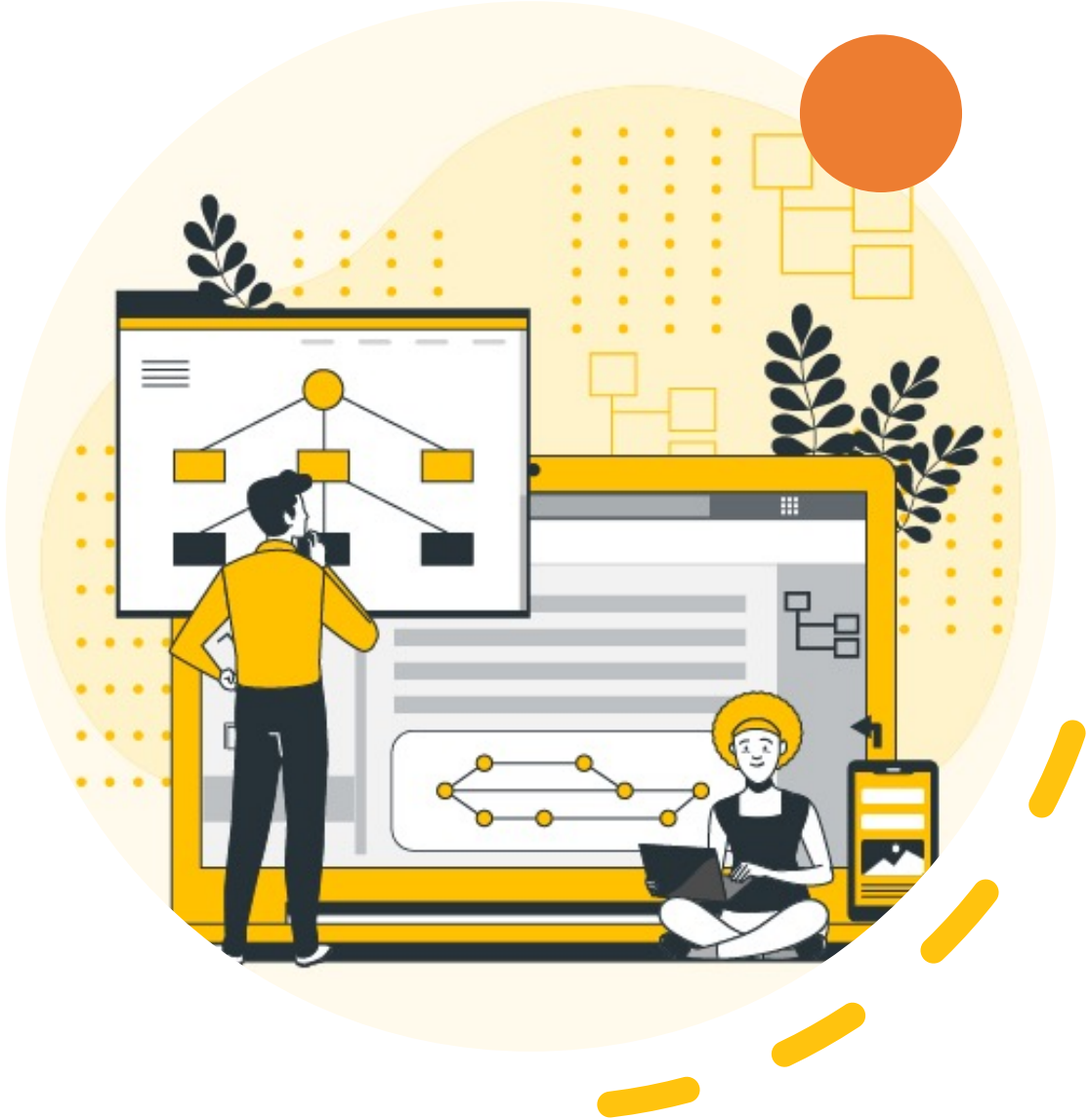
Introducción a las arquitecturas limpias

Karla A. Fonseca Márquez
karlafm@unam.mx

Septiembre 2021

La **arquitectura de software** de un sistema es el conjunto de estructuras necesarias para razonar acerca del sistema

- Elementos de software
- Relaciones entre ellos
- Propiedades de ambos.



