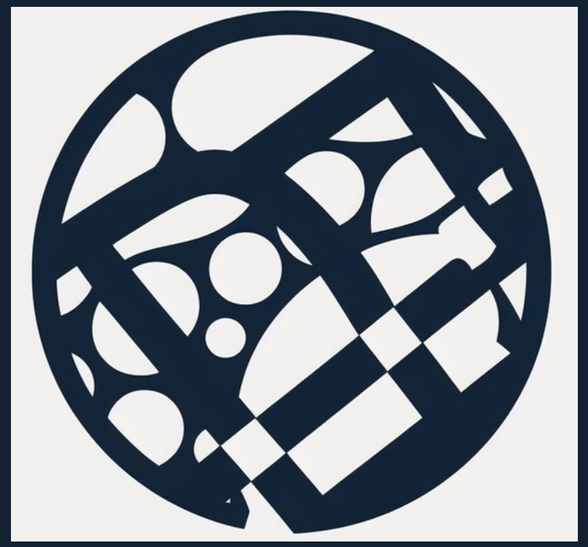
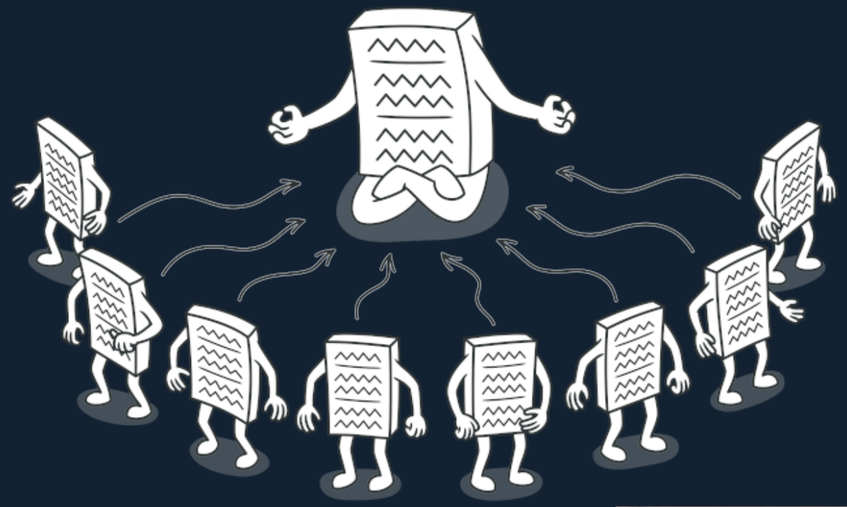
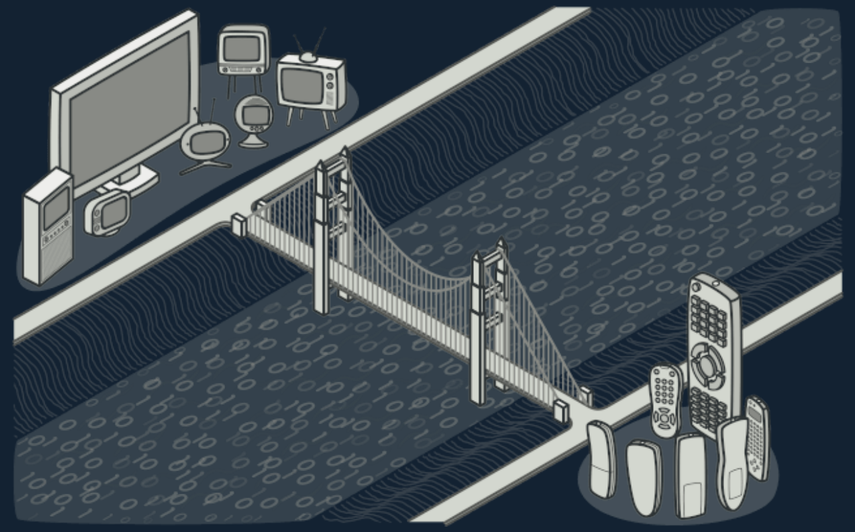
# DESIGN PATTERNS EARTH



MARCO SANTIAGO CANCHÉ MAY
JOSÉ LUIS GUTIÉRREZ COUOH
RODRIGO JOAQUÍN PACAB CANÚL
ORLANDO ISAÍAS RODRÍGUEZ COUOH
CARLOS ISREAL RUZ RUZ

