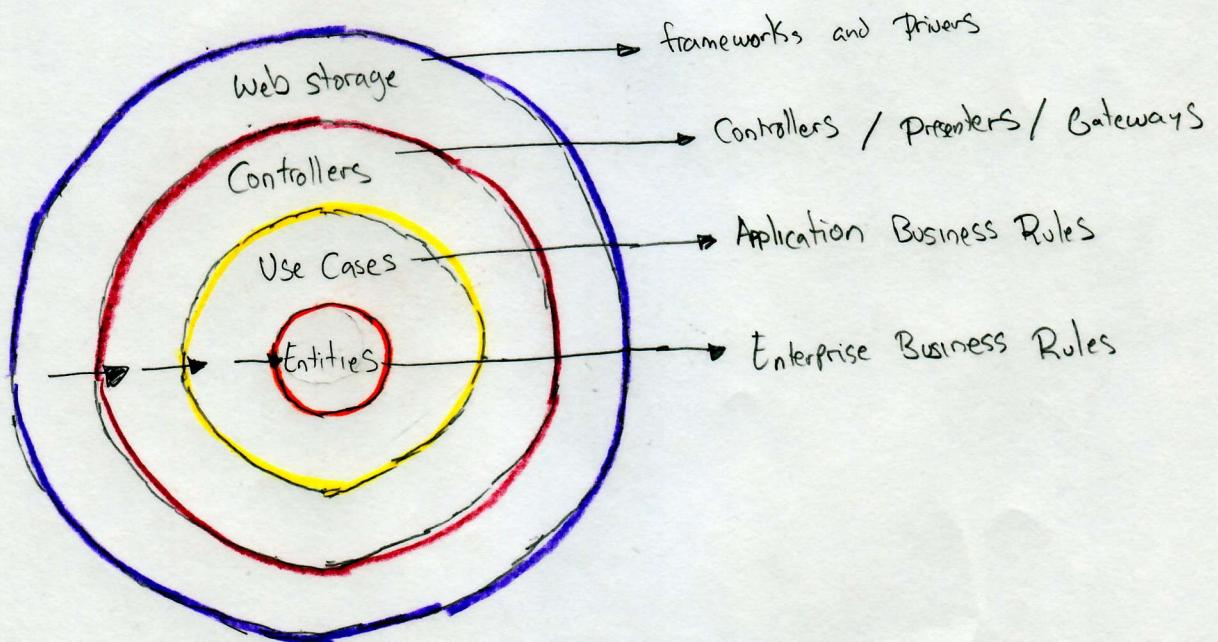
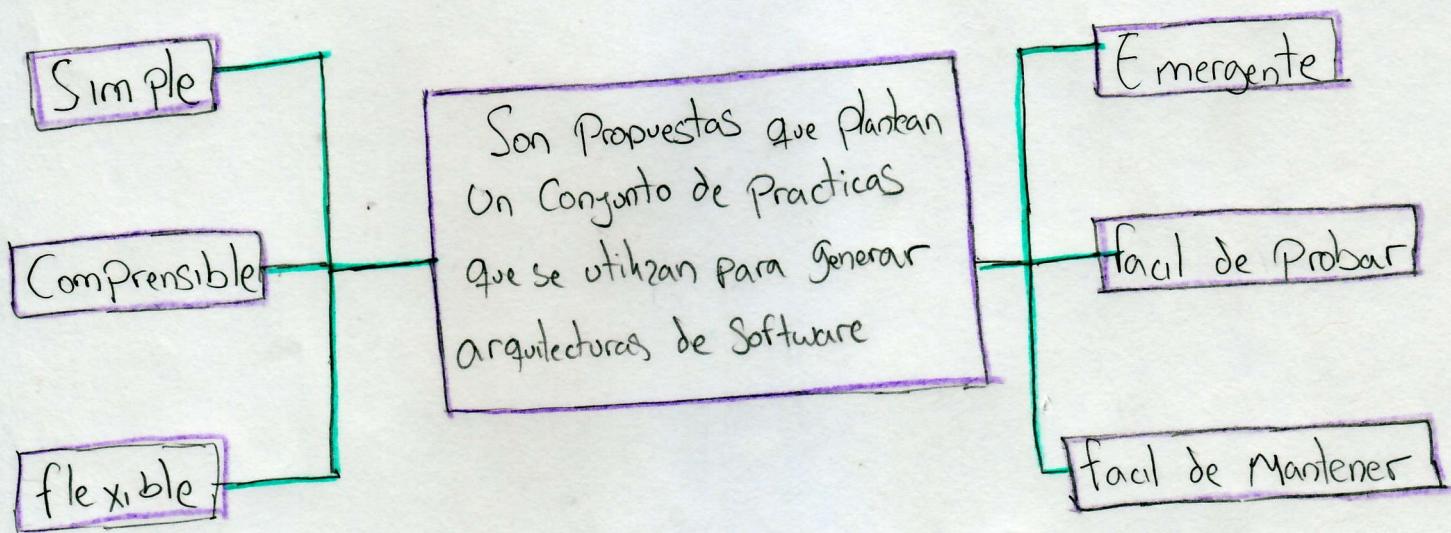


