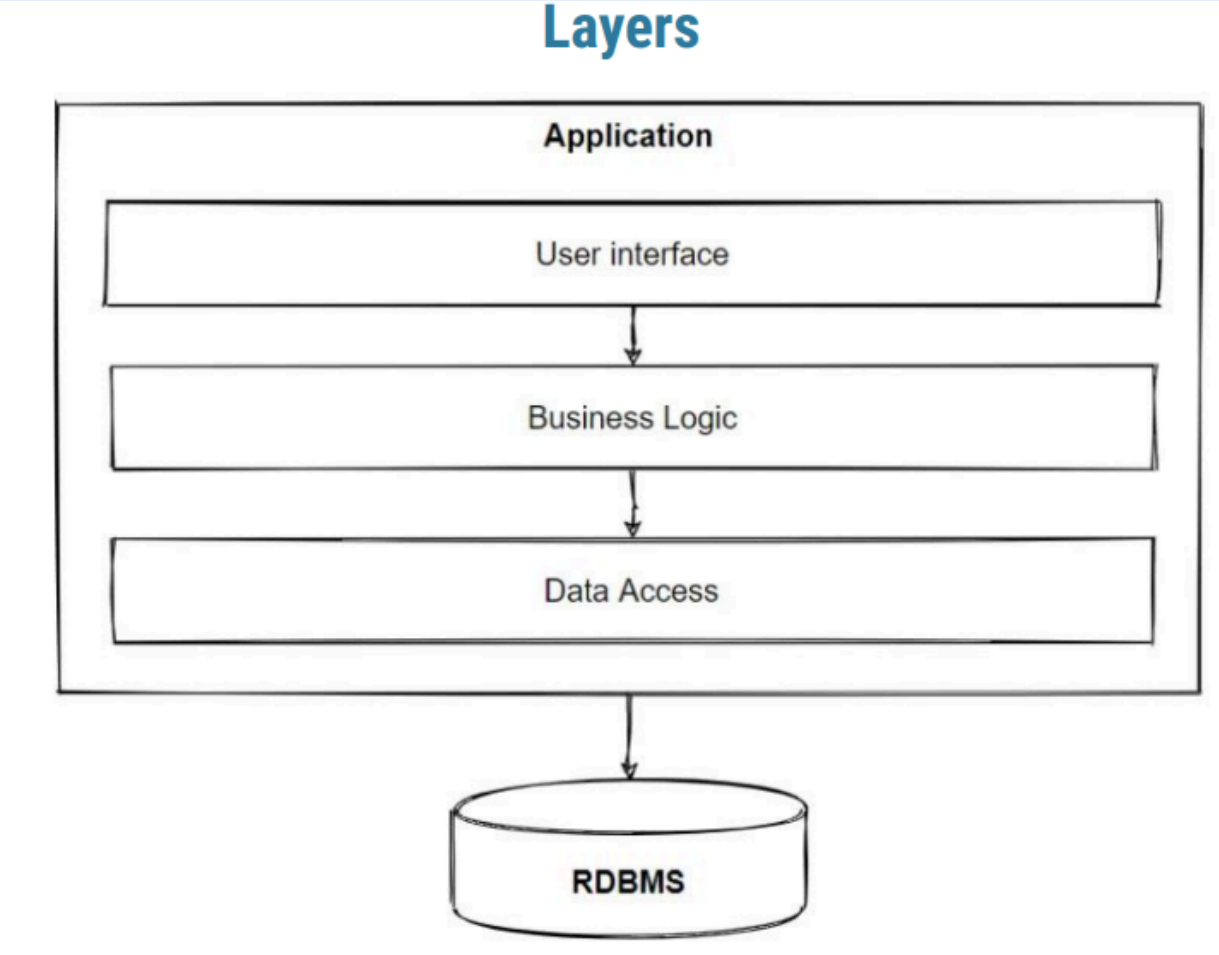
13. Patrones de aquitectura de software

Layers

- divide el sistema en diferentes niveles
- cada capa tiene una responsabilidad específica y solo se comunica con la capa inmediata superior o inferior.
- cada capa actúa como filtro o intermediario, de forma que los cambios en una parte del sistema no afectan directamente a las demás