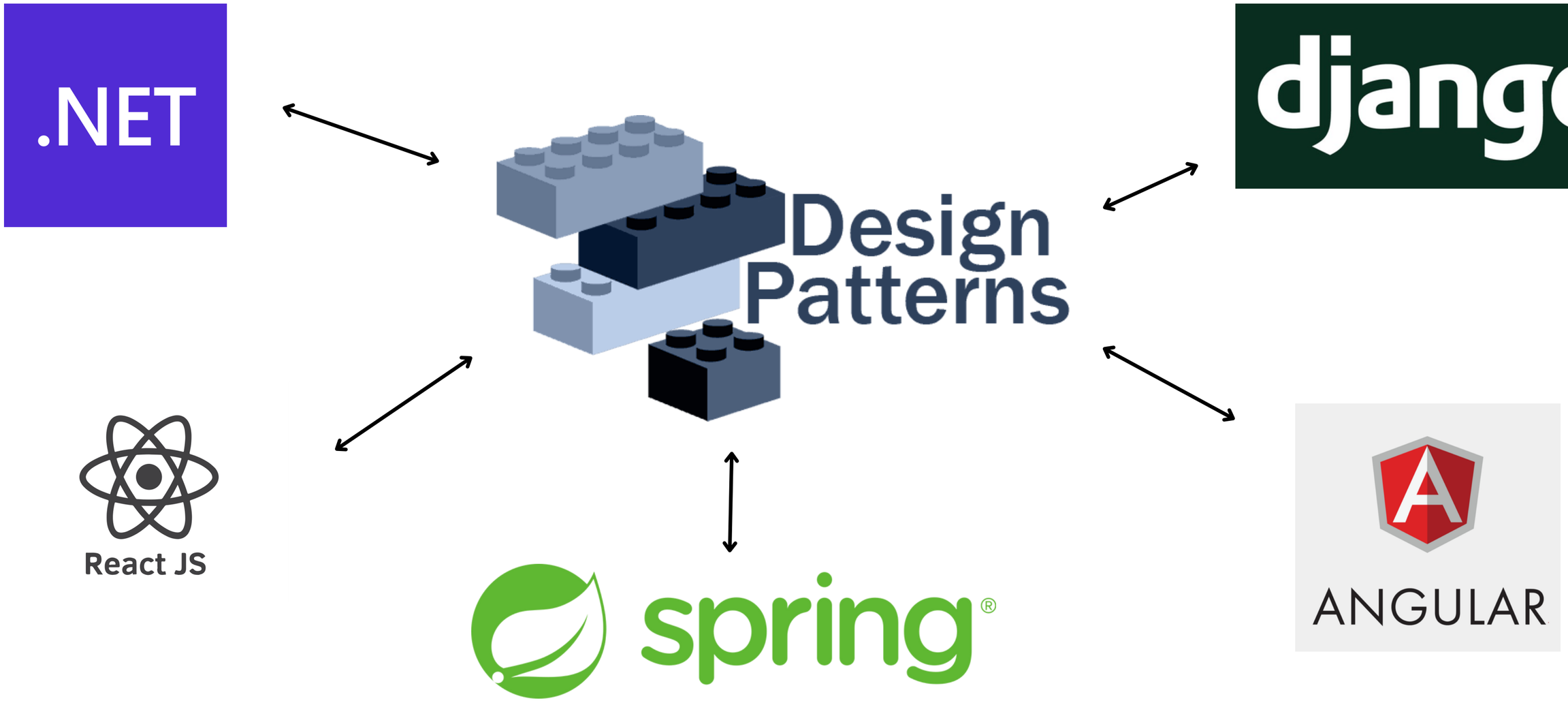
# SOFTWARE DESIGN PATTERNS





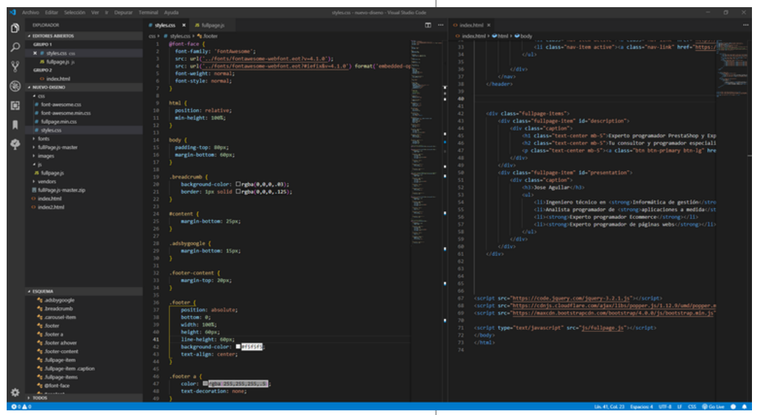

# GOOD INFORMATION WITHOUT REAL EXAMPLES

# OUR IDEA



# SYSTEM OBJECTIVES

**Clear and Concrete Resources**



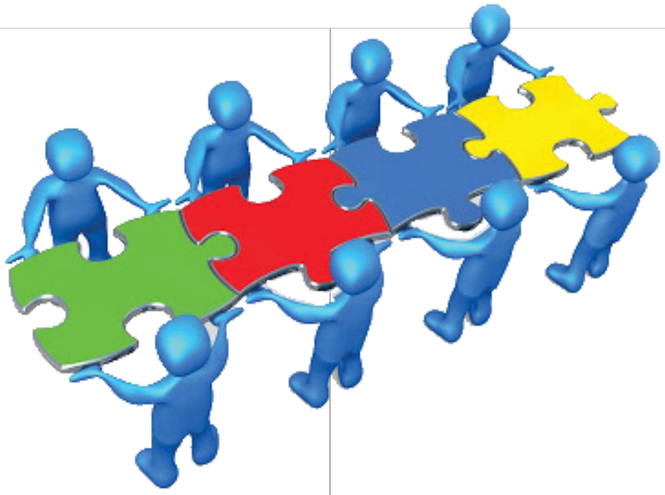
**Patterns with Real-Life Implementations**

