

*Draw It or Lose It*

# **CS 230 Project Software Design Documentation**

Version 3.0

## 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 and Design Constraints 3**](#_Toc115077321)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

**Evaluation 5**

**Recommendations 9**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/12/23 | Jason Barry | Completed the Executive Summary, Development Constraints, and Domain Model. |
| 2.0 | 11/23/23 | Jason Barry | Completed the Evaluations Sections (with Works Cited) |
| 3.0 | 12/08/23 | Jason Barry | Completed Recommendations Section (and updated Works Cited). Requirements and Design constraints were combined due to essentially being the same thing. Also updated the Table of Contents to match correct page numbers. |

## [Executive Summary](#_sbfa50wo7nsh)

Our client, *The Gaming Room,* seeks to expand their customer base by developing a web-based version of their game, *Draw It or Lose It,* which is currently only available on Android. The developed application should be available on multiple operating systems (*Windows, Mac, Linux)* and should allow multiple teams consisting of several players to go through four minute-long rounds. Each round should begin by selecting a photo from a library of photos. One team then attempts to guess the puzzle based on the drawn photo until the clock rounds out. If the team fails to correctly guess the puzzle, the opposing team has 15 seconds to make a guess of their own.

## Requirements and [Design Constraints](#_2et92p0)

1. The developed application should have the ability to have one or more teams involved.
2. Each team should possess the ability to have multiple players assigned to them.
3. The developed application should ensure that game and team names are unique which should allow users to check whether a name is in use when choosing a team name.
4. The developed application should ensure that only one instance of the game exists in memory at any given time.
5. The developed application should be available on multiple web-based platforms including *Windows, Mac,* and *Linux.*

The ability to have one or more teams of multiple players is necessary because *Draw It or Lose It* is a team-based game. Failing to meet these parameters will confuse customers who have already experienced the Android application while also ruining player enjoyment. Ensuring that game and team names are unique should make the game experience more user-friendly as users would be unable to unintentionally confuse themselves. To further make the application more user-friendly, we should ensure that only one instance of the game exists in memory at a time. By accomplishing this, users will be unable to accidentally experience multiple games which should prevent unnecessary defects. Lastly, by ensuring the developed application is available on multiple web-based platforms, *Draw It or Lose It* should be able to reach the largest possible customer base.

## [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 *Entity* class includes the protected variables *long id* and *String name.* Meanwhile, it includes the public default constructors *Entity()* and *Entity(long id, String name).* The class also includes the public methods, *long getId()* and *String getName()* which are getter methods*.* Lastly, it includes to public *String toString()* method which acts as an override.

The *Player* class inherits from the *Entity* class and declares its own *long id* and *String name* variables. The class also includes the public default constructors *Player(long id, String name)* and *Player(long id, String name, Team team).* Another public method known as *long getId()* is also created alongside the public *String toString()* method which acts as an override. The *long getId()* acts as a getter method.

The *Team* class inherits from the *Entity* class and declares its own *long id* and *String name* variables. The class begins by creating a private array list of Players which is derived from the *Player* class. Following this creation, the class declares the public default constructor *Team(long id, String name).* Next, the public methods *long getId()* and *String getName()* are declared as getter methods. After this, the public method *Player addPlayer(String name)* is declared. This method accesses the *Player* class due to the *Player* declaration and is used to check if a player exists in the aforementioned array list of Players. This works because *Team* has a cardinality of 0 or more Players (which means *Team* has access to the *Player* class). If the player doesn’t exist, they are added to the array list. Lastly, the public *String toString()* method is declared which acts as an override.

The *Game* class inherits from the *Entity* class and declares its own *long id* and *String name* variables. The class begins by creating a private array list of Teams which is derived from the *Team* class. Following this creation, the class declares the public default constructors *Game()* and *Game(long id, String name).* Next, the public methods *long getId()* and *String getName()* are declared as getter methods. After this, the public method *Team addTeam(String name)* is declared. This method accesses the *Team* class due to the *Team* declaration and is used to check if a team exists in the aforementioned array list of Teams. If the team doesn’t exist, they are added to the array list. This works because *Game* has a cardinality of 0 or more Teams (which means *Game* has access to the *Team* class). Lastly, the public *String toString()* method is declared which acts as an override.

The *GameService* class begins by creating a private array list of Games which is derived from the *Game* class. Following this, the private variables *long nextGameId, long nextPlayerId,* and *nextTeamId* are declared and initialized with the long integer, 1. After this, a private *GameService instance* is declared as *new GameService()* which creates a new singleton instance. The public *GameService getInstance()* method is subsequently used to retrieve the current singleton instance. A new game instance is then constructed using the *Game addGame(String name)* method. This method accesses the *Game* class due to the *Game* declaration and is used to check if a Game exists in the aforementioned array list of Game. If the game doesn’t exist, they are added to the array list. Public getter methods *Game getGame(long id)* and *Game getGame(String name)* are subsequently declared to return the game instance with the specified id or name respectively. Those methods work because *GameService* has a cardinality of 0 or more Games (which means *GameService* has access to the *Game* class).The class concludes by declaring the public getter methods *int getGameCount(), long getNextTeamId(),* and *long getNextPlayerId().*

The *SingletonTester* class includes a single method known as *void testSingleton()* which is used to test a singleton’s behavior.