UNO DE LOS PROPÓSITOS DE LA ARQUITECTURA DE SOFTWARE ES SOPORTAR EL CICLO DE VIDA DEL SISTEMA

ARQUITECTURA DE SOFTWARE

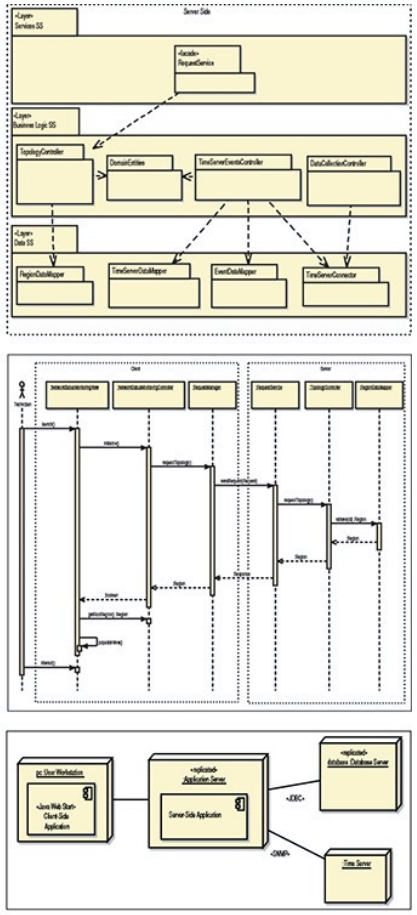
Casos de uso

Atributos de calidad

Restricciones

Decisiones de diseño

Arquitecto




KEEP
CALM
BECAUSE
CHANGES
HAPPEN

MANTENIMIENTO DE SOFTWARE

Mantenimiento correctivo

Modificaciones o actualizaciones que corrigen un defecto

Mantenimiento adaptativo

Mantener el software vigente en relación con tecnologías y negocio cambiantes

Mantenimiento perfectivo

Modificaciones a características existentes que incrementarán la funcionalidad y eficiencia

Mantenimiento preventivo

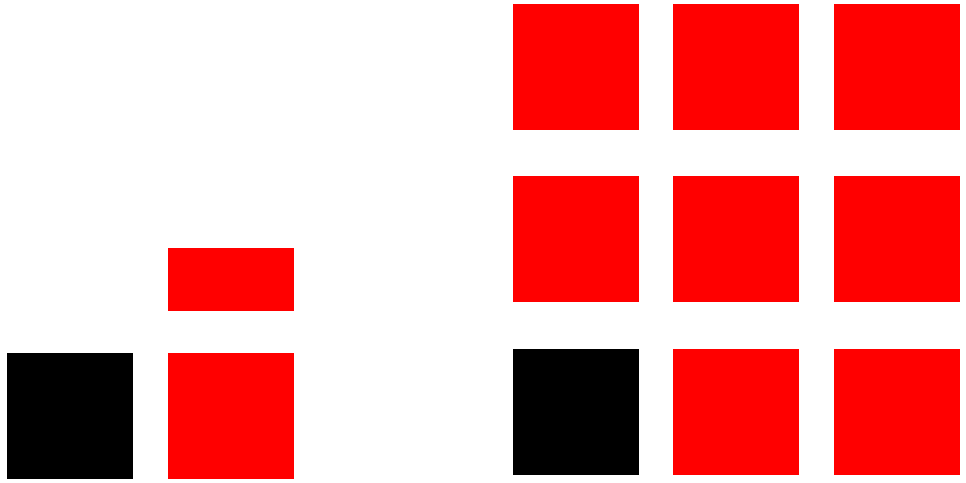
Cambios realizados al software para prevenir futuros problemas

The Cost of Poor Software Quality in the US: A 2018 Report, Consortium for IT Software Quality (CISQ)
https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/The-Cost-of-Poor-Quality-Software-in-the-US-2018-Report.pdf