SOLID

Design Principles

**Identify the SOLID principles that each pattern uses.**

# SYSTEM LIMITATIONS



**Does not allow collaboration**

**No updates in a row**

# DESIGN PATTERNS EARTH

## HOW WAS THE PROCESS?

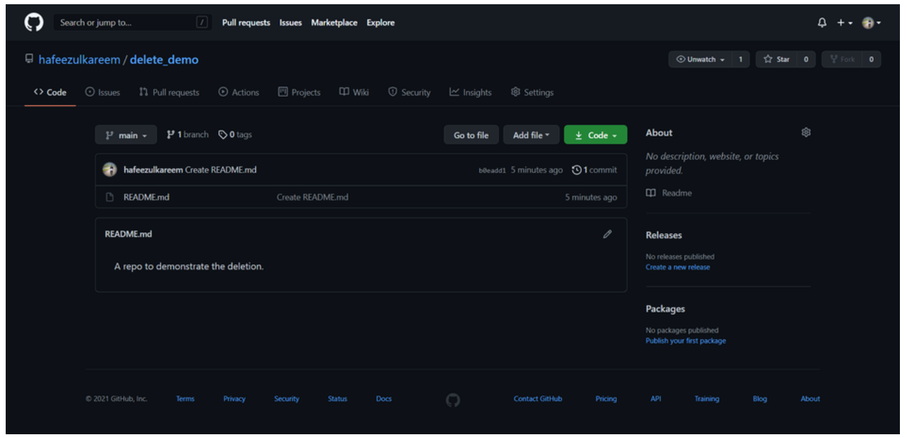
# 3 WELL-DEFINED PHASES

1. IDEAS PHASE - REPOSITORY STRUCTURE
2. RESEARCH PHASE
3. ARTIFACTS PHASE

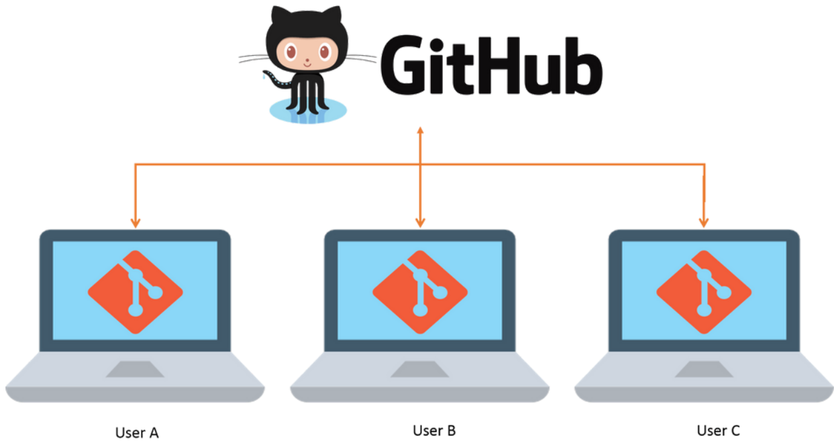
# 1- IDEAS PHASE

REPOSITORY STRUCTURE

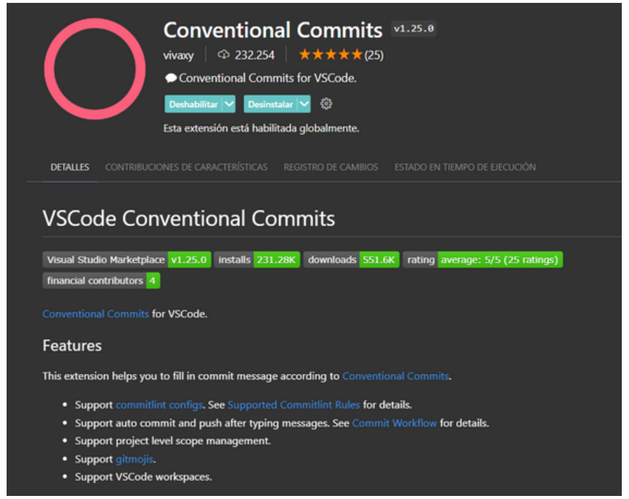
Approval for the project idea




Set up the repository structure




Protected branches




Good practices with commits

# 2- ·RESEARCH PHASE

AND TEAM ORGANITATION

RODRIGO PACAB

SCRUM MASTER

ISAIAS RODRIGUEZ

PRODUCT OWNER

MARCO CANCHÉ

SCRUM MASTER - PRODUCT OWNER

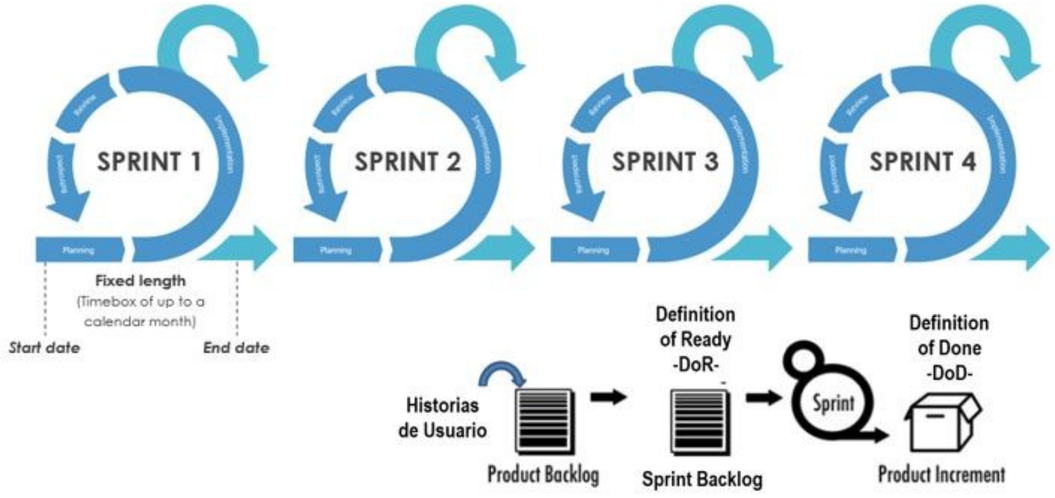
CARLOS RUZ

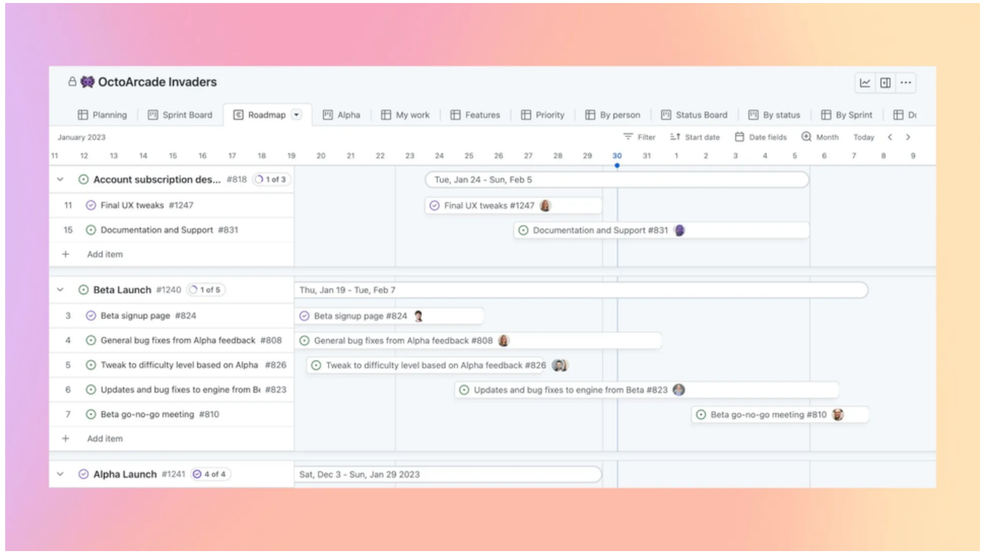
DEVELOPER

JOSÉ LUIS GUTIÉRREZ

DEVELOPER

# SCRUM


SPRINT 1
SPRINT 2
SPRINT 3
SPRINT 4
Planning
Sprint
Review
Implementation
Fixed length
(Timebox of up to a calendar month)
Start date
End date
Historias de Usuario
Product Backlog
Definition of Ready
-DoR-
Sprint Backlog
Sprint
Definition of Done
-DoD-
Product Increment


OctoArcade Invaders
Planning
Sprint Board
Roadmap
Alpha
My work
Features
Priority
By person
Status Board
By status
By Sprint
January 2023
Filter
Start date
Date fields
Month
Today
Account subscription des... #818 1 of 3
Tue, Jan 24 - Sun, Feb 5
Final UX tweaks #1247
Documentation and Support #831
Add item
Beta Launch #1240 1 of 5
Thu, Jan 19 - Tue, Feb 7
Beta signup page #824
General bug fixes from Alpha feedback #808
Tweak to difficulty level based on Alpha #826
Updates and bug fixes to engine from Beta #823
Beta go-no-go meeting #810
Alpha Launch #1241 4 of 4
Sat, Dec 3 - Sun, Jan 29 2023

# 3- ARTIFACTS PHASE

## Requirements

### Functional requirements

1. **Design pattern in software:** The software must implement a specific set of design patterns, including but not limited to, Singleton, Factory Method, Observer, Strategy, Proxy, Memento, Command, Mock Object, Repository ( or DAO), among others, as appropriate.
2. **Implementation Examples:** In each design pattern example, the software code must be shown with specific examples (observations and comments) that show the application of said pattern in situations relevant to the project.
3. **Interaction between Patterns:** If possible, it should be demonstrated how the different design patterns interact with each other in the software to achieve a coherent and efficient design.
4. **Detailed Documentation:** Each design pattern shown must be accompanied by detailed technical documentation that explains its use, advantages, disadvantages and how it is integrated into the overall project.