The final class is known as *ProgramDriver* and only includes the *main()* method.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | The *MacOS Server* is available exclusively for Apple device and provides the tools to manage *MacOS* devices. This is especially useful because it enables easy construction of application and features for clients using *MacOS.* Furthermore, the *MacOS Server* is extremely user friendly with great support and easy workload distribution. This latter feature makes *MacOS Server* the best system for increasing processing power. Unfortunately, as mentioned before, the *MacOS Server* is available exclusively on Apple devices which can be quite expensive. The lack of adoption also means many applications are not available on *MacOS.* | On the sever-side, *Linux* is a free, open-source system with a large distribution and fantastic community support. This means *Linux* accounts for roughly 80% of all servers. The variability of *Linux* allows it to support process, multi-thread operations. Unfortunately, this malleability means using *Linux* requires extensive knowledge for everything from installation to bug fixing. | The *Windows Server OS* is more user-friendly than it’s counterparts. This nature means that the *Windows Server OS* supports enterprise-level management, date storage, and a vast array of applications. Furthermore, the *Windows Server OS* has support for symmetric multiprocessor systems. Unfortunately, the *Windows Server OS* requires user-based licensing while still being more prone to security threats (when compared to other platforms). | Mobile devices utilizes servers designer for singular applications. The server essentially talks with the client’s mobile device based on their interactions with an application. This server is responsible for keeping track of user data, app data, scheduling events, and pushing changes to the application. While some applications don’t require a server, one is necessary whenever an application needs to control its content or store user data. These servers enable a mobile device to play high intensity games. After all, the game would be stored and played from a server capable of handling said game. Unfortunately, from personal experience, these servers aren’t especially powerful and only work optimally when utilizing ridiculously fast internet speeds. |
| **Client Side** | *MacOS* comes preinstalled on all apple devices and is, by extension, free. Because *MacOS* comes preinstalled on all apple devices, the operating system is been optimized for working with these devices. This ensures that quality hardware is tuned to the operating system which makes *MacOS* easy to use. Having said that, users are not limited to *MacOS*. Instead, they can freely choose between *MacOS*, *Linux*, and *Windows*. Unfortunately, apple devices are quite expensive which makes entry-level more costly. Furthermore, because *MacOS* is only available on Apple devices, your hardware options are limited to Apple devices. Lastly, updates have a tendency to break older hardware. | Linux is an open-source operating system with a wide range of distribution. Due to it’s open-source nature, *Linux* is usually completely free. Although premium versions exist, these options are still cheaper than their competitors. Unfortunately, there are very few choices for pre-built *Linux* machines. This inherently leads to a lower adoptions rate which means brand-name applications usually aren’t available on *Linux* systems. Another major issue of the *Linux* operating system is that it experiences some file-format compatibility issues. | Because *Windows* comes preinstalled on nearly all devices, it is the most commonly used operating system in the world. The widespread appeal is also beneficial to the user because it means a *Windows* device exists for every user need. Furthermore, because it comes preinstalled on most systems, most users don’t have to pay for it. The only exception to this rule is with people who build a PC for themselves. Unfortunately, the widespread appeal of *Windows* has brought significant drawbacks. Most significantly is it’s inconsistent functionality and quality. This problem is worsened by *Windows* susceptibility to all forms of malware. | The *Mobile Operating System (OS)* specializes in user-friendliness at the cost of hardware access. Because of this, the *Mobile OS* prioritizes battery life at the cost of performance. While this means mobile devices are able to operate with minimal RAM, they are unable to take advantage of fast CPUs or larger amounts of RAM. While this tradeoff increases performance speed, it means these devices aren’t capable of fully harnessing their hardware. Furthermore, a user is unable to make alterations to their hardware meaning they are unable to sacrifice speed for higher processing power. Because the *Mobile OS* is available on all mobile devices, it is (technically) the most widely used. However, these differ based on the system. These OS systems range from *IOS* (*MacOS)* to *Android (Linux)* to *Windows Mobile OS.* The benefits and drawbacks of each *OS* can be found in their respective sections. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Tools** | The *MacOS* is available exclusively on apple devices meaning the OS system is specifically designed for the hardware it is running on. This inherently means that devices using the operating have fantastic tool support. Although Apple provides fantastic support, the operating system is capable of supporting most *Linux* tools. Unfortunately, some of the services built for *Linux* only partially run on Apple devices. Furthermore, Apple utilizes planned obsolescence were future updates desecrate older devices. Lastly, like most operating systems, *MacOS* is written in the *C* programming language because this language is closer to the hardware which makes it more portable and efficient. While other languages can still be written, they require specialized tools like *Eclipse* (Java), *Code::Blocks* (*C++),* and *Pycharm (Python).* | *Linux* is an extremely customizable environment with fantastic tool support which makes it able to run on most hardware. Unfortunately, configuration can take forever. Furthermore, you are your own support team which means you can spend forever fixing small issues that you accidentally caused. Luckily, the lack of support can be countered by *Linux* from a larger, more popular distributor (like *Ubuntu*). Lastly, like most operating systems, *Linux* is written in the *C* programming language because this language is closer to the hardware which makes it more portable and efficient. While other languages can still be written, they require specialized tools like *Eclipse* (Java), *Code::Blocks* (*C++),* and *Pycharm (Python).* | The primary benefit of *Windows* is that developed software runs on a ton of hardware. Unfortunately, the amount of hardware that *Windows* runs on still pales in comparison to applications developed in *Linux.* This problem is made significantly worse because *Windows* has a history of terrible tool support which requires special software to run basic functions. This limitation has historically made developing software for *Windows* especially difficult. Luckily, in recent years, *Windows* has doubled down of supporting developers. Most notably, *Windows* can now run a lot of software through *Windows Subsystem for Linux.* Lastly, like most operating systems, *Linux* is written in the *C* programming language because this language is closer to the hardware which makes it more portable and efficient. While other languages can still be written, they require specialized tools like *Eclipse* (Java), *Microsoft Visual Studio* (*C++),* and *Pycharm (Python).* | There are several integrated development environments (IDEs) used to create applications for mobile devices. It should be noted however that these applications are designed on web-based systems, not on mobile devices. *Android* based applications are often developed in *Android Studio.* This IDE is available on all three primary operating systems and utilizes both *Java* and *C++.* Meanwhile, *IOS* applications are developed using *Xcode* which is available exclusively on systems utilizing *MacOS* and uses all forms of the *C* programming language. If you don’t have access to an Apple device, you’d need a virtual machine like *VirtualBox* or *VMWare* to access *Xcode.* |

