

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)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/28/21 | Aaron Fehir | Create software development document. Add executive summary, design constraints, and domain model narrative. |
| 2.0 | 04/11/21 | Aaron Fehir | Evaluate operating platforms. |
| 3.0 | 04/25/21 | Aaron Fehir | Recommendations updated. |

**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 currently maintains a popular Android app call *Draw It or Lose It*, which is modeled after the popular hit TV show *Win, Lose, or Draw* except that the images generated are drawn by a computer rather than human contestants. The company wants to create a cross-platform, web-based version of the Game. The present software development document lays out the design constraints and addresses potential solutions.

## [Design Constraints](#_2et92p0)

*Timeliness and Budget*

One set of constraints inherent to any software development project includes adhering to a timeline for development and delivering the final product with budgetary expectations. There was no specific guidance given by the client, but it is assumed throughout that developing a quality product quickly and inexpensively is a general priority.

*Development Team Competency*

On the one hand, we want to utilize existing developers to the best of our ability so as to not have to incur the expense of hiring additional team members. Having said that, we must also be aware that some aspects of the project may simply be beyond the capabilities of the existing team. In the latter case we will need to decide whether to outsource or conduct additional hiring depending upon the need. As it stands currently, we are hopeful that the current team can handle all aspects of the project without resorting to such measures, but we leave them on the table as last resort.

*Licensing and Copyright Concerns*

The only potential concern here is that the developed game is not so close to the *Win, Lose, or Draw* so as to incur claims of copyright infringement. This is not a serious concern, however, as the Android version of the game that is currently in existence has incurred no such difficulties. But it is something to be cognizant of as development for the web-app moves forward.

*Legal and Regulatory Compliance*

There are no known legal or regulatory compliance requirements.

*Programming Language Constraint*

The web-app will be developed using Java. As Java is also the language in which the Android version of the game was built, a smooth transition is expected.

*Operating Platform Constraints*

The application should be able to run within all major web-browsers on any operating system in which those browsers run. The most popular browsers, in no particular order, are Apple Safari, Google Chrome, Mozilla Firefox, and Microsoft Edge. The single most widely used of these is Chrome. If it should be necessary to limit the project to any one particular browser, Chrome would be the obvious choice so as to be able to reach the broadest audience possible.

## [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)

*UML Class Diagram*

****

*Explanation and Object-Oriented Programming Principles*

In addition to the ProgramDriver and SingletonTester classes, the UML class diagram shows five main classes: Entity, GameService, Game, Team, and Player. The Entity class is the parent class and Game, Team, and Player are child classes that extend the Entity class. The GameService class is neither a parent nor child class, but is utilized by the Game class in order to start a single instance of the game in memory. The GameService class has no public constructor, utilizing a GetInstance() method instead. Each class of the other four classes has one public constructor with the parameters *id* and *name*.

There are several principles of object-oriented programming (OOP) on display in this UML diagram, including portability, inheritance, encapsulation, and polymorphism. Because the program is divided into several classes, debugging the source code is easier in the case of a problem. If there is a problem with a particular method, we know where to look. Likewise, the classes are portable in the sense that it is easy to copy them into other programs. This would not be as efficient if the code were not broken up into classes.

The program also shows inheritance. Game, Team, and Player are each objects with their own set of features and behaviors. But they are also a types of the Entity class. This demonstrates polymorphism as well. A particular instantiation of the Game class, for instance, is a game, but by the same token is also an entity. The Game class calls methods of the Entity class, modifying the toString() method by means of an override.

Finally, notice the use of plus and minus symbols in the diagram to indicate public and private respectively. This relates the OOP principle of encapsulation. The private variables are directly accessible only to that class. From outside the class they can only be set via the constructor and retrieved through the “getter” functions. The basic idea of encapsulation is to hide the mechanics of the program “under the hood.” Those elements should not need to be accessed directly by an end user of the program.

