

**Draw It or Lose It**

# Project Software Design Template — Version 3.0

# Ryan Rieth — Prof. Fredericks — CS 230 — SNHU

## Table of Contents

[Project Software Design Template — Version 3.0 1](#_Toc111213996)

[Ryan Rieth — Prof. Fredericks — CS 230 — SNHU 1](#_Toc111213997)

[Table of Contents 2](#_Toc111213998)

[Document Revision History 2](#_Toc111213999)

[Executive Summary 3](#_Toc111214000)

[Design Constraints 3](#_Toc111214001)

[System Architecture View 3](#_Toc111214002)

[Domain Model 4](#_Toc111214003)

[Evaluation 1.0 (Project One) 5](#_Toc111214004)

[Evaluation 2.0 (Project Two) 6](#_Toc111214005)

[Recommendations 1.0 7](#_Toc111214006)

[Recommendations 2.0 (Project Three) 8](#_Toc111214007)

## [Document Revision History](#_grjogdjh5fi8)

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 0.1 | 07/16/22 | Ryan Rieth | Document creation. |
| 0.2 | 07/16/22 | Ryan Rieth | Executive Summary section completed. |
| 0.3 | 07/16/22 | Ryan Rieth | Design Constraints section completed. |
| 0.4 | 07/17/22 | Ryan Rieth | Domain Model section completed. |
| 0.5 | 07/17/22 | Ryan Rieth | Evaluation section completed. |
| 0.6 | 07/17/22 | Ryan Rieth | Recommendations completed. |
| 1.0 | 07/17/22 | Ryan Rieth | Final version for submission of Project One. |
| 1.1 | 07/29/22 | Ryan Rieth | Adjustments made to Evaluation section. |
| 1.2 | 07/29/22 | Ryan Rieth | Adjustments undone. |
| 1.3 | 07/29/22 | Ryan Rieth | New Evaluation section created. |
| 1.4 | 07/29/22 | Ryan Rieth | New Evaluation section filled out. |
| 2.0 | 07/29/22 | Ryan Rieth | Final version for submission of Project Two. |
| 2.1 | 08/12/22 | Ryan Rieth | New Recommendation section created. |
| 2.2 | 08/12/22 | Ryan Rieth | New Recommendation section filled out. |
| 3.0 | 08/14/22 | Ryan Rieth | Final version for submission of Project Three. |

## 

## [Executive Summary](#_sbfa50wo7nsh)

We here at Creative Technology Solutions (CTS) propose to design and implement an interactive entertainment application titled “Draw It or Lose It” to meet the request of client company The Gaming Room. The requested application comes with some requirements:

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique.
* Only one instance of the game can exist in memory at any given time.
* Each instance of a game, team, or player must have unique identifiers.

The application must not only be indiscernible from the company’s currently existing Android rendition but also seamlessly allow play across multiple platforms via a web-based implementation. Housing the game’s scripting within this web-based environment will allow exactly what is desired: access across any platform and any operating system that has web access enabled.

## [Design Constraints](#_2et92p0)

Some of the design constraints are present in the Executive Summary section above, such as multiple team playability and limiting the number of games running at a time to one. Some other design constraints involve seamless web-based functionality as well as secure encrypting for users’ privacy. Another design constraint is, of course, matching the design quality of the current Android rendition of the application to allow present users an easy adoption of the new web-based variation on the game.

These design constraints set a rigidity in the application’s development. But this rigidity is not a negative limitation to the development. As so, these constraints are not restraints but, instead, sparks to fuel creativity and critical thinking. Without a clear path and clear sub-goals, it may be impossible to figure out how to start and how to end. These constraints implicate the developers to “follow the path” but allow them to figure out their own ways on how to accomplish 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 class diagram below portrays the inner workings of the *Draw It or Lose It* application for The Gaming Room. These individual coding classes work independently to fuel certain elements of the game; they also communicate with and utilize each other in ways to service the true functionality of the game that the user will experience.

For example, *ProgramDriver* — redundantly, the class that drives the program — uses *SingletonTester*, which is a way of making sure that only one game can run at a time to make sure the application’s memory usage is well-tended to.