5. **Compliance with Coding Standards:** The code that implements the design patterns must follow the coding standards (ours) defined for the acceptance of the project code.

### Non-functional requirements

1. **Usability:** The system must work on different platforms
2. **Availability:** The software must be available 24 hours a day, 7 days a week.

**We obtained these requirements**

# ARTIFACTS

## CLASS DIAGRAM

### First Iterarion of classes

> As this is the first version of the classes, they are currently few and somewhat general. With each iteration, they will be further developed and become more complex in their relationships

**1. `DesignPattern` Class:**

**Attributes:**

- `name` (Type: String): Stores the name of the design pattern.
- `description` (Type: String): Holds a detailed description of the pattern.
- `patternType` (Type: String): Indicates the category to which the pattern belongs.
- `purpose` (Type: String): Describes the purpose of the design pattern.
- `CodeExample` (Type: Code Example): Possibly a reference to an object of the `CodeExample` class related to the pattern.

**Methods:**

- Constructor to initialize the attributes.
- Methods to get and set each attribute.
- Method to display the complete information of the pattern.

**2. `CodeExample` Class:**

**Attributes:**

- `title` (Type: String): A descriptive title for the example.
- `description` (Type: String): A brief description of the example.
- `code` (Type: String): The example code related to the pattern.

**3. `DesignPatternFinder` Class:**

**Attributes:**

- `patternList` (Type: Array of references to `DesignPattern`): Stores a list of objects related to design patterns.

**Methods:**

- `searchByName(name: String)`: List of `DesignPattern`: Allows searching for design patterns by name.
- `searchByCategory(category: String)`: List of `DesignPattern`: Facilitates the search for design patterns by category (e.g., creational, structural, behavioral, etc.).

# ARTIFACTS

## USER STORIES

## User stories

### User Story 1 - Implementation of Design Patterns

As a project developer, I want the software to implement a variety of design patterns, including but not limited to Singleton, Factory Method, Observer, Strategy, Proxy, Memento, Command, Mock Object, Repository (or DAO), as appropriate.

### User Story 2 - Interaction Between Design Patterns

As a project developer, I want to demonstrate how different design patterns interact with each other in the software to achieve a coherent and efficient design.

### User Story 3 - Detailed Documentation

As a project developer, I want to create detailed technical documentation for each design pattern implemented in the software. This documentation should explain its usage, advantages, disadvantages, and how it integrates into the project overall.

