

Win, Lose or Draw

# **CS 230 Project Software Design Template**

Version 1.0

## 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 |
| --- | --- | --- | --- |
| 1.2 | 10/18/20 | Zachary Kuran | Filled in Recommendations sections on pages 6 & 7. |

**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)

We have been asked by the staff of the Gaming Room to prepare a software design document and begin developing the game application, addressing their software requirements. In terms of requirements, they require the game to have the ability to have one or more teams involved, each team will have multiple players assigned to it, game and team names must be unique to allow users to check whether a name is in use when choosing a team name, and only one instance of a the game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

In terms of design constraints, first and foremost, it will need to be accessible to all users present and wanting to play the game. This requires us to set up inter-user compatibility in order users and teams to play against and interact with one another, and even multiple others at a single time. We also need to implement code that would forbid other teams from picking team names already in use. Furthermore, we cannot allow for multiple games to be happening simultaneously, instead using unique identifiers to each instance of a game, team or player.

## [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)

Below we have a UML class diagram that depicts the flow of the code for the game. First and foremost, all of the code resides under the package “com.gamingroom”. This packaging allows for all of the data below to be associated with one another visually as well as allows for further control of classes and data access as well as allows for ‘package methods’ which we do not use for this particular code. Next we have the class to our main() method, ProgramDriver, which is the ignition for our program, accessing class GameService when necessary and calling the testSingleton() method of class SingletonTester for the purposes of making sure that there is only one instance of a game in memory at any given time. The arrow depicting that relationship between ProgramDriver and SingletonTester is called a Directed Association, and in our case, the label “<<uses>>” is indicating that ProgramDriver uses the class SingletonTester during the course of its functioning. Next we have class GameService, which is like the instruction manual for our program. In the top box is all of our attributes of the class, all of them denoted with a “-“ to indicate that they are private and cannot be accessed without the use of specified functions from the class itself. Beneath that is the next box, filled with methods. The “+” next to most of these denote that they are public accessible in the program. Connected to the GameService class, we see the Game class. These are connected with a single line called an association which is to indicate that there is some kind of a relationship between them. In this instance, the 0…\* label on the line denotes that for every object of class GameService, there can be anywhere between 0 and infinite objects of Game. This relationship continued from Game to Team respectively as well as Team to Player respectively. Finally, the last arrow, the filled in one pointing from the Game, Team, and Player classes to the Entity class is called Inheritance and indicates that Game, Team, and Player are all subclasses of the Entity class, and therefore inherit all of the attributes and methods of the parent class, which in this case is Entity.

****

## 

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is adaptable due to its innately flexible terminal commands. This would make configuration of the server, access to it, or changes of it easily approachable. Mac is also terrific for operating multiple OS’s in conjunction at once, in case we wanted to implement multiple languages. | Linux is very similar in its advantages to Mac on Server Side, except it is also much more financially accessible. | Windows is the most widely used OS, and so has a ton of software and support available to it compared to other OS's. Windows also has the largest variety of programming libraries accessible to it, allowing for complex and unique code implementations. | In short, the server side should be localized, and hosting it on a mobile device could introduce plenty of complications. A mobile device is not the correct choice for server side. |
| **Client Side** | Mac is a fairly popular OS, and as such, there is only a moderate amount of expertise and time required for implementation on client side. It’s cost would be very middle of the road and comparable to Windows. | Linux is a much more niche OS, so the amount of expertise and time required for implementation would be maximal. Luckily, it would be minimal in cost to make up for it. | Windows is the most popular OS and so would require the least amount of expertise and time to implement. The cost would be comparable to mac. | Mobile Devices would require a moderate amount of time and expertise and widely vary in cost. |
| **Development Tools** | The most prevalent programming languages for Mac would be JavaScript, SQL, and Java. Some IDEs that are useful for building this software for Linux would be XCode, Eclipse, and Visual Studio. | The most prevalent programming languages for Linux would be Python, C++, and JavaScript. Some IDEs that are useful for building this software for Linux would be Sublime, Atom, and Eclipse. | The most prevalent programming languages for Windows would be Python, JavaScript, and Java. Some IDEs that are useful for building this software for Windows would be Visual Studio, IntilliJ, and Eclipse. | The most prevalent programming languages for Mobile Devices would be Python, JavaScript, and Ruby. Some tools that are useful for building this software for Mobile Devices would be Xamarin, Xcode, and Android Studio. |