ARQUITECTURAS LIMPIAS



Arquitectura CLEAN para el Frontend

Con las arquitecturas limpias es posible generar sistemas de software que tengan las siguientes características:

Independencia de Marcos de trabajo o frameworks

- La arquitectura no depende de la existencia de una librería o funciones específicas.

fácil de probar

- Las reglas de negocio se pueden probar sin la interfaz de usuario, bases de datos o cualquier otro elemento externo.

Independiente de la interfaz de usuario

- Esta puede cambiar fácilmente, sin modificar el resto del sistema.

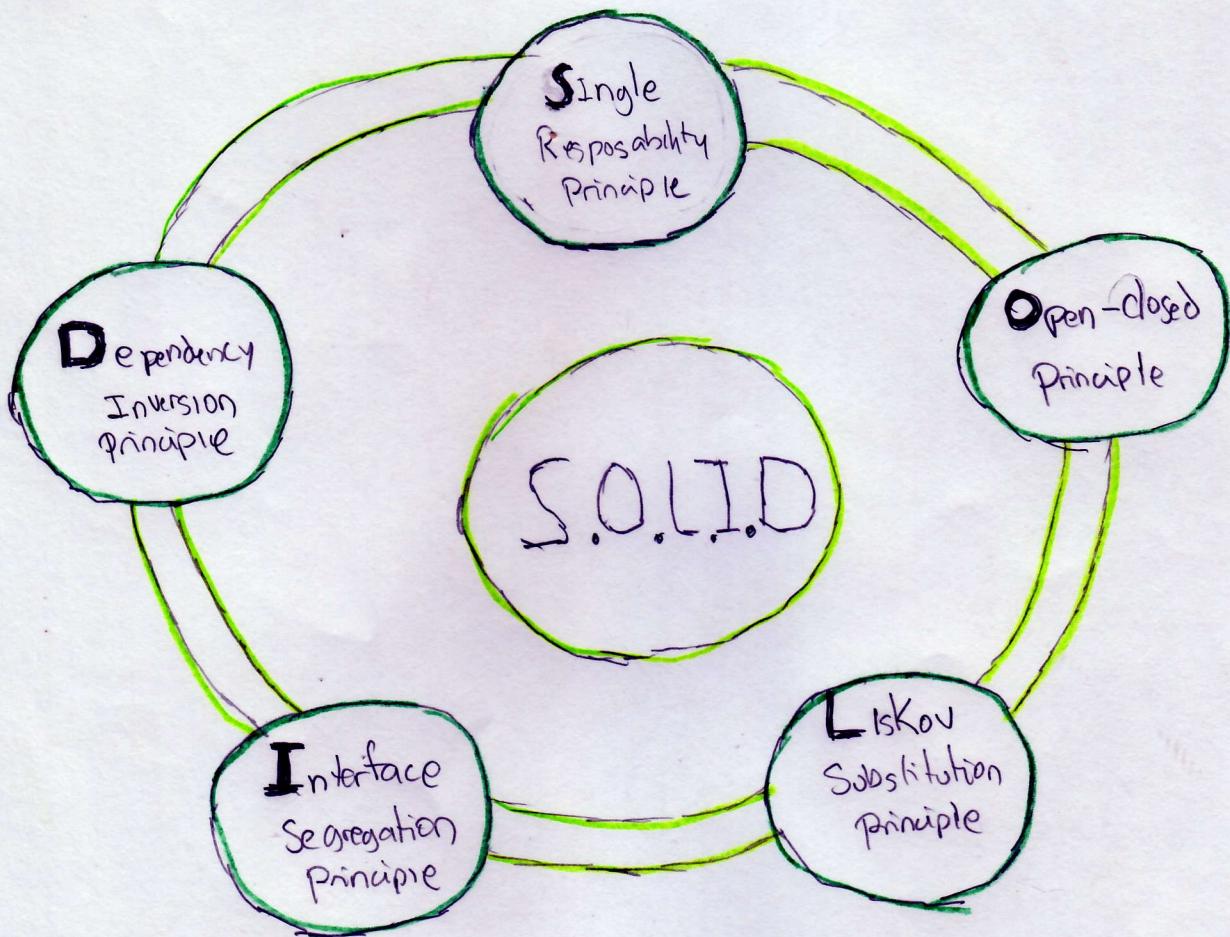
Independiente de la base de datos

- Puede cambiar Oracle o SQL server por MongoDB, BigTable, CouchDB u otra. Las reglas de negocio NO se encuentran vinculadas a la Base de Datos.

Independiente de cualquier librería externa

- Al igual con los Marcos de trabajo, es posible utilizar librerías externas las cuales pueden intercambiarse por otras sin afectar la funcionalidad del sistema.

PRINCIPIOS S.O.L.I.D