### User Story 4 - Compliance with Coding Standards

As a project developer, I want to ensure that the code implementing the design patterns complies with the coding standards defined for the acceptance of the project.

### User Story 5 - Usability on Different Platforms

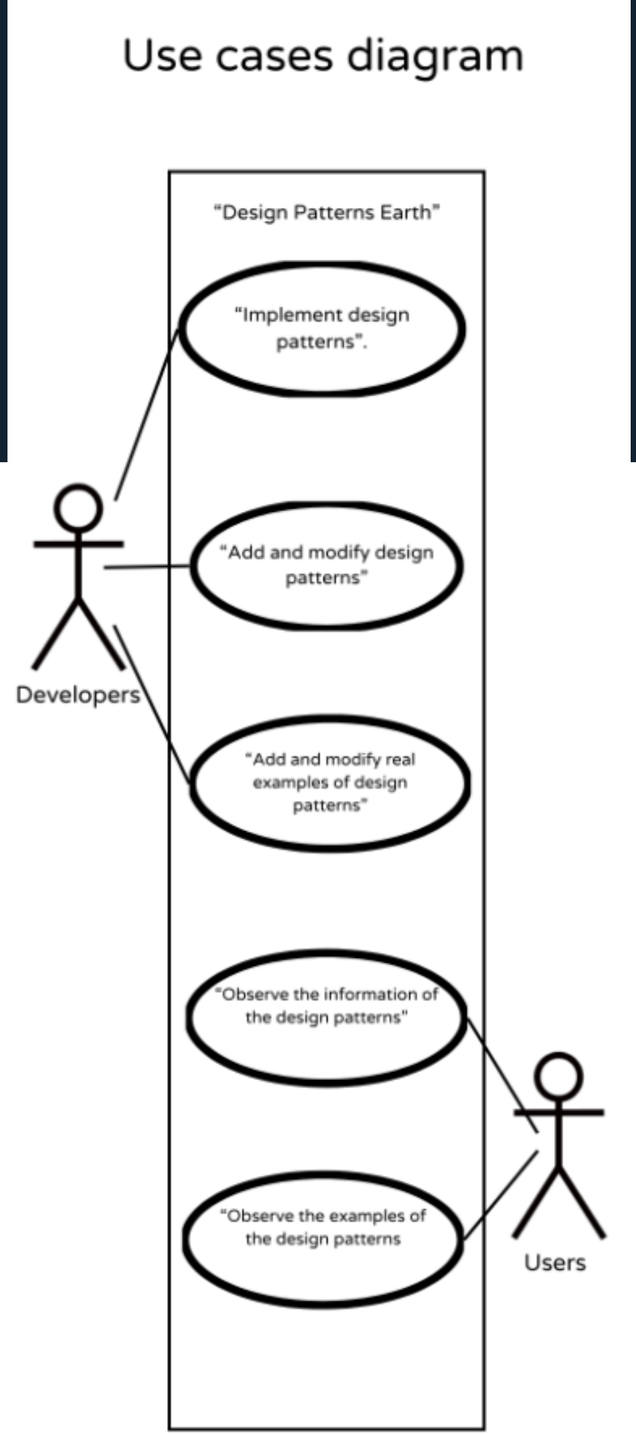
As a general user, I want the software to be usable on different platforms to ensure accessibility and convenience.

### User Story 6 - 24/7 Availability

As a general user, I want the software to be available 24 hours a day, 7 days a week, to ensure continuous availability.