ESTIMACIÓN DEL COSTO DE MANTENIMIENTO

Costo hasta su liberación

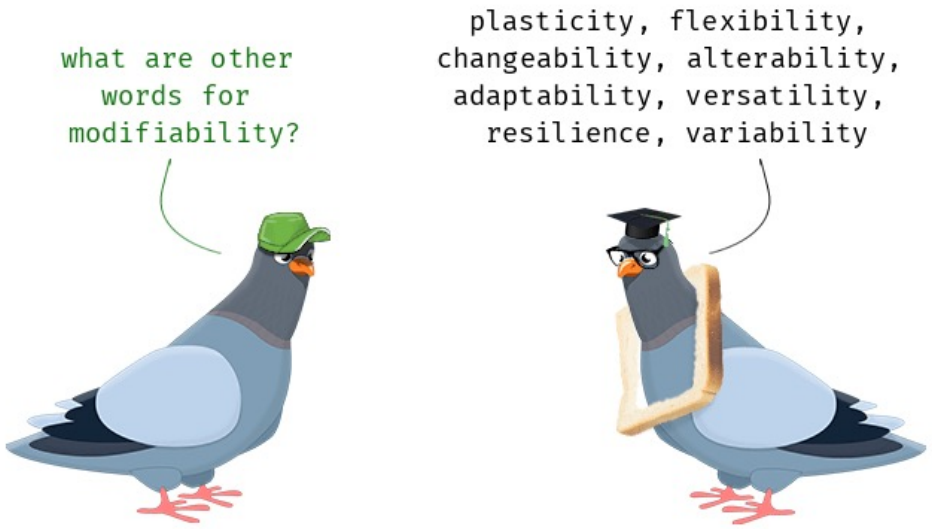
Costo del mantenimiento



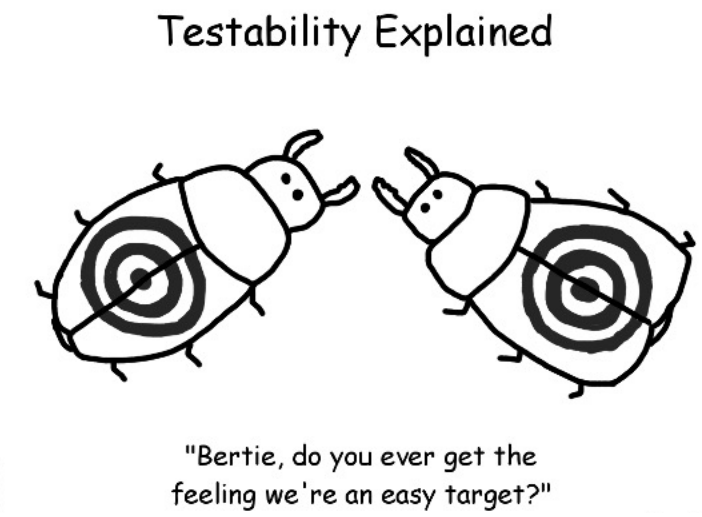
	1	1.5x	1	9x
% Costo total del software	**40%**	**60%**	**10%**	**90%**

ATRIBUTOS DE CALIDAD QUE AFECTAN EL COSTO DE MANTENIMIENTO

Modificabilidad



Capacidad de ser probado



UNA MALA ARQUITECTURA GENERA SISTEMAS:

Complejos
Incoherentes
Rígidos
Frágiles
Difíciles de probar
No mantenibles

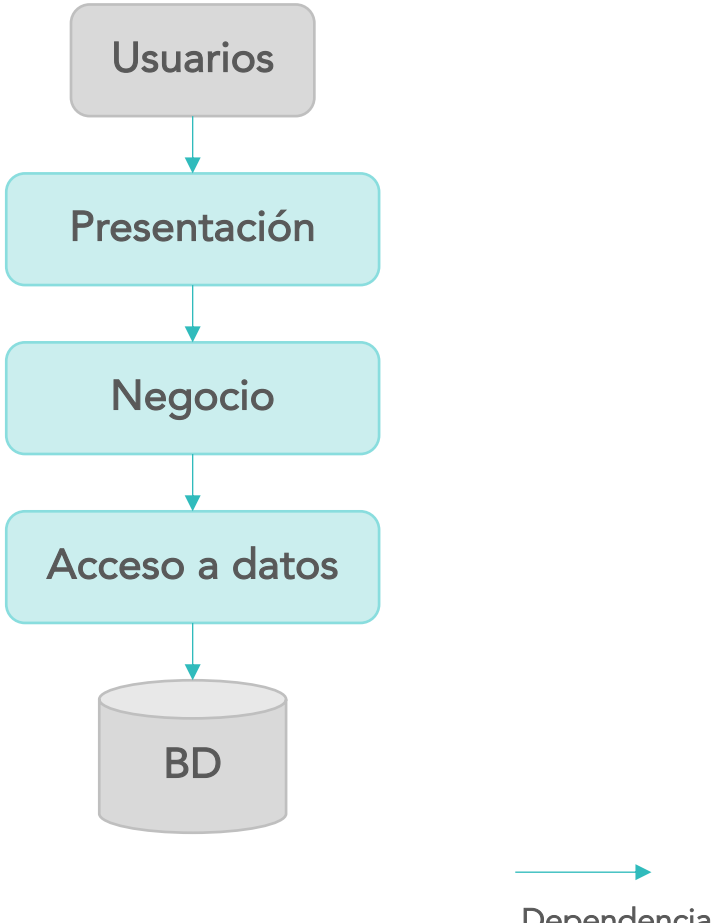
¿CÓMO LIMITAR LA COMPLEJIDAD?

