

Christopher Trimmer

CS-230-T6573 Operating Platforms

08/10/2022

Draw It or Lose It

# **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)4

[**Evaluation**](#_2o15spng8stw)5

[**Recommendations**](#_m8aleynsvzvc)11

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/12/2022 | Chris Trimmer | Document creation. Complete the executive summary, design constraints, and domain model. |
| 2.0 | 07/26/2022 | Chris Trimmer | Updating document to include OS evaluation comparing Windows, Mac, Linux, and Mobile. |
| 3.0 | 08/10/2022 | Chris Trimmer | Updating document to include final recommendation of Operating Platform. Further provide analysis of OS Architecture, Storage and Memory management, Distributed Systems, and Security. |

## [Executive Summary](#_sbfa50wo7nsh)

Our client, The Gaming Room, is interested in developing a web-based game that can be played on multiple platforms. They want to base it on their current game, Draw it or Lose It, which is only available on the Android operating system. It works loosely based on the old television show ‘Win, Lose, or Draw’. The game will include multiple players and one or more teams playing the same game instance. For this reason, we will be designing the game using the singleton design pattern. To enhance performance of searching games, teams, and players, we will also be implementing the Iterator pattern to traverse our data structures.

Since the game will be web-based and available on multiple platforms, we will have design constraints in implementing the game. For example, we need to consider a portable language, and a language that has support for RESTful services. We have determined that using Java with Jersey (JAX-RS) API will meet our needs. Furthermore, we will be following the principles of Object-Oriented Programming (OOP), and will therefore design the software based on the four pillars: encapsulation, inheritance, polymorphism, and abstraction.

## [Design Constraints](#_2et92p0)

Specific design constraints are as follows:

* Only one instance of a game can exist at any given time.
* A game must be able to contain one or more teams.
* A team must be able to contain multiple players.
* Names must be unique for the Game instance, teams, and players.
* There must be a way to check whether a name is available or not.
* The game must be web-based and available on multiple platforms.

Since only one instance of a game can exist at a time, we will be implementing the singleton pattern. We will be using array lists to hold game objects: the primary game app will have an array list of game instances; each game instance will have an array list of teams; and each team will have an array list of players. Since game names, team names, and player names must be unique, will have validation in the code to test name creation for each object.

The game is currently only available on Android. We may need to refactor code for any platform that is not Java-based. Since we are aiming for a web-based distributed environment, following a REST-based approach is appropriate. Java has a built-in API, JAX-RS, that simplifies development of REST-based applications. We will also be using the Jersey API, which extends JAX-RS and provides more features and utilities. Using a RESTful solution, we can ensure the game is widely distributed. A constraint exists based on server load. We will need to ensure we have enough backend servers to accommodate load expectations.

## [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 game will be developed following the principles of OOP. Below is the UML diagram that depicts the design of the main classes in the game. First, we annotate each class with its name, public and private attributes, and operations. This satisfies encapsulation in several ways. First, by using access modifiers we control how members and operations are accessible to other parts of the program. Second, the data members and operations are contained within the class, essentially “encapsulating” them to the class they are designed to be used for. In effect, everything required to consume the class is contained within it.

Next, we show inheritance hierarchy among Entity, Game, Team, and Player. The classes Game, Team, and Player share common characteristics so it makes sense to create a base class where they can be derived from. The base class is Entity and contains those common attributes and methods. This promotes code-reuse, enables extensibility, and is overall sound programming design.

We also see the association among the GameService, Game, Team, and Player classes. We use association because they are not subclasses of each other, rather they use the class for a purpose. In this scenario, GameService contains a list of Games, Game contains a list of Teams, and Team contains a list of Players. These associations show multiplicity of “zero to many” – meaning that a Game can have zero or more Teams, the GameService can have zero or more Games, and a Team can have zero or more players, as examples.

We are creating the GameService class as a singleton to meet requirements that a game must be a unique instance. This is annotated in the diagram with the static members and static method underlined. The GameService constructor is also labelled as “private”, which means the only way to create a new game instance is via the static method. This prevents multiple instances of the game from being created.

Polymorphism is carried out using the toString() method of the Java base Object class. The toString() method is implemented in the base class Object. Our Entity, Game, Team, and Player classes each override and implement the toString() method in their own way. Namely, they use the toString() method to print details about their respective class objects such as their attributes. For example, a Player object will use toString() to print the Player attributes, while a Team object will use toString() to print team specific objects.

Finally, abstraction is seen at several levels. Firstly, we are modeling our classes based on real-world ideas. This helps organize and express the problem in a way that is easier to understand. Even the simplest of games can be complex to create in code. Abstraction is a way to model the game in a way that is easier to manage and understand. Secondly, we create the data members and operations in a way that hides complexity. For example, in the Game, Team, and Player classes we implement array lists to store their respective associated class objects (Team holds an array list of Players, for example). Consumers of the Team class simply access the players without care or concern about the underlying data structure of the player list.