# ARTIFACTS

## USES CASES DIAGRAM

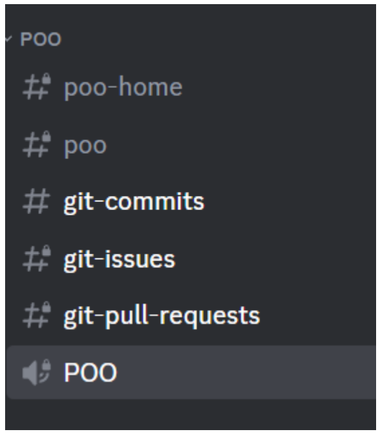


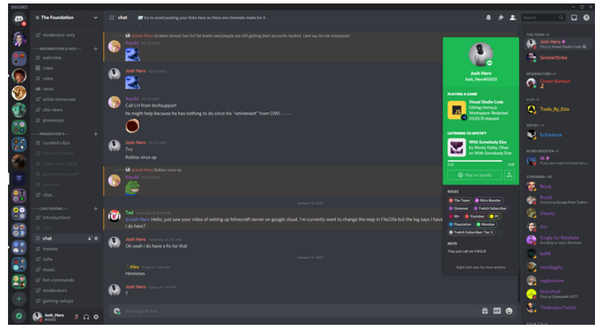
# PROCESS MANAGEMENT

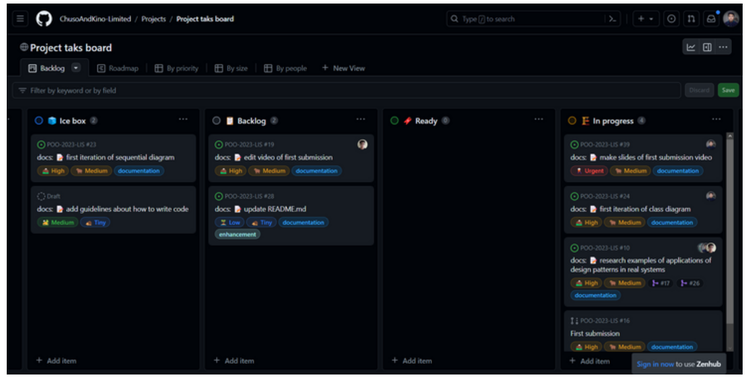
# RESULTADOS

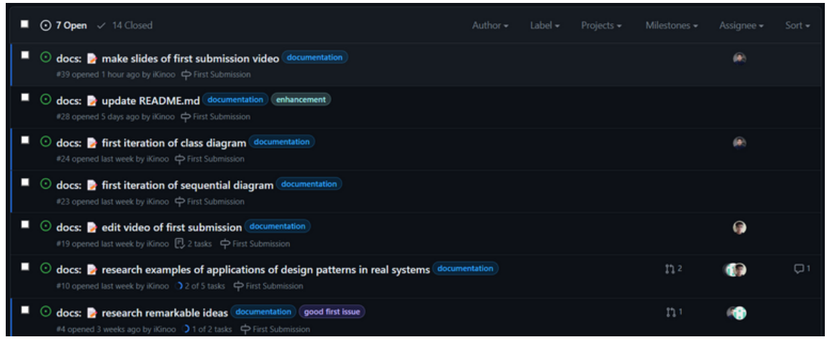
## ¿CÓMO PODEMOS MEDIR NUESTRO IMPACTO?