See also that *Entity* is “inherited” by the *Game*, *Team*, and *Player* classes. What this means is that *Entity’s* attributes of “id” and “name” are also utilized by *Game*, *Team*, and *Player* but in unique instances. In other words, each *Game*, *Team*, and *Player* can have their own unique ID and name to signify their selves from others. This is vital in making sure a new game/team/player is unique from currently existing ones or previously existing ones.

Lastly, notice *GameService* is “associated” with *Game*, and *Game* in the same way with *Team*, and *Team* similarly with *Player* (denoted by “0…\*). What this means is that there can be 0 to many instances of *Game* in *GameService* (this does not mean more than one *Game* can exist at a time, just that *GameService* can handle more than one *Game*); or that there can be 0 to many teams associated with one *Game*; or that there can be 0 to many players associated with one *Team*.

This is an element of web-based, online gaming applications that allows inter- and intra-connectivity between players across the world or a player’s game across devices.

**"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) 1.0 (Project One)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Development Requirements | Mac | Linux | Windows | Mobile Devices |
| Server Side | Mac is very stable with servers, perhaps even matching Linux in this regard at times. It is based on a BSD system like Linux.  Mac is not the most universal operating system out there, but it is well-revered in business/gaming implementation. | Linux is extremely stable with servers and hosting being based on Unix. It can handle a lot without needing rebooting or too much TLC. | Windows can be relatively unstable on the server side with a history of blue screens, bugs, system crashes, etc.  However, Windows is the most applied operating system (most universal). | Mobile devices tend to borrow from their parent device’s operating system server scenarios. So, a game made for both Mac and Windows may perform well on either, but its Apple rendition may actually be more stable in performance than its Windows counterpart. |
| Client Side | Apple is very user-friendly and is built similarly like Windows in terms of file storage and path routes, but its learning curve needs to be surpassed first. Macs are also world-renown for their privacy and security.  Mac’s hardware and software communication is unmatched as Apple alone handles the development of both, simultaneously.  Mac is known to be on the expensive side because of this reason, though, and even scripting using Mac’s Swift language may be more expensive than, say, C++ for Windows. | Linux is extremely difficult to learn and handle, beyond its learning curve. Vast knowledge and experience is recommended. However, once it is learned, there is not much maintenance required.  Linux is open source and, therefore, is the most affordable operating system at *free* cost. Linux, in other words, can easily be budget friendly. | Windows is the most widely used operating system, so finding computability for it is not difficult. Windows can require a lot of maintenance, though.  Windows, also, is the least secure of Mac, Linux, and Windows. Malware is easy to develop for Windows.  Windows is very user friendly with its graphics user interface and does not have a big learning curve.  Being so widely used, scripting for Windows is aplenty and cost effective. | Mobile devices are probably the most widely, universally used technological machines in the world. Nowadays, they are referred to as “computers in our pockets”. It comes as no surprise that more companies are shifting focus to mobile development.  If knowledge is known on the mobile device’s parent operating system, than developing for the mobile aspect is relatively easy. The issues come with shifting design to a mobile face, as well as adjusting memory and CPU usage to be friendlier to a mobile device held in the hand.  Functioning on a mobile device is subjective, too, where there is a vital focus to make usability general enough for many types of hands and hand sizes to comfortably use. |
| Development Tools | Mac does run quite a bit off Objective C, a superset of C, but Mac is known for Apple’s Swift programming language, which is Mac-unique, easy to use, but has a vastly different feel than other operating systems’ languages.  Apple’s own Xcode IDE is probably best for Swift coding. | Linux rests comfortably and firmly on the C scripting language with some Python and C++ thrown in there at higher levels. A critical knowledge set of C is required for Linux.  Eclipse is a great IDE for various languages including C, Python, and C++. | Windows divvies up is properties across C, C#, and C++.  Eclipse, again, works well for coding in these languages. Visual Studio does offer versatility in programming in these languages, too. | C++ is well-known to be the choice programming languages for many, many game engines. Mobile game-based languages, however, can be dependent on the device is runs in — for example, an Apple mobile game may be coded with Swift.  Multiple IDE’s may be required for mobile gaming scripting, such as Xcode for Swift development and, say, Eclipse for C++ coding. |