- Separación de intereses
- Reducir el acoplamiento
- Incrementar la cohesión
- Reducir el tamaño de módulos
- Inyección de dependencias

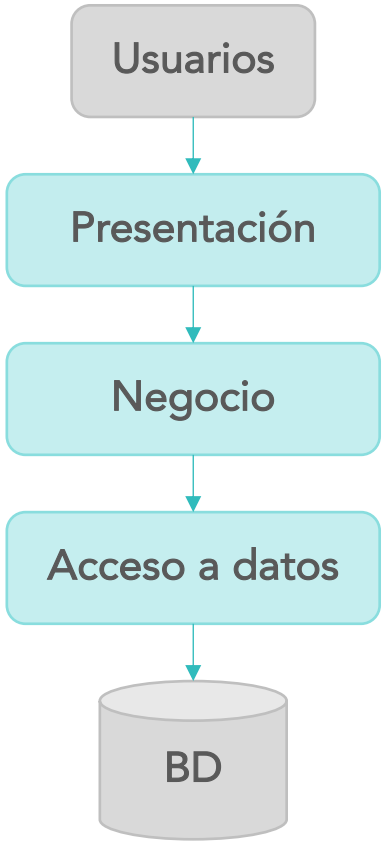
- Patrones arquitectónicos
- Patrones de diseño

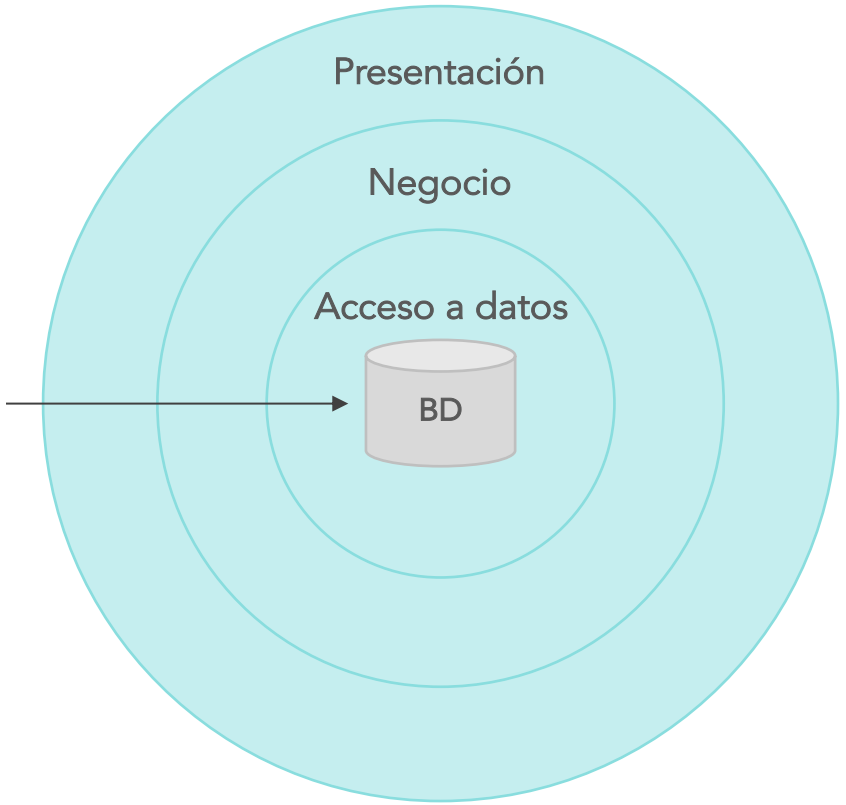
ARQUITECTURA EN CAPAS

- Los componentes se organizan en capas horizontales.
- Cada capa tiene un rol y responsabilidad específicos dentro de la aplicación.
- Uno de los grandes beneficios de este patrón arquitectónico es la separación de intereses



DISEÑO CENTRADO EN LA BASE DE DATOS





Dependencia

¿QUÉ ES LO MÁS IMPORTANTE DEL SOFTWARE?


El problema o negocio que intentamos resolver


Los casos de uso


Las reglas de negocio

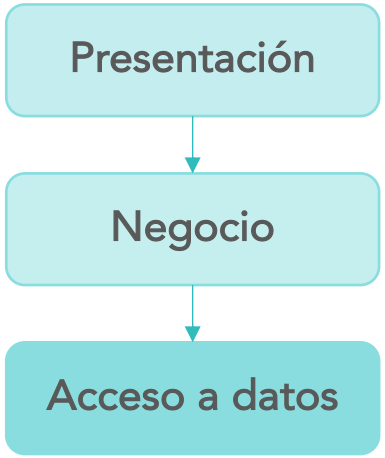

El manejador de la base de datos

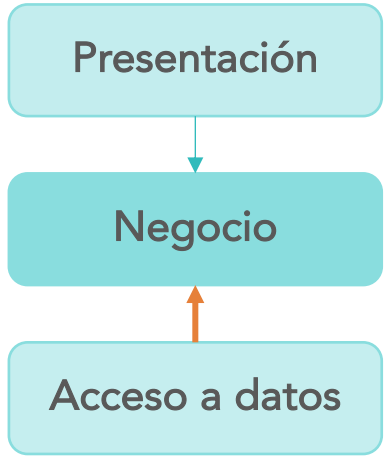

La interfaz de usuario


El framework que se elija

(ESTOS SON DETALLES)