**"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 | Operating Platform | Remarks |
| Server Side | Mac | The cost to build a Mac Server can be high. There will potentially be a lot of people playing, so we need a powerful machine to host the game. A rough estimate for a machine to use as a webserver is approximately $6000 just for the starting point of a Mac Pro. At a minimum, we will want to upgrade the processors, and RAM, which will increase cost. To make it a server, we need to buy the server application, which is a small additional fee.  From a software and programming perspective, the Mac server is ideally suited for software development, however there might be a learning curve if the team is not familiar with Mac products. While Mac OS is GUI-based, the team will need to be familiar with using Terminal (or some form of command line utility). Specific to software development, Apple has XCode built into its OS. For cross-platform development, we need to add other IDE's. |
| Linux | The build for a Linux server will be dependent on how much load we expect. Overall, we may see cost savings here, as we are not tied to specific hardware like the Mac. We could also choose to run the server on a virtual machine, so if we have existing equipment that is suitable, then we could see cost savings.   To operate a Linux server, we should consider which flavor of Linux we are going use. There really isn't a standard OS (like Windows and Apple/Mac OS). This gives us flexibility in using any flavor we want; however, the team will need to become familiar with it. A benefit of Linux in general is the number of open-source tools for the various flavors. This enables flexibility and adaptability to whatever our clients need. However, it could also increase complexity. |
| Windows | Again, the price of hardware is dependent on how much load we expect the server to handle. The cost may be comparable to a Mac Pro, but the benefit is that we have more variety in hardware compared to Mac.  There are various versions of Windows Server (2019, 2012, etc.), so the team will need to be comfortable with whatever version we use. WIndows server comes with a lot of community support, and the team may already be familiar with it. So, there is potentially less training involved with this option. |
| Mobile | For developing the server side for mobile apps varies based on the mobile device. Any of the above OS's will suffice fo mobile development, but we may choose a Mac-based solution based on the ability to use the built in X-Code, which easily translates to iOS- based development. For android devices, we may consider any of the options. The main thing needed are the tools for supporting REST-API based programming. |
|  |  |  |
| Development Requirements (cont.) | Operating Platform (cont.) | Remarks (cont.) |
| Client Side | Mac | Mac clients can be expensive. A general starting point for the iMac’s are around $1500. Again, these workstations will probably need to be upgraded for the developers, which will increase cost.  There may be a considerable investment in time and training if the team is not familiar with Mac software and applications. This includes the inherent Apple IDE (XCode), and other IDE's required for cross-platform development. |
| Linux | Linux clients can be stand-alone machines, or a virtualize machine running on top of a Mac or Windows-based OS. As mentioned previously, for the development we would probably want the team to be on the same flavor. The game needs to be able to run on any flavor. The benefit of Linux is that it is open-source and that are a lot of tools and API's to help with interoperability.  The main downside is the increased complexity of interoperability. The more API's and tools that we need to use, increases the complexity, and will require more development effort.  From a gaming perspective, Linux is not used as much for gaming as other platforms. However, since the game is web-based we mainly just need to make sure it is accessible via REST-based approach. So, there is a tradeoff between amount of development effort, and actual game play. |
| Windows | Windows is more popular and will be more familiar to the development team. Standing up Windows clients is inexpensive, and the team will probably have a much quicker start up time. Most of the programming languages required for this effort can be performed on the Windows platform, and suitable IDEs are easily available.From a gaming perspective, there are many users who used Windows based OS. So, developing specifically on this platform will help us in testing and ensuring functionality on the most popular platform. |
| Mobile | We will be using various IDE's to write the mobile development code. Our developers will need to be able to use and test based on a variety of mobile devices. The primary mobile OS's to develop for will be Android and iOS. To carry out game development for mobile development will require specific skillsets on those OS's, or training our current staff.  From a gaming perspective, customers are increasingly using mobile devices for gaming. We will have the requirement to maintain our mobile app solution to be inline with Android and iOS specific updates. This will require ensuring compatibility with the previous versions of Android and iOS and future versions. |
|  |  |  |
| Development Requirements (cont.) | Operating Platform (cont.) | Remarks (cont.) |
| Development  Tools | Mac | At a minimum, the development team will need to be familiar with or learn Terminal/Command line programming, XCode, and various IDE’s used in software development for cross-platform applications. VS Code is portable, which specializes in C-based languages. The Eclipse IDE specializes in Java. XCode has all the tools needed to development of iOS apps, but we may need more specialized tools for programming for other platforms.   Mac is relatively easy to use, as it is GUI-based. However, command line will still need to be used for development purposes.  There are various languages that the team will need to know for this project: Java, C-based languages, JavaScript, HTML, and CSS, and of course the IDE(s). This applies to all the operating platforms. |
| Linux | As stated earlier, the Linux environment will require the development team to be more familiar with command line tools and using a terminal window. Similar to Mac, the tools associated with Linux include cross-platform IDE’s such as VS Code and Eclipse. This will depend on if our final decision is C-based or Java-based (or both), respectively. There are various languages that the team will need to know for this project: Java, C-based languages, JavaScript, HTML, and CSS. This applies to all the operating platforms. One of the main benefits of the Linux environment is the number of open-source tools. Since we are developing the game to be available on all platforms, the ability to use open-source tools is especially helpful. This will require flexibility and adaptability on the part of the development team, and of course a learning curve. On the other hand, this also creates a certain amount of complexity, as there will be more tools integrated in the project. |
| Windows | Development tools such as VS Code and Eclipse can be used on the Windows side, just as easily as they can be used on the Mac and Linux side. Overall, Windows may be easier for a development team to use simply because of familiarity.  Development team may not need to use as much command line programming, because Windows is primarily GUI-based. However, command line programming can be faster and easier to perform tasks.  There are various languages that the team will need to know for this project: Java, C-based languages, JavaScript, HTML, and CSS. This applies to all the operating platforms. A benefit of the Windows environment is that the team will probably be more familiar with it, so we will probably have faster start-up time. This also applies to the IDE's. VS Code and Eclipse are widely used. We will need to use UWP, or a virtualized Mac environment for programming iOS apps. |
| Mobile | To program mobile apps, the development will be using IDEs on the respective platform that is chosen. For example, Unified Windows Platform (UWP) can be used to program apps for the Microsoft Store. UWP can be ported to Android and iOS. XCode can be used to build apps for Mac and iOS. There are various languages that the team will need to know for this project: Java, C-based languages, JavaScript, HTML, and CSS. This applies to all the operating platforms. For mobile apps and gaming platforms, we may need specific tools. For programing Android apps, we could use the Android Studio IDE. It is available on all platforms, and it is free. For iOS programming, again, we can use IDE’s and tools such as UWP and XCode that have already been mentioned. We can also use a game specific IDE such as Unity, that can be used for cross-platform development. |