## Recommendations

1. **Operating Platform**: Considering the fact that we will inevitably have to operate in multiple programming languages due to the fact that we plan to expand into multiple computing environments, I would recommend the Mac OS as our primary operating platform. This is because Mac OS is unparalleled in terms of running and integrating multiple separate operating systems on a single given system.
2. **Operating Systems Architectures**: At the heart of the Mac OS architecture lies its core OS, which is base a Unix core, called Darwin. The Mac OS graphics subsystem is made of three parts: Quartz, OpenGL, and QuickTime. For its application layer, it uses Classic, Carbon, Cocoa, and Java. Lastly, for its user interface it runs Aqua. (“*Mac OS X Architecture and Terminology :: Chapter 1. Mac OS”)*
3. **Storage Management**: For this particular program, I would recommend a cloud storage, in particular a Unix based cloud storage as our main storage management system. This would be useful for multiple reasons. Firstly, hardware based storage is more expensive that cloud system storage, and similarly, the amount of storage available to our application would be far easier to adjust through a cloud based system. Also, considering that we need would require access to the internet for application users to link up and compete in a game, requiring the user to have internet access to have access to the cloud would be a natural fit, really provoking no downsides to the implementation of this storage system. Additionally, I would recommend Unix based storage strictly because we are already using Mac OS to build our application. And finally, if we ever wanted to update the system to allow users access to say different pictures in the future, it would be faster to rollout updates universally on a cloud based system (“Linux Cloud Hosting vs. Windows Cloud Hosting - The Best Server Cloud?”)
4. **Memory Management**: Mac OS will be using a combination of Virtual Memory, a process in which an operating system overcomes the physical limitations of a devices RAM through the implementation of chunks of memory called pages, and something called the Kernel, which maintains the list of pages available to a system to ultimately store, call, and organize memory for the application in an efficient manner.
5. **Distributed Systems and Networks**: The distributed system that we will be using to implement the necessity for our application to communicate between various platforms will be something called a Client-Server system. Through this system, “clients contact the server for data, then format it and display it to the end-user. The end-user can also make a change from the client-side and commit it back to the server to make it permanent.” In this way, there are multiple benefits, firstly being that by the client both sending inputs to the server for the server to verify and confirm sent information, it must also translate sent information, thereby allowing multiple platforms to communicate with the server and be extent with each other when the server sends back data to the client to make permanent. Next, by implementing multiple connecting nodes that speak to one another, there will be less of a chance for the system to fail due to the fact that one node may crash but other nodes may be able to make up for the work load that was dropped by said node. Finally, it becomes must easier to breakup workloads between and rollout changes to differing nodes (“Please Enable Cookies”).
6. **Security**: An additional benefit of cloud based storage is that all of the information inputted by our users will not be stored locally, but instead will be stored on a server that we have access to. This is good news, as we have direct control over what information can go into and get out of said server, thereby allowing us to add any additional security measurements we see fit to our application. Additionally, through our Unix cloud storage system, we have multiple avenues of information security available to us, such as SSH, Secure Copy, SFTP, and rsync (Ruostemaa).

*Mac OS X Architecture and Terminology :: Chapter 1. Mac OS ...* etutorials.org/Mac+OS/using+mac+os+x+v10.3+panther/Part+I+Mac+OS+X+Exploring+the+Core/Chapter+1.+Mac+OS+X+Foundations/Mac+OS+X+Architecture+and+Terminology/.

“Please Enable Cookies.” *StackPath*, blog.stackpath.com/distributed-system/.

staff, Janne Ruostemaa, et al. *How to Secure Your Linux Cloud Server*. 24 July 2020, upcloud.com/community/tutorials/secure-linux-cloud-server/.

Team, Editorial. “Linux Cloud Hosting vs. Windows Cloud Hosting - The Best Server Cloud?” *Atlantic.Net*, Atlantic.Net, 28 July 2020, www.atlantic.net/vps-hosting/linux-vs-windows-cloud-hosting/.

*Mac OS X Architecture and Terminology :: Chapter 1. Mac OS ...* etutorials.org/Mac+OS/using+mac+os+x+v10.3+panther/Part+I+Mac+OS+X+Exploring+the+Core/Chapter+1.+Mac+OS+X+Foundations/Mac+OS+X+Architecture+and+Terminology/.