

Ryan Korper

# **CS 230 Project Software Design Template**

Version 2.0

## 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 | <7/16/23> | Ryan korper | project |
| 2.0 | 7/30/23 | Ryan Korper | Project 2 |
| 3.0 | 8/13/23 | Ryan korper | 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 company, The Gaming Room, would like to create a game that can run on many web based platforms. The games name is draw it or lose it which is on android. The game will allow teams to face off against each other to compete and win. The game will consist of 4 rounds lasting a single minute, drawings are rendered at a steady rate until fully complete at 30 seconds. The team needs to guess the image and if they fail to guess before a minute, the other teams have 15 seconds to answer them selves.

## Requirements

*The requirements include: one or more teams, each team including more than one person, game and teams being able to be named, only one instance should be active, and runs on multiple platforms.*

## [Design Constraints](#_2et92p0)

Biggest design constraints are going to be the ability to get it to run on multiple platforms. We will be looking at recoding or restructuring code on other platforms to get it to run smoothly. While not the biggest issue. Getting everything to properly communicate will be important.

## [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)

Entity being created creates a relationship between the three classes: Game, team and player. They will all get information of the class entity. The UML diagram shows us with the open arrow form game team and player leading into entity. Further more we can see that game service starts its own line with the class. It calls on game for most information, game refers to team for information, team refers to player, and it all is to make sure proper information is being showed in the program.

**"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** | Apple has advantages in the ability of being fairly cheap to operate and get a server.  The server being cheap helps to keep this option being really nice. As server up keep can be very expensive over time. | Linux will offer some of the best custom server options in being very free form of an OS, however this makes the cost potentially more to upkeep.  Despite having custom server options, the ability to keep it up will end up costing more when more people potentially try to bring it up | Windows is streamlined to keep user upkeep to a maximum. This is backed by it being the most used OS, however this will affect us with being more susceptible to attacks on the server as this OS is more common.  The ability to have a streamlined experience makes managing the server really easy for us, this might be the best way to keep the most important server related items but if it cant relate to other platforms, it wont change much. | As almost everyone has a phone, these servers are easy to come by as they is plenty of room to work with. This means that it will save a lot of money.  Phone servers in general are cheap, it is why we see so many games pop up on phones. The server side up keep is easy to maintain. |
| **Client Side** | You need some knowledge to use apple devices. Cost is similar to windows. The OS will work well with mobile devices as apple os for mac and iphone are similar.  This client side will keep you similar to windows in cost, which means that windows and mac users are going to be the easiest to up keep. Mac and windows are some of the most used systems, in general, so the use will be sort of streamlined. Oddly enough there are emulators for apple on widnows you can use, so there is some over lapping you can see. It is emulated but the coding and functions are still the same for the emulator, not the machine that is emulating itself. | Need to very very fluid with linux to use. Cheap but a lot of time to learn and master to keep to maximum efficiency.  Linux having a large uphill learning curve makes it not the easiest to use, but allows the most freedom to allow users to have a very in touch server/ client side upkeep. | Cheap and common. Needs almost no expert knowledge as most people use windows devices everyday.  The client side for windows is like riding a bike. Maybe need some help at the beginning but once you know what you are doing, you basically know everything without getting really deep. And windows allows for so many custom user settings that if you don’t want to have ease of use, you can start to branch into more indepth user systems. | Provides the ability to have open source and easily patched in short time. Apple os will go with Mac obviously, and the game is already on android.  The game already being on android means that we already have functioning code. Im not sure if there is some type of “code translator” but you can always repurpose code from one to other device. |
| **Development Tools** | We can use swift to code on mac, we will use html, CSS, and javascript, with supporting tools using java, python, and ruby.  Most of the tools we use in general will be able to transport code across all platforms. Whether you just email your code to yourself or maybe use some cloud service. | Visual studio, eclipse, and notepad ++ can be used, allowing us access to almost all coding languages and html, css, javascript being more common than python, java, c++, c# etc.  Linux, simply put, will probably have access to almost all forms of IDE and code, maybe not including what is specific to apple, but we know that you can code freely for whatever you want. | Just like linux, we can use visual studio, eclipse, and notepad++ with access to all the same coding languages. A little easier to use with more general purpose as a focus.  Windows has access to almost any tool on the market, which includes coding apple products, however you just cant publish with out certain tools, Xcode being a prime example. | Android and swift will be our common grounds here. It will allow us to code in html, css, and javascript to create these apps that we are looking at.  Just like apple, we have the general tools we will use. As stated above, swift will be used. |

## 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 think windows is the best place to go as it is fairly easy to use and is fairly cheap. Also there is a lot of options on what language and IDE you can use so transferring code will be easy and require less work stations.
2. **Operating Systems Architectures**: I think they have access to the best applications to show system resources and more. They will show the proper services for users or servers. Further more this is in relation to windows being the most common OS. The IDEs will work on windows with ease, and other than directly uploading the game to Apple OS’s, they can essentially run everything from windows.
3. **Storage Management**: Windows is streamlined to help you manage your files and keep track of it all. This will help limit the storage and maximize the maintenance of the server and data. The cloud will also be able to be used for the built in storage system for files to be created and tracked properly so accidents wont happen. Windows offers a storage management by itself but if you do not want to use it you have options like SUSE Enterprise storage.
4. **Memory Management**: as stated above, the streamlined data management will help keep the databases created in the game to properly keep track of itself. This will keep outside sources from messing with your code or game and keep it clean overall. Windows basically uses pages with processes demanding it as necessary for what is processed before other processes.
5. **Distributed Systems and Networks**: Since windows is a place that has a very widely used OS, this means that working on multiple code is not impossible, as you can look at ways to have code almost be translated between Oss. This created even more of a reason to keep the OS as windows as it will keep a streamlined way to keep code all in one place. In reference to having connectivity, there are LAN options for workplaces, and outages can always be helped with having generators in the building to keep power.
6. **Security**: windows also has a lot of options for security, with even having built in security that is able to keep out a lot of common threats on the internet. This gives us not only a homefield advantage, but this also means that upgrading is easy. Windows is always going to have streamlined ways to keep the user happy. If the base security does not work, you can always use multilayer protection in the sense of multiple virus detectors, and to avoid directly using on the computer, you can create a usb that has the boot drives for these antivirus software.