

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | <mm/dd/yy> | <Your-Name> | <Brief description of changes in this revision> |
| 2.0 | 05/31/23 | JJ Greenwell | Updated the evaluation section for Project 2. |
| 3.0 | 06/15/23 | JJ Greenwell | Updated the Recommendations section for Project 3. |

**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 client wishes to design a web-based game that works on all popular operating systems. They do not know how to design this on their own, so they wish to have our team make it for them.

## Requirements

* *The game must have one or more teams.*
* *The game must assign multiple players to each team.*
* *The game must be able to check if a user or team name is unique, so no names are used more than once.*
* *Only one instance of the game can exist in the memory at one time.*

## [Design Constraints](#_2et92p0)

The first design constraint is game functionality: The game must function following the given rules of the game. This means that each game must have at least one team, more if possible, with multiple players assigned to each given team. We must also make sure that no two players have the same name, and that no two teams have the same name.

The second design constraint is that the inner workings of the game must work according to the client’s request, which is that no more than one instance of Game exists in the memory at a time. A singleton package will be necessary for this.

## [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 UML shows us that there are three different types of Entities: Game, Team, and Player. These entities each also use the GameService class. And given the 0 – many relationships between GameService and the other entities, we can see that there must be at least one game, then each game can have multiple teams, and each team can have multiple players. The diagram also shows that the game has a ProgramDriver that runs main. The ProgramDriver class also uses a singleton class, which we know is a design constraint for the project.

**"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** | To use Mac on the server side, that would require we develop the project with Macs, which can tend to cost more, and the licensing would be pricey. | Linux is completely free, and it has a lot of useful tools for hosting a server. It does require a bit of extra knowledge to operate, but is not all that much more complicated. | Windows also has a lot of useful tools for server hosting, but the licensing for it can be expensive. Fortunately, even with the expensive licensing, it is not limited to only Microsoft products. | Mobile devices really do not have enough power to be effective servers for an entire website, so it would be best to use one of the other three options. |
| **Client Side** | Mac is very popular and user-friendly, but development would require Macs be used, which will require extra time and money. It does have decent security measures on hand. | Linux is slightly less used than Mac or Windows, and would also be a bit more complicated to design. However, there would be no licensing costs. | Windows is also very popular, and has good built-in security measures. | Mobile devices are popular because of their convenience, but would require a lot of extra development time due to how different they are from PCs. |
| **Development Tools** | Macbooks can use practically any programming language. Mac also has its own integrated development tool, Xcode. | With Linux it depends on which distribution you use, but most have Python installed beforehand. From there, there are a large number of other development tools that can be used. | Like with Mac, Windows can use practically any programming language. Windows does not have its own integrated tools, but it has Visual Studio, which comes from Microsoft. It can also use a wide variety of other IDEs for various languages. | Most mobile apps are not developed on mobile devices, but there are tools on PCs such as Unity that can develop with apps in mind. |

## 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 Linux be used for the server, as it would have less licensing costs, as well as more openness with other parts of the program, such as security.
2. **Operating Systems Architectures**: You will first need to select a Linux distribution. Each has its pros and cons, but most can be used to achieve similar outputs. It would be well to be prepared that since Linux is very bare bones, extra time and effort may need to be put into developing things such as security. Fortunately, there is a lot of room to develop effective security measures, and while it may take more time and knowledge, I feel it is worth it considering it does lower the overall cost of the project.
3. **Storage Management**: Solid State Drives are slightly more expensive than Hard Disc Drives, but since each has its own processor, its processing speed is significantly improved and easier to manage. Assuming the Operating System is used effectively and optimized well, the program should run very smoothly.
4. **Memory Management**: One general memory management technique is to monitor server traffic and reduce required memory at low traffic times. This could be done by lowering the required memory at times that are statistically low traffic, or by monitoring the servers manually. Beyond that, Linux has an entire subsystem dedicated to memory management, appropriately named “Linux Memory Management”.
5. **Distributed Systems and Networks**: First, the game should be developed for as many platforms as possible. In order to do this, we will have to develop different versions of the project on each respective platform and give each platform its own respective servers. This may seem time-consuming, but it will improve the user experience. It would also be useful to have cross-platform servers, so different platforms can play together. Plus, user information and progress should be stored on either our servers or the cloud, so they don’t risk losing progress should they experience an issue with the program.
6. **Security**: There are plenty of security tools for any operating system that are good to make use of. Some examples would be tools in the code such as trip wires to prevent the code being altered or taken by unauthorized users. Another simple, but very effective measure would be to make sure the code is developed with best practices, as having too many lines of code presents more code that can be manipulated and cause issues. Another simple yet effective measure when it comes to security is to have people manually monitor the program as much as possible, so that they can be quick to catch or prevent attacks that are not picked up by the program’s security measures. It would be wise to read through the security features in each Linux distribution as well when deciding which to use, as some may have more effective built-in security measures than others.