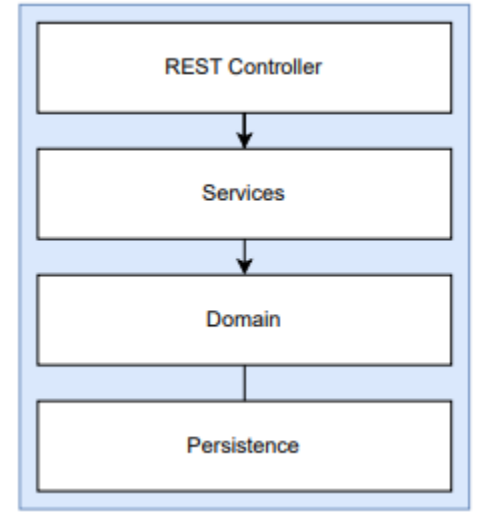
ejemplo genérico



donde:

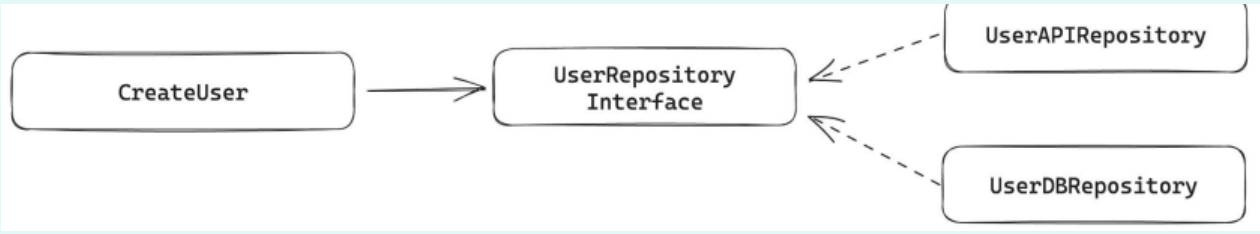
- la capa de interfaz representa lo que se muestra ante el usuario (vista)
- la capa business logic, representa lo concerniente al modelo de negocio (modelo)
- data access, representa lo referente al mantenimiento de datos persistentes (controller)

ejemplo particular: backend moderno



donde:

- REST controller: expone los endpoint HTTP para los clientes que la consuman (o el servidor web)
- Services: entidad que orquestas la interacción entre las diferentes entidades involucradas en un servicio expuesto. se encarga que el input se traduzca a operaciones en el dominio y que estas persistan (a través de una interfaz para la persistencia)
- Domain: define las entidades del negocio, reglas, validaciones, etc. Este ees el modelo del dominio.
- Persistencia: acceso a base de datos (repository). se interactua a través de interfaces.

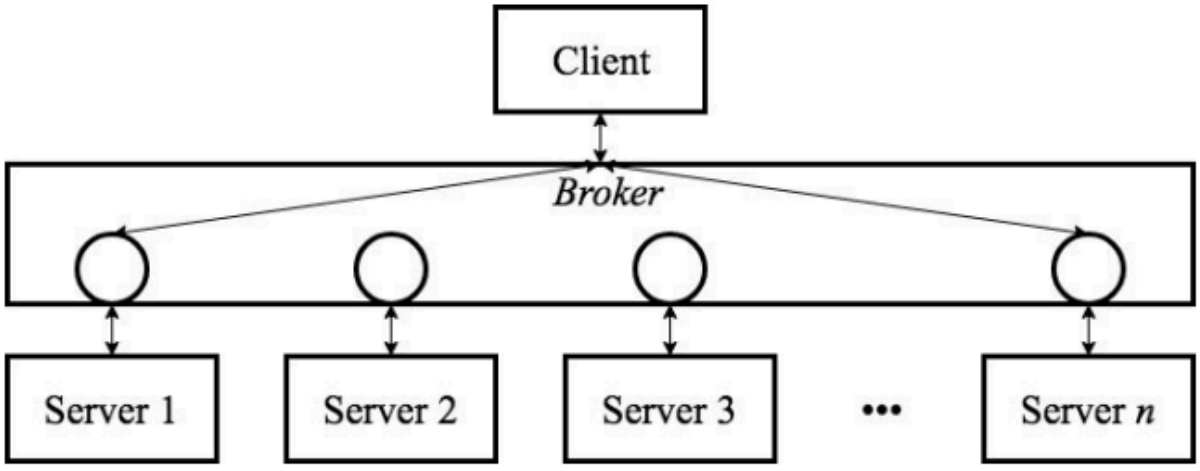


Broker

Se busca tener un intermediario (el broker) que coordina la comunicación entre los elementos del sistema.
Podemos:

- usar el broker para tener n servidores y que estos se puedan comunicar entre sí por medio del broker (basta con tener una conexión TCP cn el broker, los servidores entre sí se desconocen).
- Usar el broker para que el cliente se contacte indirectamente con el servidor (no conoce la dirección exacta del serv con el que se contacta).
  Por ejemplo, una app de inversión donde vos le das tu dinero, le dices qué acciones quieres invertir y el broker busca un match con los servicios que él contacta.

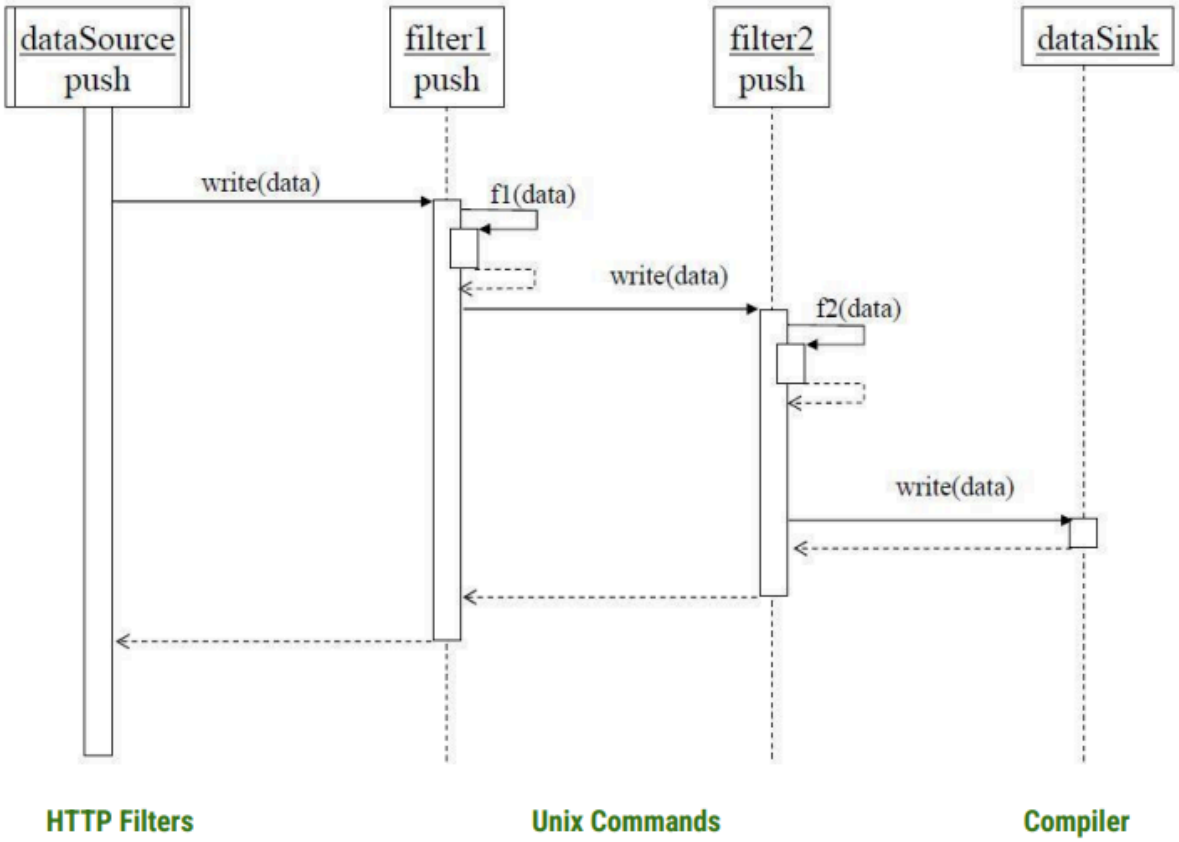
Broker



Pipe & filter

- dividimos una tarea compleja en una serie de filtros o mapeos, cada uno realiza una operación específica que se opera mientras la información "fluye" por los pipes.
- la información la fluimos de un filtro al siguiente