## Evaluation 2.0 (Project Two)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Development Requirements | Mac | Linux | Windows | Mobile Devices |
| Server Side | Mac servers are easily administered and feature intuitive GUI set-ups, a vast array of support, and can easily distribute workloads across multiple machines. This is all wrapped with an unlimited user license.  Mac servers, however, can only run on Apple hardware. They also require knowledge of Apple-based scripting. Mac servers are great if Mac clients exist in a network (they are also pricey). | Linux offers servers for multi-user, multi-process, and multi-thread operations, but vast and extensive knowledge is necessary for maintenance and bug fixing.  Linux servers are secure, offer a wide variety of distributions, integrate Linux open-source software, and can be controlled via a GUI. They are also incredibly complex and lack long-term support (though they are free). | Windows offers servers that support perhaps the largest array of operations, data storage, and applications ranging from user-level to enterprise-level.  Windows servers come with virtual memory management, multitasking capabilities, and support for peripheral devices. All of this is blanketed under at least a 10-year support lifespan. These servers are also more prone to security risks than any other due to Windows’ extreme commonality. | Mobile devices run on their operating systems that stem from parent operating systems, so many of the pros and cons of each listed operating system can be carried over to mobile devices that run on stemming systems — iPhones from OS X, Galaxy devices from Android, etc.  Mobile devices tend to be the riskiest in terms of security as so much information is stored on them pertinent to the user. Connecting to servers from these devices is, subsequently, risky for both client and server. |
| Client Side | The Mac operating system— interchangeably, OS X — is a closed operating system. What this means is that OS X is very hard to come by and very difficult to script malware and viruses for. For clientele, OS X is incredibly safe and secure, but it possesses a sharper learning curve and is less user-friendly as well as customizable when compared to other operating systems.  OS X is 100% owned, run, and maintained by Apple, which means that it works swimmingly with other Apple products. Without the use of third-party services such as Microsoft Office, though, it is very difficult to make OS X compatible with other systems and programs, unless the programs are scripted specifically for Apple devices. | While using passwords is a common practice, and hacking passwords is a common epidemic, Linux and other UNIX-based operating systems encrypt user passwords and only store the encoded password. The operating system itself is the only entity that can decode the password — one would hope. The issue here, though, is once encrypted the password is lost from the operating system’s control. If someone gets their hands on the password file, decryption can be performed easily.  Linux is open source, but it is difficult to grasp. Out of all operating systems listed, Linux would be the most in terms of cost and time as high expertise is required. | With Windows being very common, most spyware, malware, and software of malicious intent are scripted for Windows machines. Windows, too, has a history of poor though well-intentioned protection and security when dealing with these kinds of matters.  Windows is the easiest of all operating systems to grasp (subjectivity aside). It is also the most common, where cost and time to script for Windows would be minimal when compared to others in this table. Expertise is rampant for Windows, too, as it is so common. | Mobile devices nowadays run off operating systems that branch from parent systems — for example, iOS on iPhones has an Apple-based kernel system while Android is similar to Linux-based kernel operation.  To produce applications for mobile devices, not only is understanding of the operating system itself necessary but also its parent operating system.  Scripting for mobile applications has exponentially grown since the debut of the smartphone, so time, cost, and expertise will be manageable and easy to secure. |
| Development Tools | Apple runs mostly on Objective-C and Swift with some C++. Swift language.  Xcode is Apple-specific, Eclipse and Visual Studio is third-party. The latter two are free wholesale, Xcode costs $100 a year to upload to Apple’s App Store. | Linux ueses good ol’ C language (and assembly code). Many IDEs provide scripting for C.  Seamonkey, Bluegriffon, and Sublime Text are available IDEs for Linux. Respectively, they cost nothing, roughly $200, and $100. | Windows is extensively written in C.  GitHub, Visual Studio, and XLMSpy are a few IDE examples. Respectively, they are free, free, and roughly $500 in cost. | Mobile devices find support in development tools on their respective parent operating systems. In other words, applications are coded on parent operating systems and ported to the devices. They, thus, follow the pros and cons of their respective parent operating systems. |

