

# **CS 230 Project Software Design Template**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 10/18/2021 | Brandon | Software design constraints  Operating platforms |

## [Executive Summary](#_sbfa50wo7nsh)

The client wants to create a game that has multiple teams where multiple players can be assigned. The game and team’s names must be unique, and only one game of that type can be running at a time.

## [Design Constraints](#_2et92p0)

Certain components must be unique only allowing one game, team or the same player allowed in a game at a time. The game must have a way of preventing duplicate games or teams from being created. There must be a way to assign players to teams. The game should allow one or more teams to play in a game.

## [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 program driver calls the functions that create the unique classes. Using the SingletonTester class, ProgramDriver identifies whether the instance is unique. Several classes inherit from the Entity class. The GameService, Game, Team, and Player classes are all related zero to many.

Id and name are used to identify the unique instances of the game, team, and players. Inheriting classes use mutators from the entity class to create new objects. The objects created are stored in their respective data structure and each is unique to the instance.

**"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** | Secure, not as widely used. | Open source very old, reliable functionality. Highly customizable. | Easy to understand GUI, rapidly updated and supported. | Not recommended for larger projects, mobile convenient, new. |
| **Client Side** | Reliable platform. Can be very expensive. May have slower turn around time as it is not as widely used. | Many open source options to choose from. Inexpensive platform. Can be secure and requires infrequent maintenance. Has a learning curve | Easier to use user interface allows less experienced admins to maintain systems. Licensing for software can be expensive. May not be compatible with programs. | Less space needed.  Linux based applications means similar set up time and lower cost from open source software.  Can be ran from most modern mobile devices. |
| **Development Tools** | macOS Server, apache, WordPress, XAMPP, Xcode | Ubuntu, Apache, RedHat, Caddy, Netbeans. | Microsoft visual studios, windows server, asp.net, SharePoint | AIDE, glassfish, apache, jetty |

## 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 windows or Linux as the operating platform depending on what plays nicely with their other systems. Linux is the less expensive option and has a vast amount of customization and can work with almost any other technology server side. Cloud computing would allow a small company to distribute their product to almost anyone with a stable internet connection.
2. **Operating Systems Architectures**: Linux would integrate with both windows or Linux/Apache servers.
3. **Storage Management**: cloud storage, would allow us to back up data when not readily needed.
4. **Memory Management**: Linux is excellent at memory handling and is lightweight by design. Cloud computing would allow for smaller systems with less memory of their own to have access to our game.
5. **Distributed Systems and Networks**: We are using a web-based platform allowing multiple devices to access it at a time wirelessly. The majority of the computing would happen on the server allowing less power devices to communicate expanding the usability of the game. The user would download the client that connects them to the server that allows us to identify the player and team they are associated with.
6. **Security**: The client and the server should have separate protections. The application will be encrypted so no personal information from either party is unintentionally leaked. The application should require a login, and 2 factor authentication. WSS utilizing XML can be used for encryption.