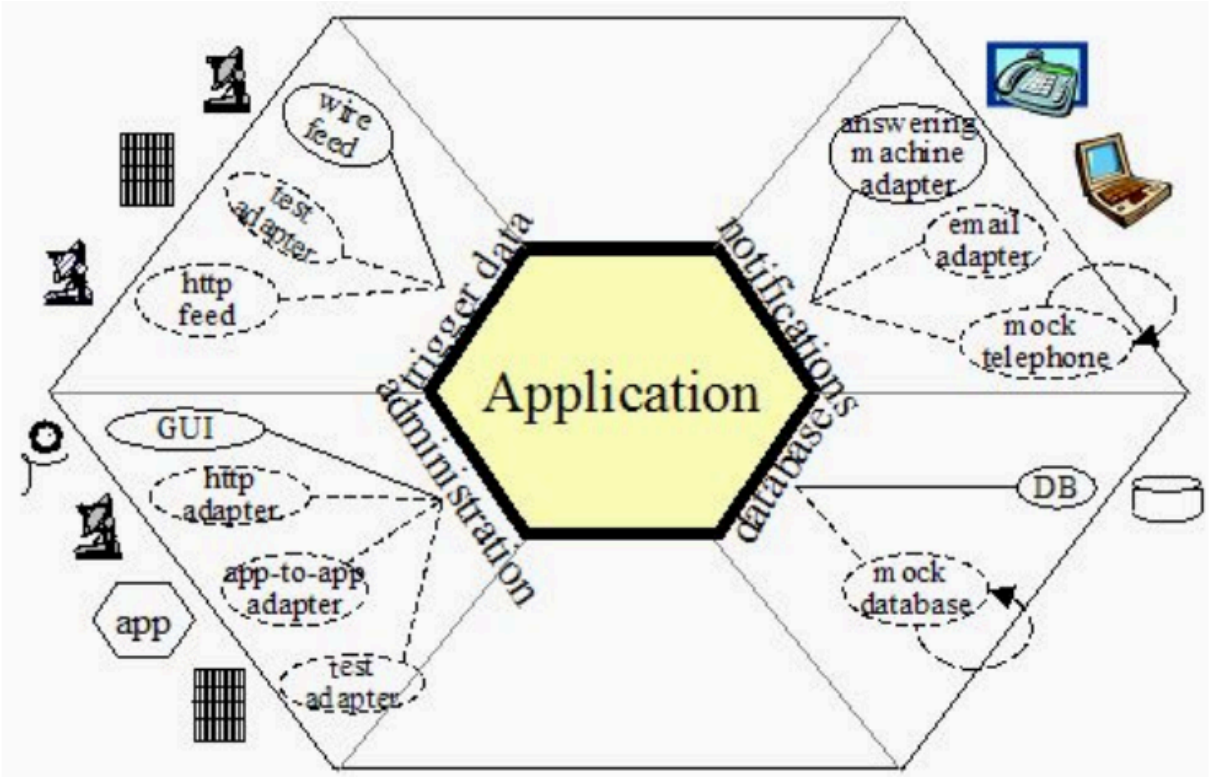
- cada pipe/filtro funciona como un módulo independiente (realiza una subentrada de la real, la procesa y deliverea su salida al siguiente pipe)