## [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** | Closed source OS, which means higher licensing costs.  OS is perfectly suitable from a technical standpoint, containing many more features than what is needed to host the application. | Open-source OS, which means lower licensing costs.  As Linux distributions come in many flavors and are highly customizable, the OS can be configured to contain all and only those features needed to host the application. | Closed-source OS, which means higher licensing costs.  OS is perfectly suitable from a technical standpoint, containing many more features than what is needed to host the application. | Not a suitable server-side choice due to low processing power and memory constraints. |
| **Client Side** | Irrespective of the OS, it is necessary for interactions between the server and the client to happen quickly and without undue strain on system resources. REST API is a structure that can help facilitate this need. MacOS supports most major web browsers and provides a framework for secure data transmission, but not all file types are supported. | Linux also lacks support for some media file types (particularly proprietary types), which limits options and can lead to additional development time. Support for web browsers is also more limited, but some of those are compatible with the REST API. | Windows supports the widest variety of file formats and supports all major web browsers. No major issues utilizing the REST API. | Processing capability is limited, so some features of the game may not work well unless the game is specifically designed to utilize limited resources. As there is already an Android version of the game available, this is likely not a concern for us. |
| **Development Tools** | The program will be written in Java, which is a cross platform language that can be compiled once and then run on any major OS. Eclipse is a full-feature IDE that supports java and is open source, which will help keep costs down. | Supports Java and Eclipse. | Supports Java and Eclipse. | Android supports Java. iOS does not. Developing in a mobile environment is not recommended. The IDE’s available are extremely limited in the features they offer. |

## 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:**

It is recommended that the Gaming Room utilize serverless architecture running Linux. Serverless architecture is a cloud-based solution. It offloads the maintenance and storage of the server to a third-party, such as Amazon AWS. This will help with scaling. If the game becomes very popular or if we decide to add new image packages, we will need to adjust our storage requirements accordingly. Cloud-based server services can take care of this for us automatically on an as needed basis. Likewise, should we remove old image packages or the game dips in popularity, we can adjust down just as easily, making this a lower risk option than purchasing physical storage and maintaining local servers.

**2. Operating Systems Architectures:**

In a traditional three-tier architecture, there is a middle layer that communicates between the user interface and the data store. Opting for a serverless solution allows this layer to be modularized, which also helps with scale.

The Linux OS itself is modular as well. Beyond the kernel, separate modules are responsible for separate functions, which makes efficient use of system resources.

**3. Storage Management:**

In order for the game to run smoothly, it will be best for the entire image library to exist in storage. At 8 MB each, the 200 images will require 1600 MB of space in storage. Any text or image files that are part of the user interface will also need to be stored, as well as the source code itself.

A technique that can be utilized to manage the storage of these files is storage virtualization. Storage virtualization allows the data to be stored in multiple containers while using a structured array to make it appear to the system as if there is just one large pool of data (Taylor, 2020).

**4. Memory Management:**

One consideration will be the number of users participating in the game at any given time. The more active users there are, the more memory will need to be utilized to maintain the data on those users. We also know that there will be 200 8MB photos stored in the file system. How these photos are stored, retrieved, and rendered will also impact the amount of memory needed for the game’s operation.

A solution that can help to address both of these concerns is the use of virtual memory. As main storage is limited and secondary storage more readily available, virtual memory is a technique that places objects in secondary storage, but makes them appear as if they are in main storage (“Virtual Memory”, 2019). This method utilizes both hardware and software. By mapping logical memory addresses onto physical addresses that are dynamically allocated at runtime, virtual memory technique allows processes to be quickly swapped in and out of main memory as needed, writing to whatever chunk of physical memory is most accessible at the time. Without this technique, objects would need to be assigned a persistent physical location and then accessed at that location even when it’s not easily accessible.

Virtual memory also allows for processes to be broken up into segments using a system called paging. The advantage here is that not all of the data needs to be stored in one physical location, but can nonetheless be accessed via a single logical memory address (“Virtual Memory”, 2019).

**5. Distributed Systems and Networks:**

A serverless architecture will generally utilize multiple physical services, requiring load balancing on the back end. This is one more advantage of opting for a cloud-based solution as we will not need to maintain this balance ourselves.

We also must think about how our game will be distributed to clients. For this, we will utilize REST API to exchange information over HTTP. Because REST API is stateless, messages will be presented uniformly across all clients.

**6. Security:**

Finally, we must consider security. While the data we collect and store is not especially sensitive in the broad scheme of things, we will nonetheless need to take various security measures. The purpose of these measures is multi-fold. On the one hand, we want to maintain user privacy. But we also want to make sure that only paid subscribers are able to utilize the gaming service, which protects our profits. Unless we protect our source code, there is also the potential problem of bootleggers or malware intrusions. For these reasons, we want take several standard security measures, such as requiring user authentication and a strong password policy, encrypting user data (especially payment information) and maintaining our copyright on the legal front.

**Works Cited**

“Virtual Memory in Operating System” (2019, August 16). Geeks for Geeks. Retrieved from <https://www.geeksforgeeks.org/virtual-memory-in-operating-system/>

Taylor, Christine. “Storage Virtualization” (2020, January 8). Enterprise Storage Forum. Retrieved from <https://www.enterprisestorageforum.com/hardware/storage-virtualization/>