## Recommendations 1.0

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**: Windows is the recommended operating platform for its popularity, usability, and compatibility with the existing Android-based *Draw It or Lose It*.
2. **Operating Systems Architectures**: Windows is possessed in two architectures: 32-bit and 64-bit. A main difference is that 64-bit can handle much more random access memory (RAM) than 32-bit — total addressable memory on 32-bit is 4GB, 64-bit can exceed this.
3. **Storage Management**: Windows uses a drive-based system for file storage, in which folders and files fall into directories. Storing local game data in an appropriate folder and file path falling under the C:\\ directory (main Windows directory) and storing web-based game data online perhaps under an account umbrella are the recommended mindsets.
4. **Memory Management**: Windows utilizes a few different memory management techniques with one being memory compression. Essentially, when memory gets to a certain point, Windows compresses it and stores it elsewhere until it needs to be delved into, then is compressed again after the delving is done.
5. **Distributed Systems and Networks**: Choosing the proper architecture for distributed systems is key. More web applications are developed in Three-tier, so utilizing three-tier may be the most universal and most compatible. Being able to communicate data between devices and server will be foundational. Reliability is high since there is no single-point of failure. In other words, a task is split into pieces — or across various computers — so that it is done more quickly and easily. This depends on reliable communication to virtual server instances that end when their tasks are done.
6. **Security**: Encryption, encryption, encryption. Encryption is key in clientele security, and encrypting user data will make sure it is at least vastly more difficult to access. Storing user data separate from the user account or in folders with controlled access may also provide more stability and security. Utilizing VPNs — virtual private networks — or VPN-like connections will be very compatible with Windows and allow secure information protection, especially when dealing with private and public networks.

## Recommendations 2.0 (Project Three)

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:** In my previous analysis, Windows was selected as the recommended operating system. Following further analysis and research, the decision remains the same recommending the Windows operating system. It is the most compatible operating system with high user-friendliness and popularity, but beyond these facets its server set-up with memory management, a long lifespan, and immense multitasking capabilities make Windows desirable for vast connectivity with users.
2. **Operating Systems Architectures:** While Mac and Linux can handle memory. Windows’ two architectures — 32-bit and 64-bit — address memory in a more universally compatible way while also allowing a dual choice between 4GB memory allotment (32-bit) and a theoretical 16EB memory allotment (64-bit).
3. **Storage Management:** Previous analysis recommended game data being stored locally on a Windows machine and web-based game data being stored online in a Windows server network. This recommendation remains following deeper analysis. Windows drive-based system for file storage benefits greatly from some local storage coupled with server storage for maximum compatibility.
4. **Memory Management:** Windows utilizes several different memory management techniques, especially throughout the evolution of the operating system. One such technique is memory compression where, essentially, Windows compresses memory and stores it elsewhere until it is requested by the user, re-transferred and opened fully, and then compressed and stored again on a repeating cycle until the device is turned off.
5. **Distributed Systems and Networks:** Initial analysis revealed that Three-tier is, perhaps, the most implemented distributed systems architecture. For *Draw It or Lose It* to be the most compatible and universal with existing distributed systems and networks, Three-tier is still recommended for utilization. There is no single point of failure in Three-tier as tasks are split into pieces so that they are done more quickly and safely; because of this, reliability and efficiency are high. These tasks are handled by virtual server instances that end when the task is complete, further adding to security in these matters.
6. **Security:** Encryption is, arguably, one of the most important implementations for security for users and developers. Encrypting user data immensely increases the difficulty to hack into the data and expose it. A smart manner of bolstering encryption is storing the user’s game data separately from the user’s game account and slapping controlled access on this location. This will not only multiply security but also stability in storing the data. To further this secure mindset, *Draw It or Lose It* should utilize VPNs, or virtual private networks. Windows is the operating system that is most susceptible to malware due to its reigning popularity, commonality, and compatibility as most software is made *for* Windows. Windows more properly uses VPNs than other operating systems and using VPN-like connections would not only allow secure storage of user data but also high compatibility and access on Windows machines/Windows servers, particularly if a user wishes to use a public network — or private, for that matter.