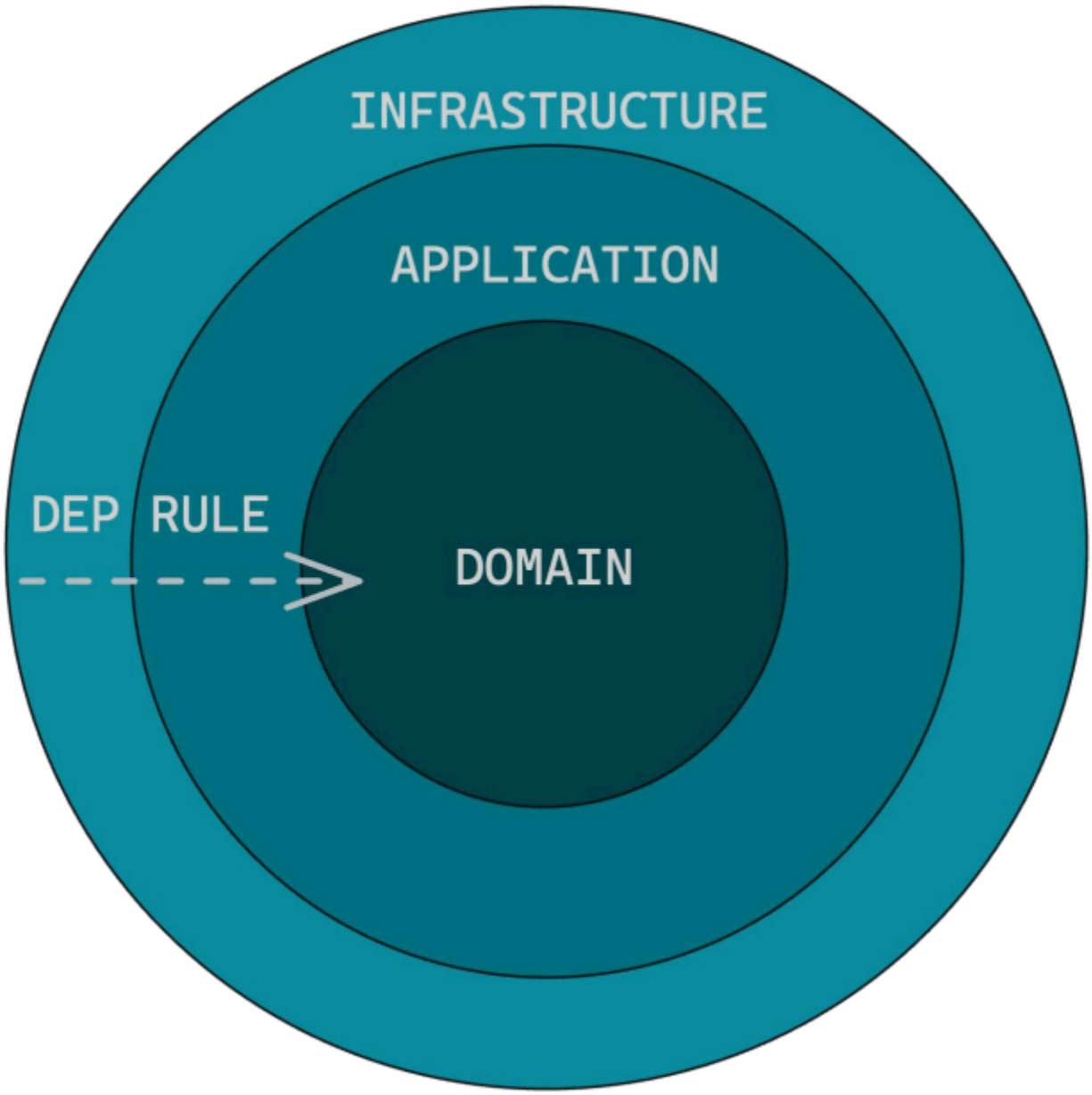


Hexagonal architecture (puertos y adaptadores)