Identificamos nuestros  métricas principales y objetivos de negocio

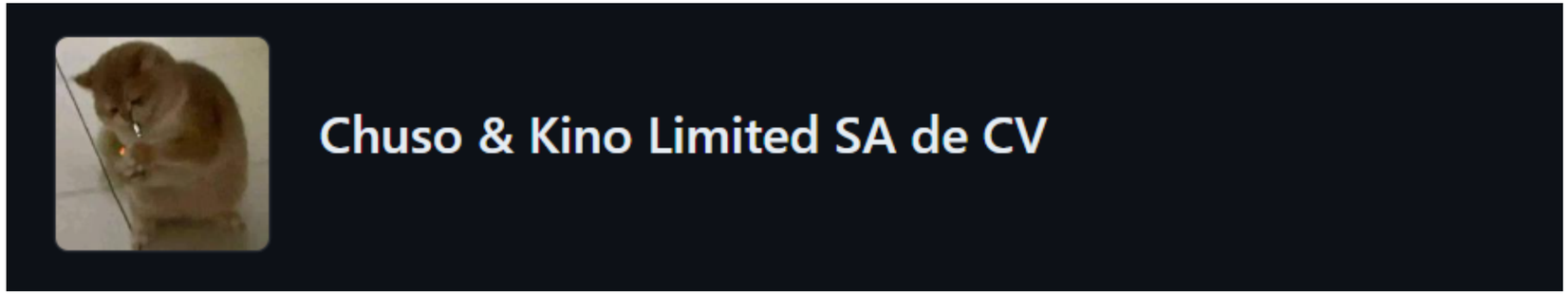




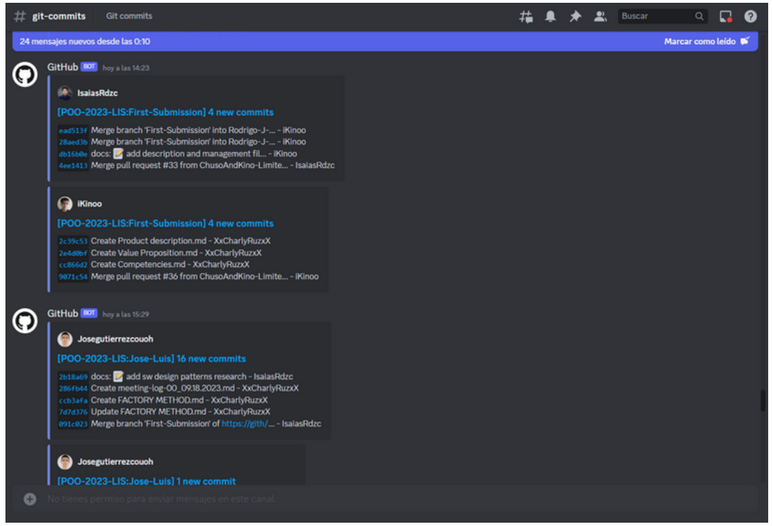



# WE NEEDED AN ORGANIZTATION

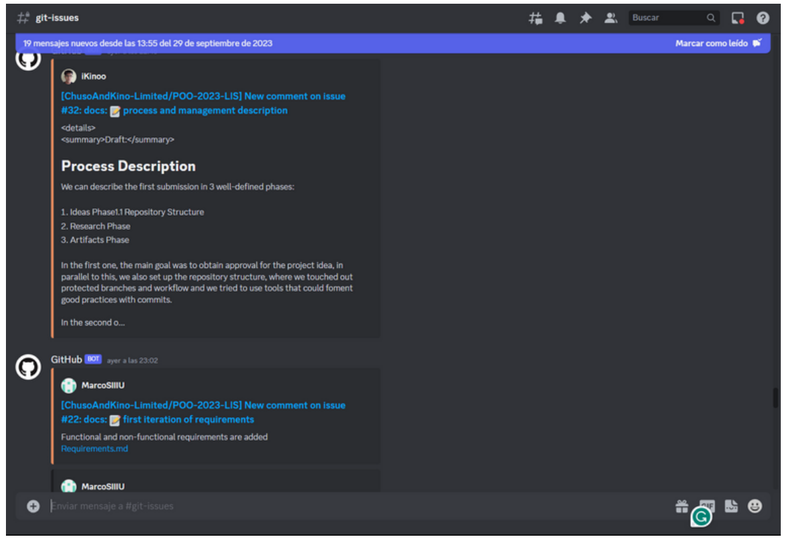



**More than one person has privileges**

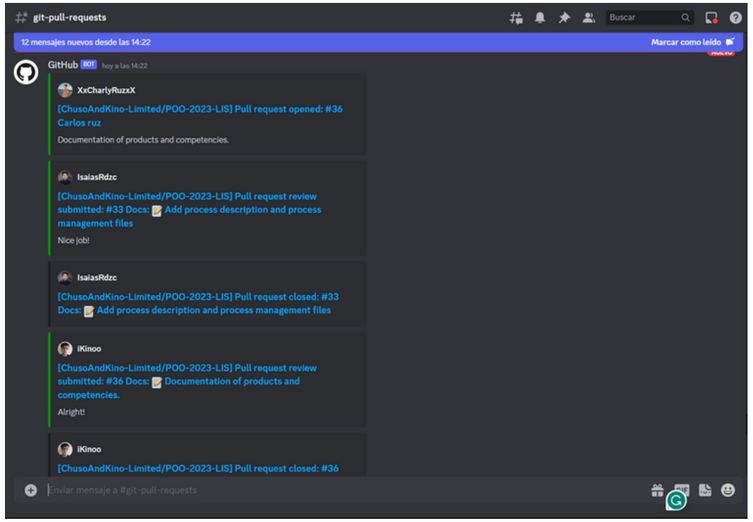
# DISCORD




For commits

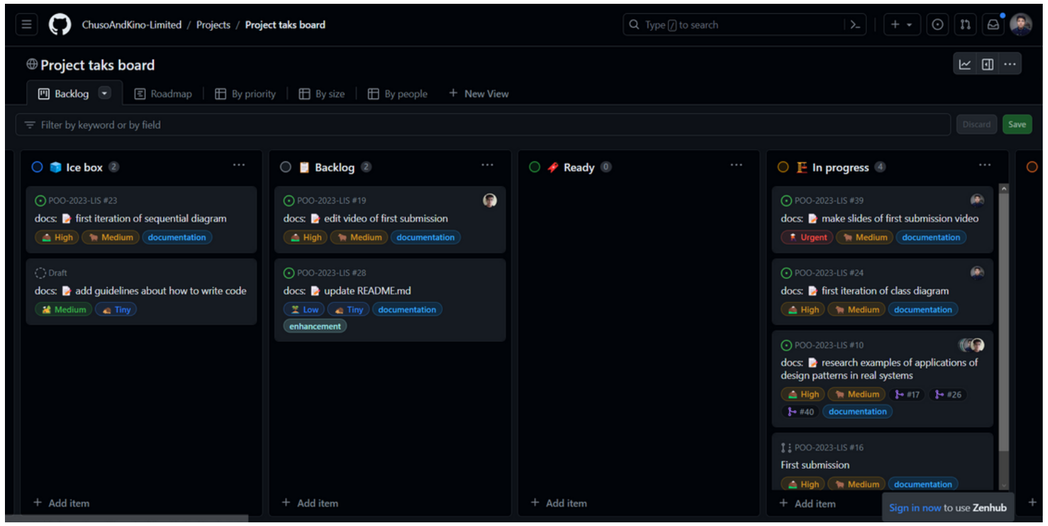



For issues

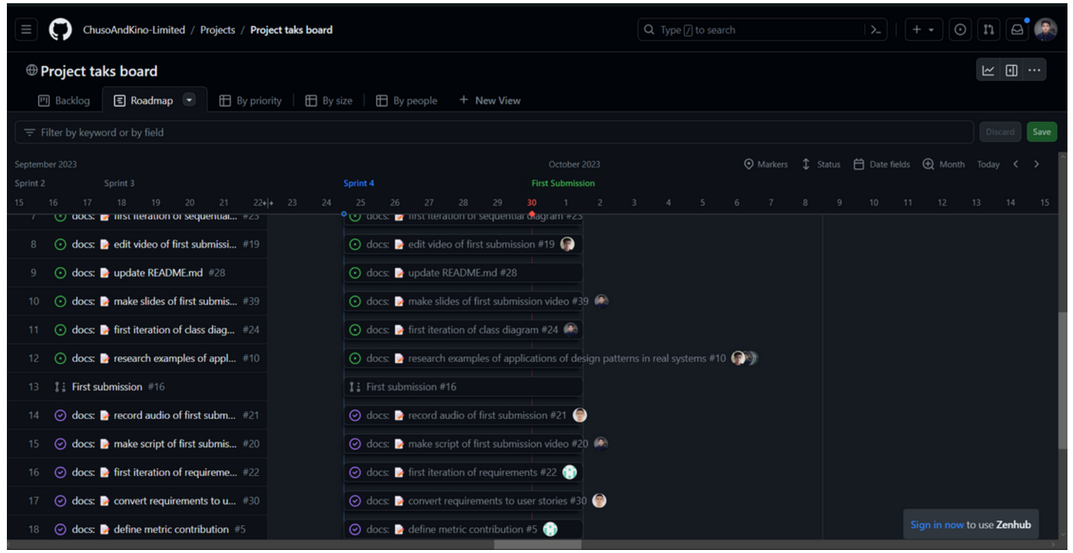



For pull request

# GITHUB PROJECTS

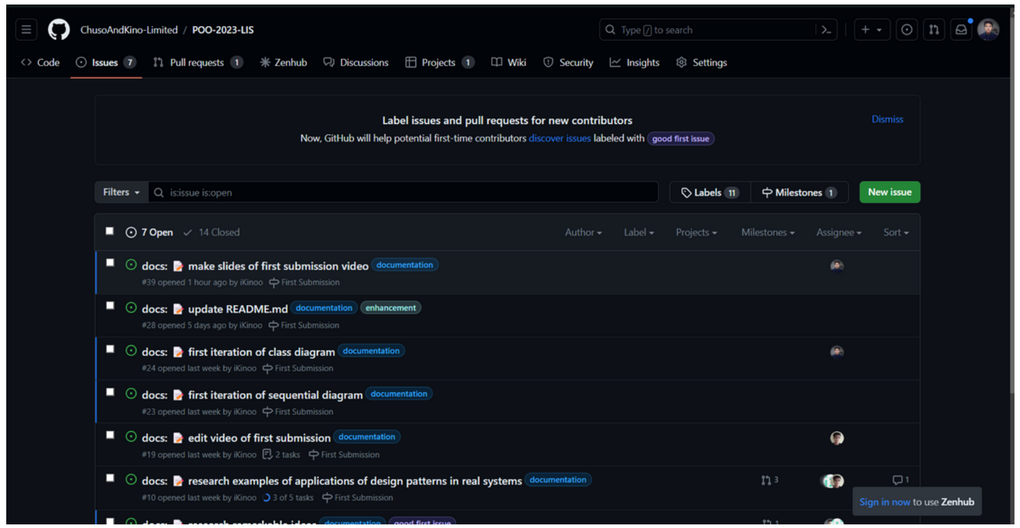