ARQUITECTURA CENTRADA EN EL DOMINIO





Dependencia

Arquitecturas limpias

ARQUITECTURAS LIMPIAS

Simple

Comprensible

Flexible

Son propuestas que plantean un conjunto de **prácticas** que se utilizan para generar arquitecturas de software

Emergente

Fácil de probar

Fácil de mantener

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

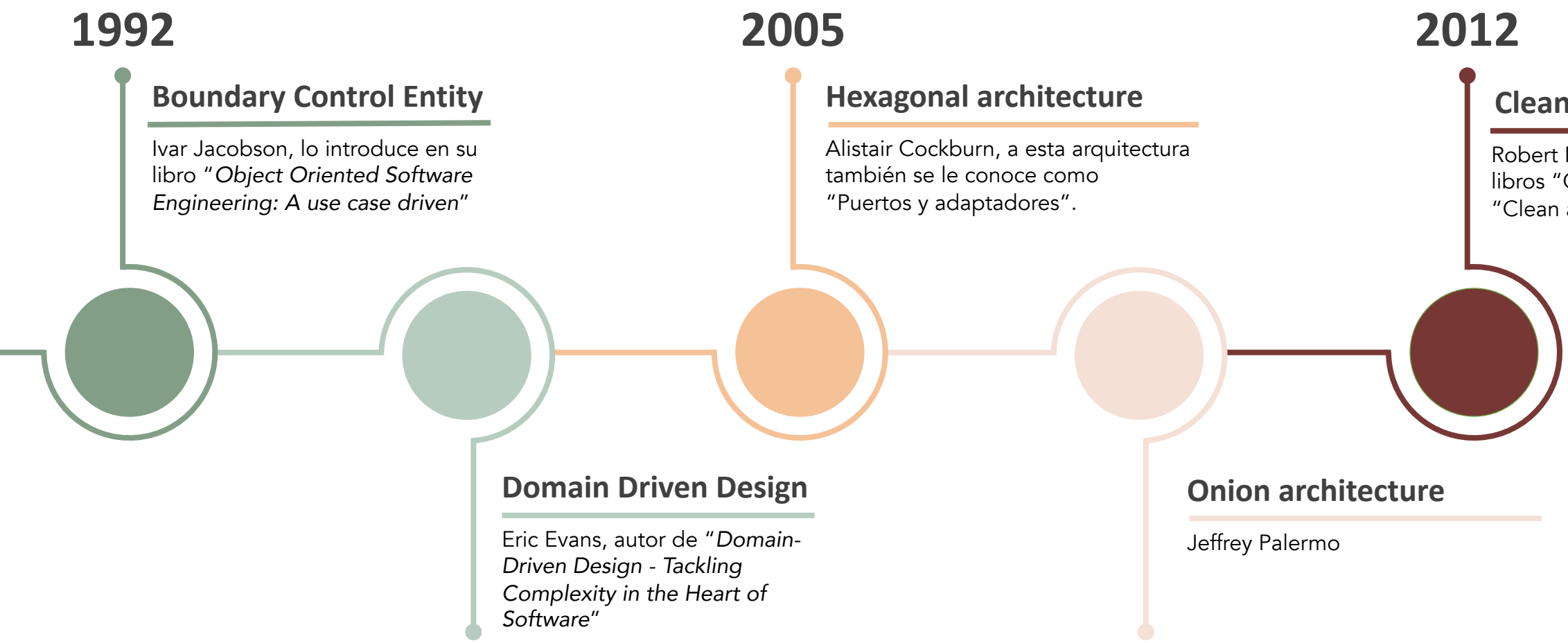
- Independiente de marcos de trabajo o frameworks.
  La arquitectura no depende de la existencia de una librería o funciones específicas. Esto permite utilizar dichos marcos como herramientas, sin necesidad de incluir el sistema en su estructura.

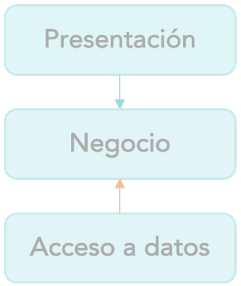
- Fácil de probar.
  Las reglas de negocio se pueden probar sin la interfaz de usuario, la base de datos, el servidor web o cualquier otro elemento externo.

- Independiente de la interfaz de usuario.
  Esta puede cambiar fácilmente, sin modificar el resto del sistema. Por ejemplo, una interfaz de usuario web podría reemplazarse por una interfaz de usuario de consola, sin cambiar las reglas de negocio.

- Independiente de la base de datos.
  Puede cambiar Oracle o SQL Server por Mongo, BigTable, CouchDB u otra. Las reglas de negocio no se encuentran vinculadas a la base de datos.

