mac. On top of the Linux operating system, I believe it’s important to encrypt all data, both at rest and in transit. This will ensure all users are kept private, from both malicious actors, as well as our internal systems and teams.