## Recommendations

1. **Operating Platform**: Based on the client’s needs and desires, I recommend using *Linux* as their operating platform. *Linux* is a free, open-source platform that excels due to its stability and performance. Furthermore, *Linux* does not suffer from the major security issues that compromise *Windows-*based systems. While *MacOS* doesn’t suffer from these issues either, *Linux* offers far more customizability. This customizability ensures the developed software can be tailored to suit the various platforms that the client seeks to pursue.
2. **Operating Systems Architectures**: Given the choice to utilize the *Linux* operating platform, I recommend that the client utilizes a multi-thread architecture. By using this operating system architecture, each component of the developed software are able to evolve independently. These components include the client-side presentation tier, the software application’s tier, and the internal data tier. The client-side presentation tier would be responsible for showcasing the user interface while the software application’s tier would govern the application’s logic. Lastly, the internal data tier would be responsible for managing data storage. Because each of these components is able to evolve independently of one another, finding and addressing any issues should be far easier to manage.
3. **Storage Management**: One requirement of *Draw It or Lose It* is that the application should possess the ability store long-term assets. During gameplay, these assets should be ported into the application at a fast and steady rate. Furthermore, these images must maintain a high level of quality. Therefore, I recommend utilizing the storage management technique such as cloud storage integration. This choice is necessary for offloading some of the necessary storage requirements. Because of our choice to utilize *Linux,* I recommend using the *Ubuntu Service* which is both free and open source.Cloud services like *Ubuntu Service* provide a streaming approach to memory management which allows the massive library of assets to be stored on a host server. Meanwhile, clients will only download and store assets as needed, which should reduce the burden on the device’s local storage.
4. **Memory Management**: Memory management focuses on actions which are currently being executed and, as such, is extremely important to the operation of *Draw It or Lose It.* To that end, I recommend utilizing Virtual Memory which should separate logical memory perceived by the users from physical memory. Utilizing this technique allows secondary memory to be used as though it were a part of main memory. Simply put, virtual memory ensures that developed program are not“…constrained by the amount of physical memory that is available” (Silberschatz et al., 2008, p. 398). Therefore, by utilizing virtual memory, programming is made easier while also still being able to run faster than other processes. Luckily, virtual memory is one of the primary benefits of *Linux* because all memory stored within the operating platform is virtual memory.
5. **Distributed Systems and Networks**: Draw It or Lose It must be able to communicate across various platforms which necessitates the use of a distributed system and network. To meet these commands, I recommend utilizing a RESTful API. REST is a distributed hypermedia system which “…is a network of resources accessible via HTTP with hyperlinks that contain the addresses of other linked resources” (CS 230 A RESTful API). REST is specifically recommended because it provides excellent support for Linux platforms which eases implementation.
6. **Security**: One of the most critical parameters that our client requires is an emphasis on security and protection. Luckily, security was one of the primary concerns when choosing to recommend the *Linux* platform. The security benefits offered by *Linux* originate from it’s use as an open-source software. Due to this, whenever new vulnerabilities are discovered, free security patches are issued which help keep the platform secure. Another security benefit of *Linux* is it’s dependence of user privileges. To this end, all users require individual IDs and passwords. Because users are kept separate, even if a user was able to infect the system with malware, the likelihood of the malware contaminating other systems is minimized. *Linux* also maintains a log file which tracks all file and system access. Thus, in the unlikely scenario that a user deployed malware, administrators would be able to identify the root of their security problems and take preventative measures to combat future threats. As mentioned before, *Linux* is an open-source platform. This choice means even the most powerful agencies on the planet have invented software and techniques that help protect the system. For example, the United States’ National Security Agency (NSA) developed *SELinux* which allows administrators to manage security options by defining access permissions for all applications, processes, and files. This system would subsequently deny users for accessing information violate these permissions. By default, all users are automatically assigned lower access rights which hinders root access and, by extension, their ability to spread malware. These are just some of the security benefits offered by *Linux* which will prove vital throughout the development process.