- Independiente de cualquier librería externa.
  Al igual que con los marcos de trabajo, es posible utilizar librerías externas, las cuales pueden intercambiarse por otras sin afectar la funcionalidad del sistema.


1992
Boundary Control Entity
Ivar Jacobson, lo introduce en su libro "*Object Oriented Software Engineering: A use case driven*"
2004
Domain Driven Design
Eric Evans, autor de "*Domain-Driven Design - Tackling Complexity in the Heart of Software*"
2005
Hexagonal architecture
Alistair Cockburn, a esta arquitectura también se le conoce como "Puertos y adaptadores".
2008
Onion architecture
Jeffrey Palermo
2012
Clean arquitecture
Robert Martin, autor de los libros "Clean code" y "Clean arquitecture"

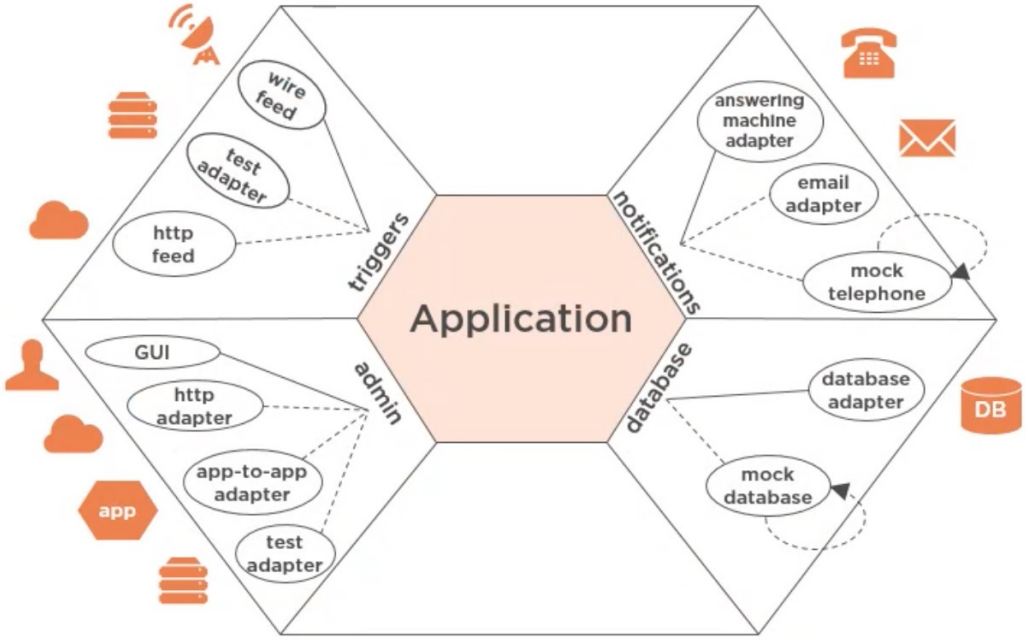


ARQUITECTURA HEXAGONAL O “PUERTOS Y ADAPTADORES”

“Crea tu aplicación para trabajar sin una interfaz de usuario o una base de datos para que puedas ejecutar pruebas de regresión automatizadas, trabajar cuando la base de datos no esté disponible.”

“Los problemas tanto del lado del usuario como del lado del servidor son causados por el mismo error de diseño y programación - la mezcla entre la lógica de negocio y la interacción con entidades externas”

La regla a seguir es que el código del interior no debe filtrarse a la parte exterior.