## Recommendations

In this section I will provide my final recommendation for the Operating Platform we should for this development effort. Furthermore, I will provide analysis of the OS architecture, storage and memory management, distributed systems and network, and security aspects for this effort.

1. **Operating Platform**: I recommend we use the Windows operating platform for this development effort. Windows is the more popular and widely used operating platform today, and our team will most likely have more experience with this platform than the others. Using a system that we are more familiar with should help us develop and implement the project more quickly. Furthermore, we will be using products within the Microsoft ecosystem, such as Azure, Office365, and SharePoint, and therefore we will be able to easily integrate these tools and services into our workflows. All our development tools are capable of being run on Windows, as well. Finally, we should consider that most of customers will be using Windows based clients, so by developing on the Windows platform, we can more easily perform testing of the game using this platform as well.
2. **Operating Systems Architectures**: We will be using a client-server architecture for this development effort. We will have Windows servers that will perform Active Directory services to manage accounts and policies. We will also be using Windows-based servers to host the main application. Furthermore, we will have Microsoft Azure SQL database servers that will be use for storing and maintaining the state of data used in the application.  
     
   The most suitable architecture type will be a three-tier (multi-tier) architecture. The client will communicate with the application server, and the application server will access the database server. Likewise, in reverse order, the database server will communicate with the application server, and the application server will communicate back to the client. This forms a clear separation of concerns among the main entities in the architecture.  
     
   Our development systems will be operating in the communication layer architecture following the OSI-model. This is composed of seven layers: Application, Presentation, Session, Transport, Network, Data-Link (including media access), and Physical. At the operating system level, we will also follow the traditional OS layer architecture, which can consist of all or some of these seven layers: User Interface, Application, Middleware, OS Interface, OS Kernal, Hardware Device Drivers, and Hardware (Lavieri, 2019). This follows a logical path of communication the OS will use in the computing system. As noted by Lavieri (2019), the benefit of this layered approach at the OS level is that a change in one layer should not negatively impact the other layers.
3. **Storage Management**: We will be using SSD drives as our mass storage medium. SSDs are faster than typical SATA hard drives, which enable developers for process data, compile code, and deal with graphics and images much faster. Typical SATA hard drives may be cheaper, so these may be an option if we need mass storage for data that is going to be strictly at rest and not used for time intensive operations. Furthermore, since our operating platform is Windows, we will be using NTFS as our main file system. NTFS has been used on the Windows platform for nearly three decades and has proven itself as an effective file system. NTFS enables security features such as ACL-based security and support for BitLocker technology (Microsoft, 2019). Some of our servers will be hosted in MS Azure cloud. To ensure optimal performance of these servers we will also configure them with SSD disks.  
     