Acronimo S.O.L.I.D

- Simple responsibility principle → Principio de responsabilidad única
- Open-closed principle → Principio de abierto - cerrado
- Liskov Substitution principle → Principio de sustitución de liskov
- Interface Segregation principle → Principio de Segregación de interfaces
- Dependency Inversion principle → Principio de inversión de dependencias.

Principio de responsabilidad Unica

Se refiere a la responsabilidad unica que debiera tener cada Programa con una Tarea bien especifica y acotada

Principio abierto/cerrado

Toda clase , modulo , metodo ,etc .debera estar abierto para extenderse pero debe estar Cerrado para Modificarse.

Principio de sustitucion de liskov

Si la clase A es de un subtipo de la clase B , entonces deberiamos reemplazar B con A Sin afectar al Programa .

Principio de Segregacion de Interfaces.

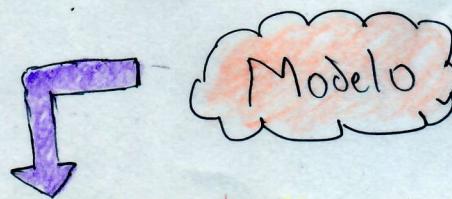
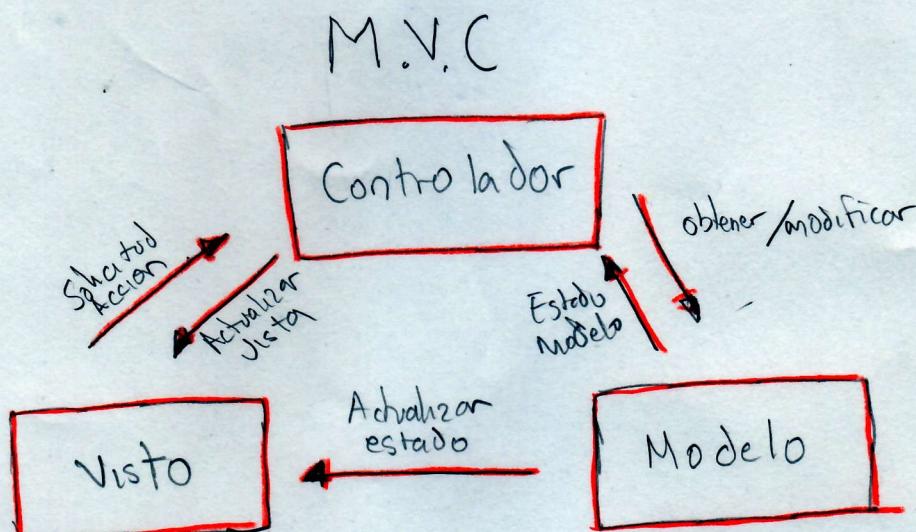
Ningun cliente deberia estar obligado a depender de los metodos que no utiliza .