https://alistair.cockburn.us/hexagonal-architecture/

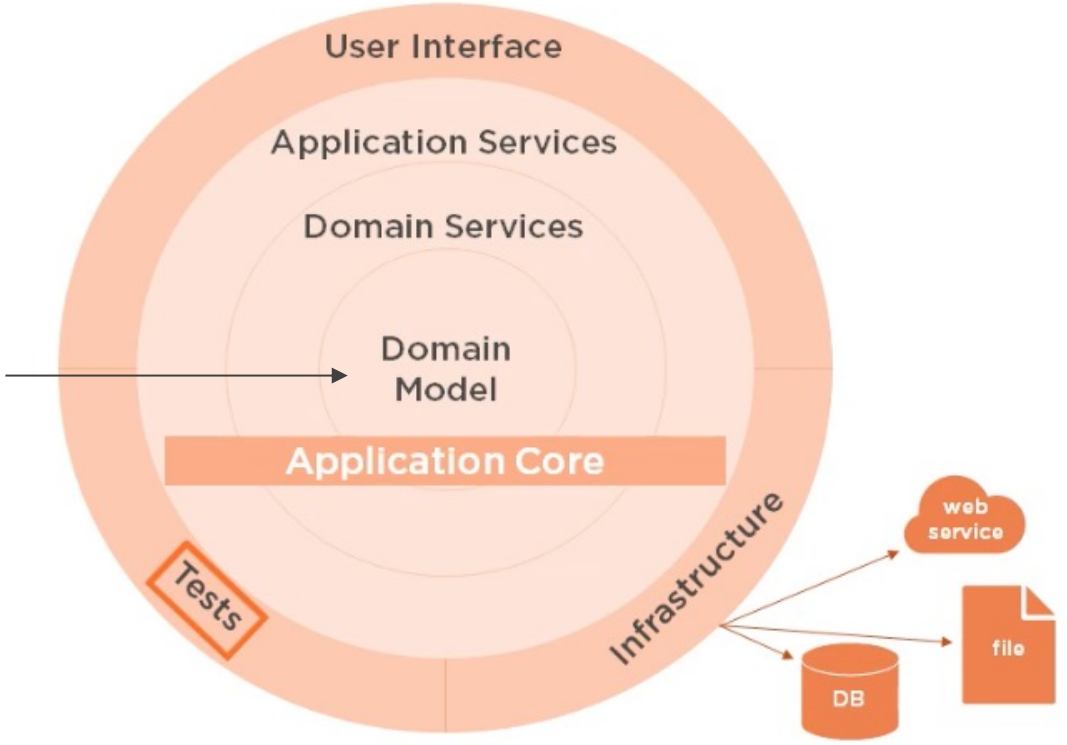
ONION ARCHITECTURE

La aplicación se construye alrededor de un modelo de objetos independiente.

Las capas internas *definen* interfaces.
Las capas externas *implementan* interfaces.

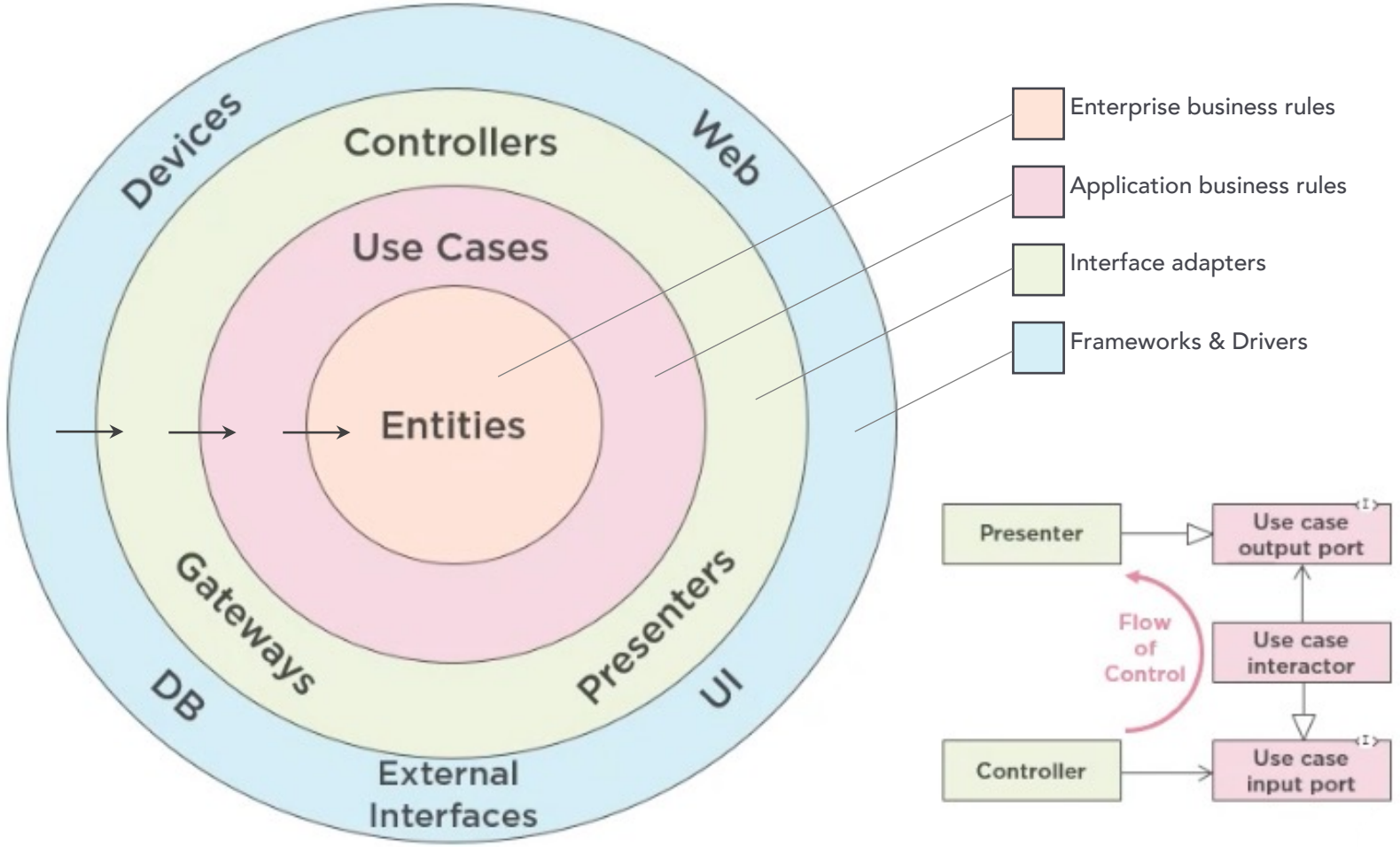
La dirección de acoplamiento es hacia el centro.