   Since we are going to be using Windows as our primary Operating Platform, we will also leverage Microsoft OneDrive for personal cloud storage for our developers. We will use Microsoft SharePoint Online for document and content management. For development, we will be using Git as our version control system. We will use GitHub as our Git repository. GitHub is cloud-based and can be easily accessed and used via the web and using a Git command line tool such as Git Bash.
4. **Memory Management**: One of the primary concerns we will have for game performance is ensuring clients and servers do not waste main memory dealing with images. As described by Silberschatz, Galvin, and Gagne (2008), random access memory (RAM) is main memory, and is where the actual application will be running. On the client workstation, this is where the actual game will be executing for the duration of the game. The game ‘Draw It or Lose It’ will consist of many image files. We don’t want to load every image into main memory, as this will likely cause an overbearing amount of page-file swapping and cause horrible performance on the client system. The strategy therefore will be to only load the required image for a specific game at run-time to main memory. When the game is in progress, it will only load that image into main memory, and therefore save overall space of the running application. When the next round of the game begins, the program will unload the image and then contact the server to load a new image or pull an image from the local system if it has been pre-loaded.  
     
   For development workstations, we will require a large amount of RAM, as developers will be using memory intensive applications, utilizing coding IDE’s, and perform programming tasks such as compiling and testing the software application itself. Developers will need to be able execute many these applications and tools simultaneously, without causing excessive paging.
5. **Distributed Systems and Networks**: As noted earlier, our OS architecture will be a client-server model. As part of the design of this game, the client has requested the application to communicate between various platforms. It follows that we will implement the solution as a distributed system. Our recommendation is to follow the REST architectural style. From a network perspective, communication between servers and clients will occur using the TCP/IP protocol. This is a stateless communication process, which basically means that the server-side will not maintain the state of the client. The servers will be “listening” for requests from the clients. Communication will happen in the form of HTTP verbs such as PUT, GET, POST (and others). The server will respond appropriately based on the current data at the time of the request. This is another form of separation of concerns and is what makes it possible for clients on different platforms to be able to access and run the game application.  
     
   Since the game will be reliant on network connectivity, we will need to address outages and availability. We are recommending using Microsoft Azure for our databases and servers, which include the actual servers and data residing in the cloud. In the case of network outages and maintenance, however, communication with these servers could be disrupted. Azure cloud uses a high availability architecture, which has a goal of ensuring 99.99% uptime on server instances (Setlem, 2022).
6. **Security**: Windows has various built-in security features that are intuitive and generally work without much user intervention. From the perspective of data integrity, we can encrypt our hard drives using BitLocker. When BitLocker is enabled, data on the hard drive is encrypted. The users specify a password during the setup process, and this password is used to access the drive. We can also encrypt files and folders individually using Microsoft’s built-in Encrypting File System (EFS). The user can simply enable encryption on a specific file or folder, and he or she will be the only person able to access it.  
     
   In the Window’s environment we can also set accessibility rights and privileges based on roles, individual users, and user groups. For example, we might have a shared folder that we only want Developers to have access to, so we can create a user group that includes only those personnel. We then apply the appropriate permissions the folder that includes the developer group. This can also be applied at the file level, as well. In an enterprise environment, security groups, permissions and privileges can be managed on a server and implemented via active directory and group policy to the workstations.  
     
   Window’s also has built in security control for how applications are run and installed with User Account Control (UAC). The idea behind this feature is that the user runs applications from the context of a user with standard privileges. If the user needs to run an application or perform a system change, such as installing new software, UAC will require them change context to administrator privileges (Montemayor, 2022). Finally, Windows also has a built-in security system, aptly called Windows Security. This includes a family of security features including anti-virus and malware protection, a firewall, account protection settings, and hardware-level security settings.

**References**

Lavieri, Edward (2019). Hands-On Design Patterns with Java. Packt Publishing Ltd.

Montemayor, Dulce (July 12, 2022). User Account Control (Windows 10) - Microsoft 365 Security.

Docs.microsoft.com.

https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-

control/user-account-control-overview

‌Microsoft. (2019, June 17). NTFS Overview. Microsoft.com.

https://docs.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview

Setlem, Rajesh (July 15, 2022). High availability - Azure SQL Database and SQL Managed Instance.

Docs.microsoft.com. Retrieved August 9, 2022, from

https://docs.microsoft.com/en-us/azure/azure-sql/database/high-availability-

sla?view=azuresql&tabs=azure-powershell

Silberschatz, Galvin, Gagne (July 2008). Operating System Concepts, 8th Edition [Book]. (July 2008).

Www.oreilly.com.

https://learning.oreilly.com/library/view/operating-system-concepts/9780470128725/