Works Cited

Bhardwaj, R., & Rashmi Bhardwaj. (2023, July 31). *Mobile Operating system VS Desktop Operating System*. IPWithEase. <https://ipwithease.com/mobile-operating-system-vs-desktop-operating-system/>

Chapel, J. (2019, July 1). *Cloud Storage Cost Comparison: AWS vs. Azure vs. Google*. Medium. <https://jaychapel.medium.com/cloud-storage-cost-comparison-aws-vs-azure-vs-google-844dfff3d324>

*Difference between Mobile Operating System and Desktop Operating System*. www.javatpoint.com. (n.d.). <https://www.javatpoint.com/mobile-operating-system-vs-desktop-operating-system>

Durbin, C. (2019, February 26). *Developer tools: The Operating System*. DEV Community. <https://dev.to/wuz/developer-tools-the-operating-system-3n55>

Patel, B., & Written by Bhaval Patel (2023, November 7). *How to develop IOS apps on windows in 2023 [5 best ways]*. <https://www.spaceotechnologies.com/blog/develop-ios-apps-on-windows/#:~:text=Since%2C%20without%20using%20Xcode%20(an,access%20to%20virtual%20Mac%20OS.>

*I want to develop Android apps - What Languages should I Learn?*. Android Authority. (2020, December 18). <https://www.androidauthority.com/develop-android-apps-languages-learn-391008/>

Jhonny. (2018, November 16). *Why does my Mobile App need a Server?*. SideCode. <https://sidecode.io/mobile-apps/mobile-development-mobile-app-server/>

Kaspersky. (2023, August 16). What is Linux and is it really Secure?. usa.kaspersky.com. <https://usa.kaspersky.com/resource-center/definitions/linux>

Marijan, B. (2023, August 17). *Server Operating System: Server OS Types & How to Choose*. Knowledge Base by phoenixNAP. <https://phoenixnap.com/kb/server-operating-system>

*Managing and Optimizing Memory in Linux*. Site24x7. (n.d.). <https://www.site24x7.com/learn/linux/optimize-memory.html#:~:text=The%20entire%20memory%20in%20Linux,space%20from%20the%20hard%20disk.>

OpenSource.com. (2018, May 31). *What is a Linux Server and Why does your Business Need One?*. Linux.com. <https://www.linux.com/news/what-linux-server-and-why-does-your-business-need-one-0/>

Silberschatz, A., Gagne, G., & Galvin, P. B. (2008). *Operating System Concepts, 8th edition*. Textbook, John Wiley & Sons.

Southern New Hampshire University (SNHU). *CS - 230 A RESTful API*.

*Ubuntu*. (n.d.-d). [https://ubuntu.com/download/server?country=](https://ubuntu.com/download/server)

*What is REST: Rest API Tutorial*. REST API Tutorial. (2023, December 7). <https://restfulapi.net/>

Wikimedia Foundation. (2023, November 20). *Comparison of Integrated Development Environments*. Wikipedia. <https://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments>

*Windows, Mac, or Linux? We Compare the Pros and Cons of these Computing Platforms*. ZDNET. (n.d.). <https://www.zdnet.com/article/windows-mac-or-linux-we-compare-the-pros-and-cons-of-these-computing-platforms/>