

The Gaming Room

# **CS 230 Project Software Design Template**

Version 2.0

## 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 |
| --- | --- | --- | --- |
| 2.0 | March 29th | Samuel Bailey |  |

**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 software that has been designed is beneficial to web-based platforms. This is great for anyone playing the game on a website! Otherwise, the game hardly works or in some cases not at all. The design of the game is clean and can work well in a simple case. The problem is not all cases are simple, this is a tricky subject considering there is not one specific thing that can be done to fix this scenario. Unfortunately, the application will simply have to be rebuilt per platform of use.

## [Design Constraints](#_2et92p0)

When building applications for different platforms each company of the platforms likes things to be done in a certain way. If not don’t this specific way, then the application is losing out on features of all kinds as well as how smooth it is with the platform it is on. For example, if we build a website-based game it won’t run on consoles/phones/tablets. This is a problem if you want the game to work everywhere. In some cases, it just has to be completely remade.

## [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)

Although looking at something like this diagram below can look intimidating it is actually easy to understand. Let me explain, starting from the top left with programDriver pointing to singletonTester. PrgramDriver is the main/core of the program, the part that tells the rest of the program so it will work. The singletonTester is the test cases to ensure the program works. GameService, Game, Team, & Player are all parts of Entity. They inherit all that Entity offers. Everything inside of the boxes is method/variables.

****

## [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** | Mac’s are notorious for simply working. From a sever side though, mac doesn’t offer servers, but they would integrate well with almost any server. | Linux is simply known for running servers. Some of the most efficient running servers in the world are on linux. Linux servers is the stuff that just works and no one knows. | Windows like all the others offers so many types of servers, you need a class just to learn about them all. Mostly efficient and are commonly used for small-medium businesses. | Mobile devices don’t host servers. Even if you can bootleg it, it’s not a good idea. |
| **Client Side** | Once an application is built on xCode it can go live to any client anywhere. The only cost is paying for it to be released on the store for others to download. | Linux is difficult for clients to use since they keep sysAdmins in mind when creating linux. It’s uncommon for people to even have a linux in the first place. | Windows owns the business client side of the world. Although the younger generations have drifted away due to its unappealing UI/UX design. | The great thing about mobile devices is everyone has them and you could reach the largest audience if released on a phone. |
| **Development Tools** | Mac’s are almost forced to use Xcode for all mac based applications. | For linux you need to know shell and be able to create batch files. This is hard to find in devs since it’s hardly used. Any IDE would work. | Windows is the largest dev platform in the world. You can use almost any language to create the game and can also use any IDE on this platform. | Mobile phones simply do not use IDE’s. However making the phone would require either xCode or android studio depending on the platform. |

## 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**: In my professional opinion I think the game should be made on Window computers first to test its success. Windows owns 76% of computer market share. This is a huge market for the application to grow into and if successful in this market will have a great foundation to outreach to other platforms.
2. **Operating Systems Architectures**: Using the windows operating system there is a few different architectures that are in operation. Mainly for windows they use 64 bit architecture. For chip architectures it depends on the type of chipset that the computer is using two of the main architectures are x86 and ARM/RISC.
3. **Storage Management**: For a windows-based application I would recommend using a SQL type database. Being a small startup, I would recommend using the cloud like GCP to use a cloud instance.
4. **Memory Management**: The bulk of the game will be downloaded onto the hard drive of the computer that being SSD or a traditional hard drive. However, when the game is started up it will pull fast use data into the RAM for much faster load times to increase user experience.
5. **Distributed Systems and Networks**: For working through different platforms the user accounts for the game can be stored in the SQL cloud this can be used for linking onto different machines as well as other platforms down the road. Obviously, there is a dependency for the cloud at this point to load the user data but GCP has a promised uptime of %99.99. Connectivity to the system is worldwide since google has datacenters across the world.
6. **Security**: For security purposes I would recommend creating in house API’s and not using any open source. From the datacenter perspective there is limited control since we don’t actually own the datacenters but rent them. Lots of the security is going to depend on the engineers and developers that create the actual game. For using the user data in between platforms and machines is called logging. The engineers can either use other company API’s or create their own for in house accounts, most companies use both.