**Backlog with Kanban**

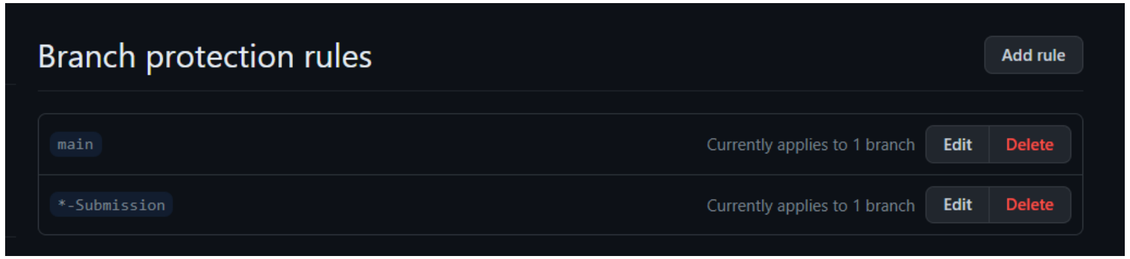



**Roadmap view (sprints)**

# GITHUB PROJECTS

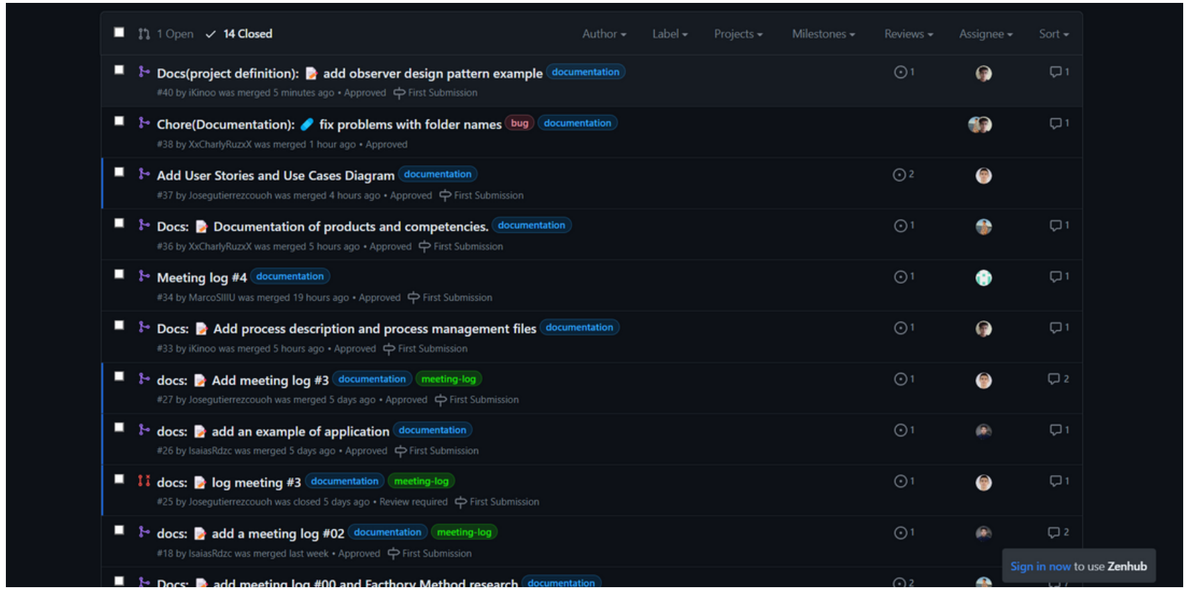


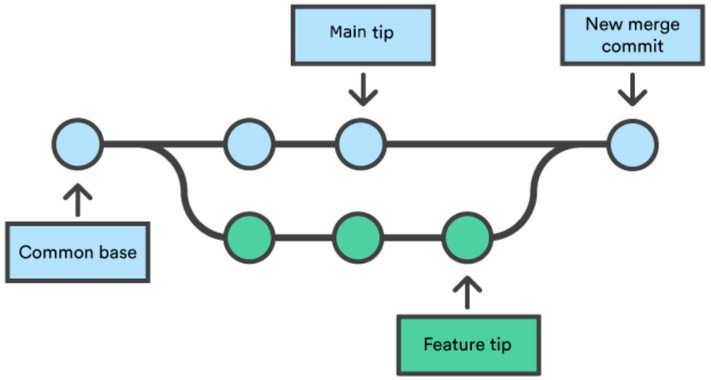

**Issue was manipulated as one task or a group of tasks**




**Protected branches**

# GITHUB PROJECTS




**A contributor could only merge their work into the first-submission branch through a Pull Request, and this Pull Request must be approved at least by one person**