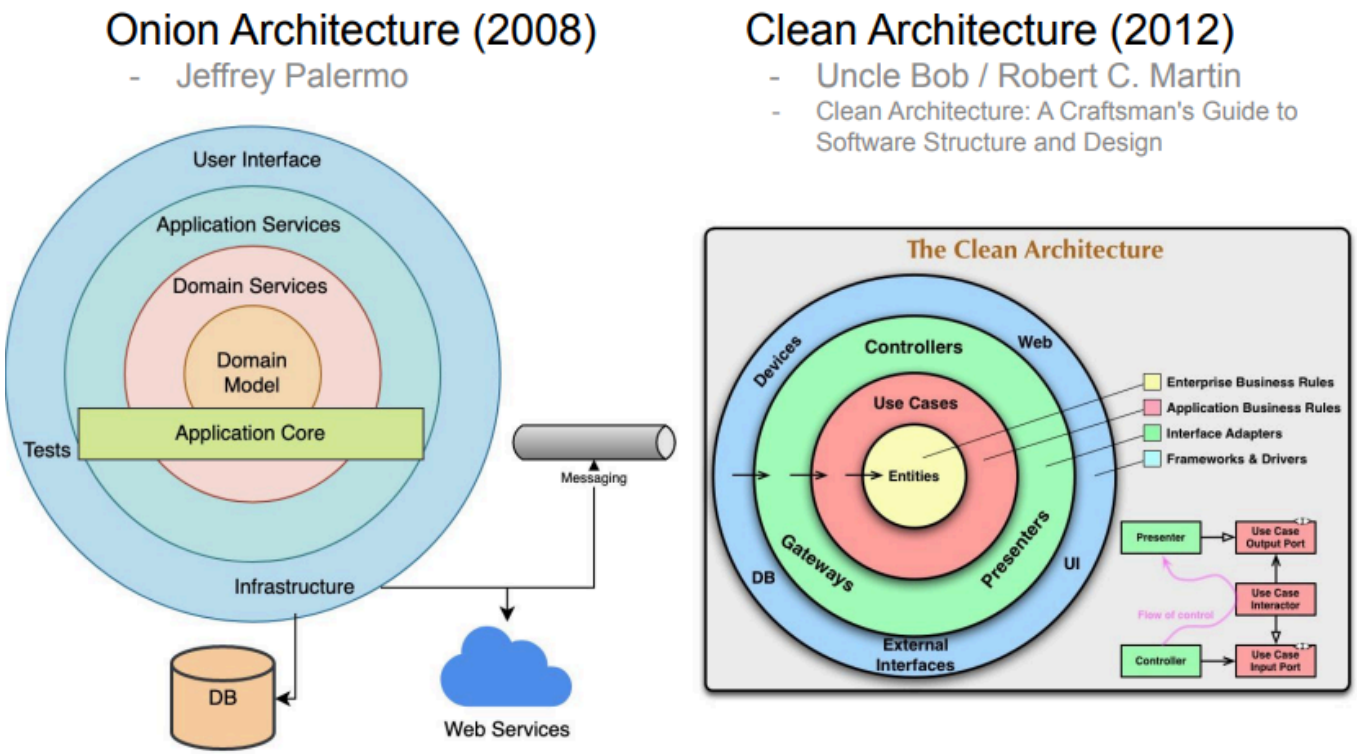
- Hay dos partes del sistema: la reglas del negocio y todo lo periférico.
- la reglas de negocio son el núcleo de la app, lo periférico pueden ser: interfaces de usuario, bases de datos, redes, etc.
- se busca que el núcleo no depende de los detaller técnicos de lo periférico
- se diagrama como un hexagoo interno uno externo, el externo se divide en triangulos, cada triangulo es el puerto donde se puede conectar una

implementación de la componente necesitada.





Timeline after hexagonal architecture



donde tenemos:

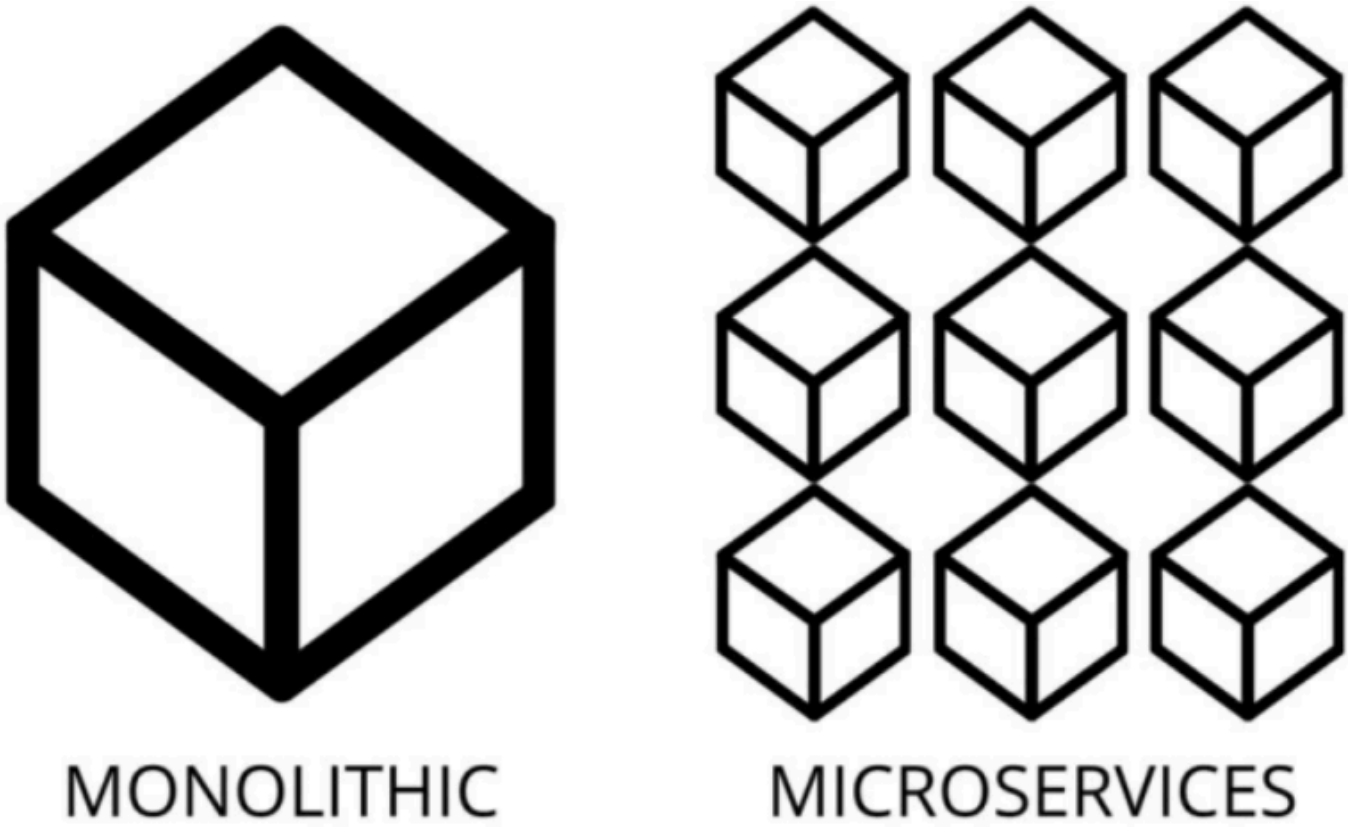
- onion architecture: la capa más interna es el *modelo de dominio* (reglas de negocio, las entidades, etc), la siguiente son los *servicios del dominio* que proveen los servicios que coordina las entidades del negocio, la siguiente son los *sevicios de applicación* que se relaciona las respuesta http, etc; la siguientes se *interfaz de usuario*, donde se conectan (A travpes de mensajes por internet) los servicios web, las bases de dato, etc. Los tests abarcan todo excepto user interface.
- Clean Architecture:
  - *frameworks & drivers*: la capa más externa, son las interfaces externas con las que interactua nuestro sistema, estas pueden estar en constante cambio. Son los servicios externos que usamos, pueden ser:
    - UI
    - DB
    - Devices
    - ObjectRelationalMapping (como hibernate)
    - servicio de envio de mails

- *interface adapters*: adaptan los datos para que se puedan mover entre capas. Aqui está la logica para transformar entidade en datos que la UI o la persistencia puede entener, asi tambien como los DTOs. inlcuye:
    - controllers: que reciben input
    - presenters: formatean la salida
    - gateways: adaptadores hacia sistemas externos como bases de datos o APIS.
- *uses cases /Reglas de negocio de aplicación*: lógica específica de la aplicación, orquesta pasos necesarios para cumplir un requerimiento funcional. coordina entidades, validaciones, flujos,etc. SOLO DEPENDE DE ENTIDADES
- *entidades/reglas de negocio de la empres*: las reglas más fundamentales del dominio, contiene la invariantes de cada entidad (como useer o product por ejemplo). NO DEPENDEN DE NADA.

microservicios

Arquitectura que descompone una aplicación monolítica en un conjunto de pequeños serviciosautónomos y desarrollados de manera independiente.
Cada servicio tiene una responsabilidad específica y se comunica con otros servicios a través de interfaces bien definidas (suele usar protocolos ligeros como

HTTP o mensajería asincrónica).



micro frontends

cada microfrontedn representa una funcionalidad o sección autónoma del frontend