Principio de inversion de dependencias

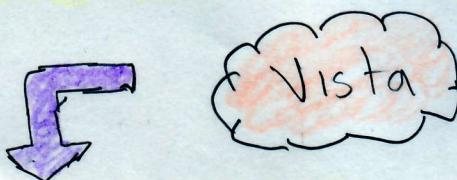
No debe existir dependencias entre los Modulos , en especial entre Modulos de bajo Nivel y de alto Nivel

PATRONES DE DISEÑO

Modelo Vista Controlador (M.V.C)



Contiene el conjunto de clases que definen la estructura de datos que se trabaja en el Sistema



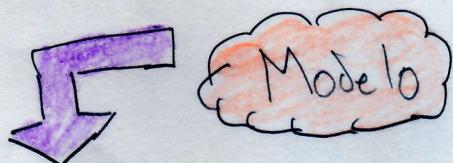
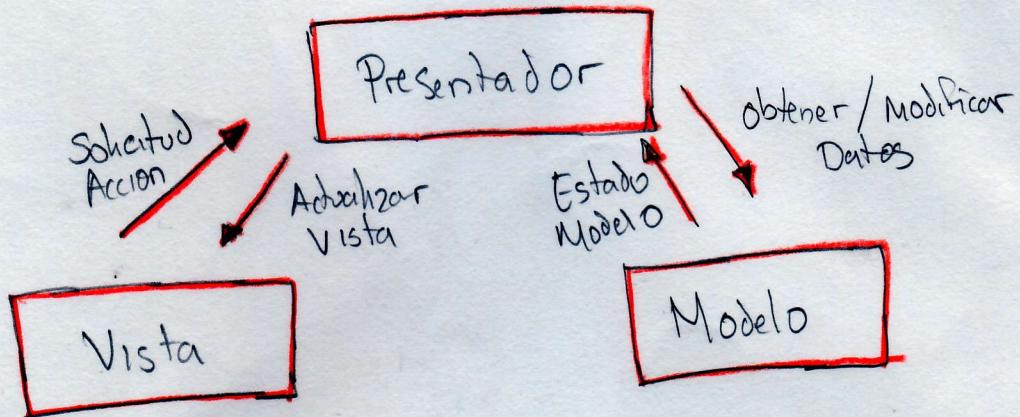
Contiene la interfaz de usuario de la aplicación



Capa intermedia entre la vista y el Modelo

Modelo Vista Presentador (M.V.P)

M.V.P



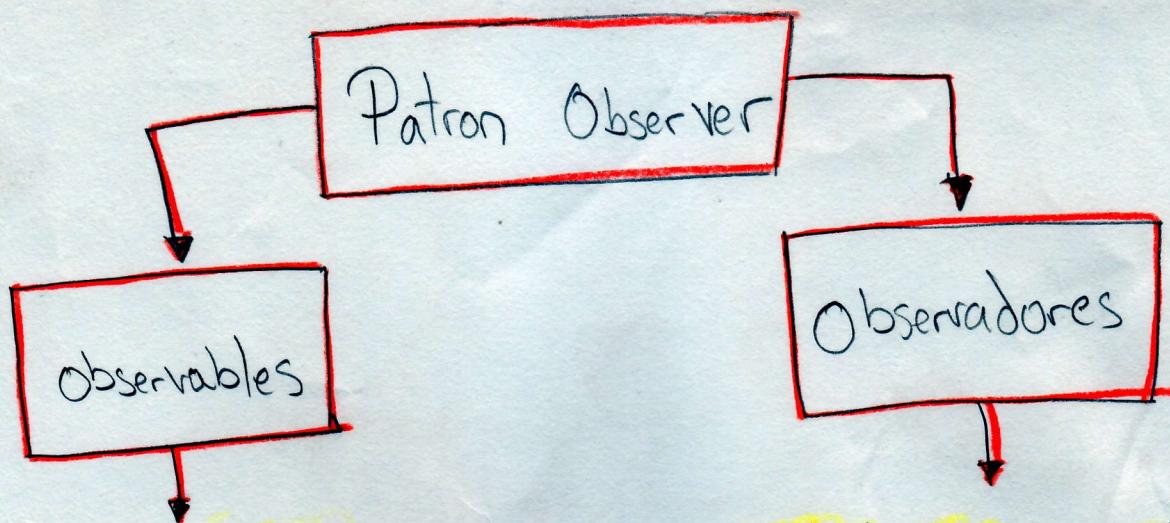
Capa encargada de gestionar los datos; su principal responsabilidad es la persistencia y almacenamiento de datos



La vista NO es una Activity o un fragment, simplemente es una interfaz de comportamiento de lo que podemos realizar con la vista.



Capa que actua como intermediaria entre el modelo y la vista.



Son objetos con un estado concreto, capaces de informar a los suscriptores suscritos al observable u que desean ser notificados sobre cambios.

Son objetos que se suscriben a los objetos observables y que solicitan ser notificados cuando el estado de estos observables cambie.