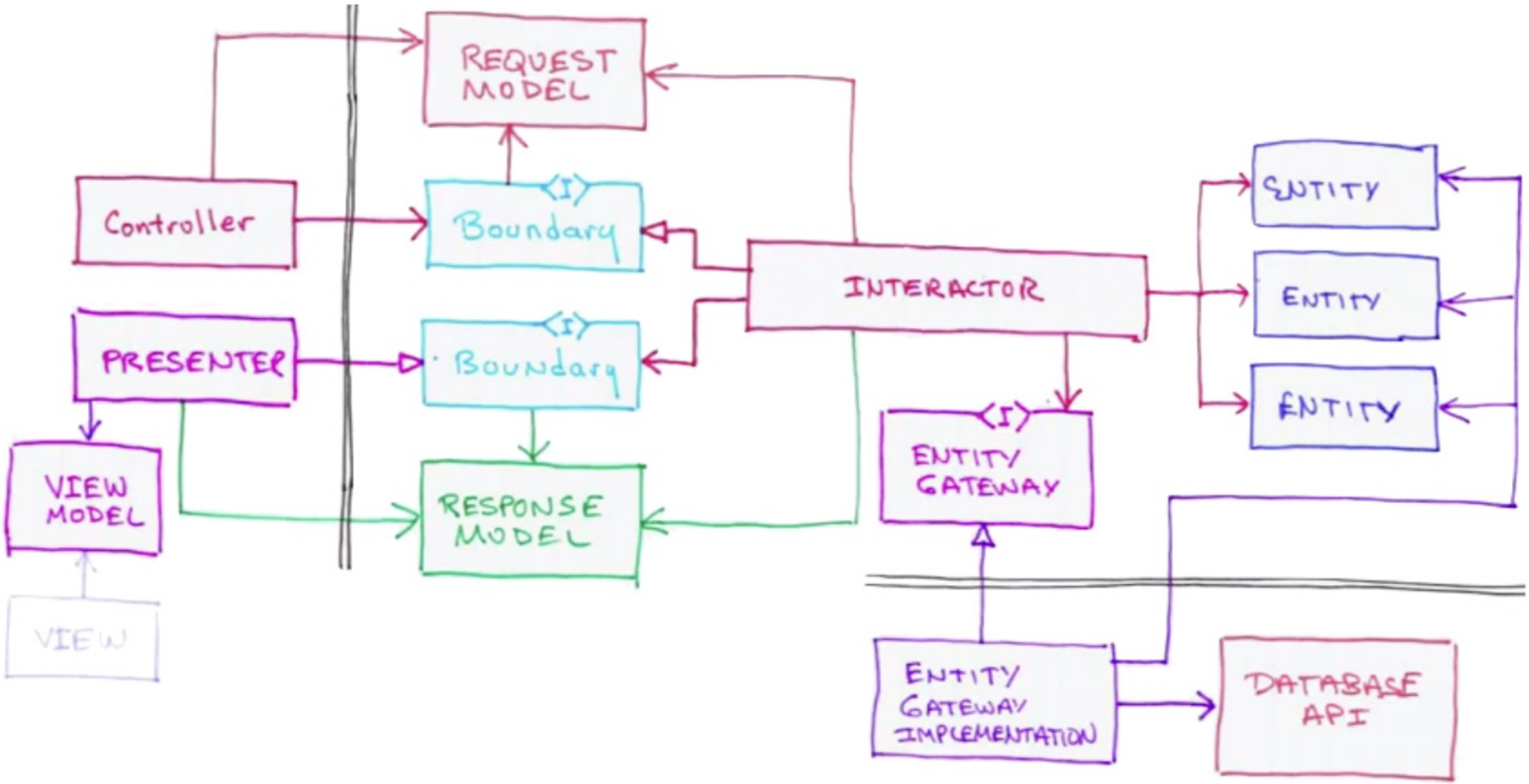
Todo el código central de la aplicación puede ser compilado y ejecutado separado de la infraestructura.

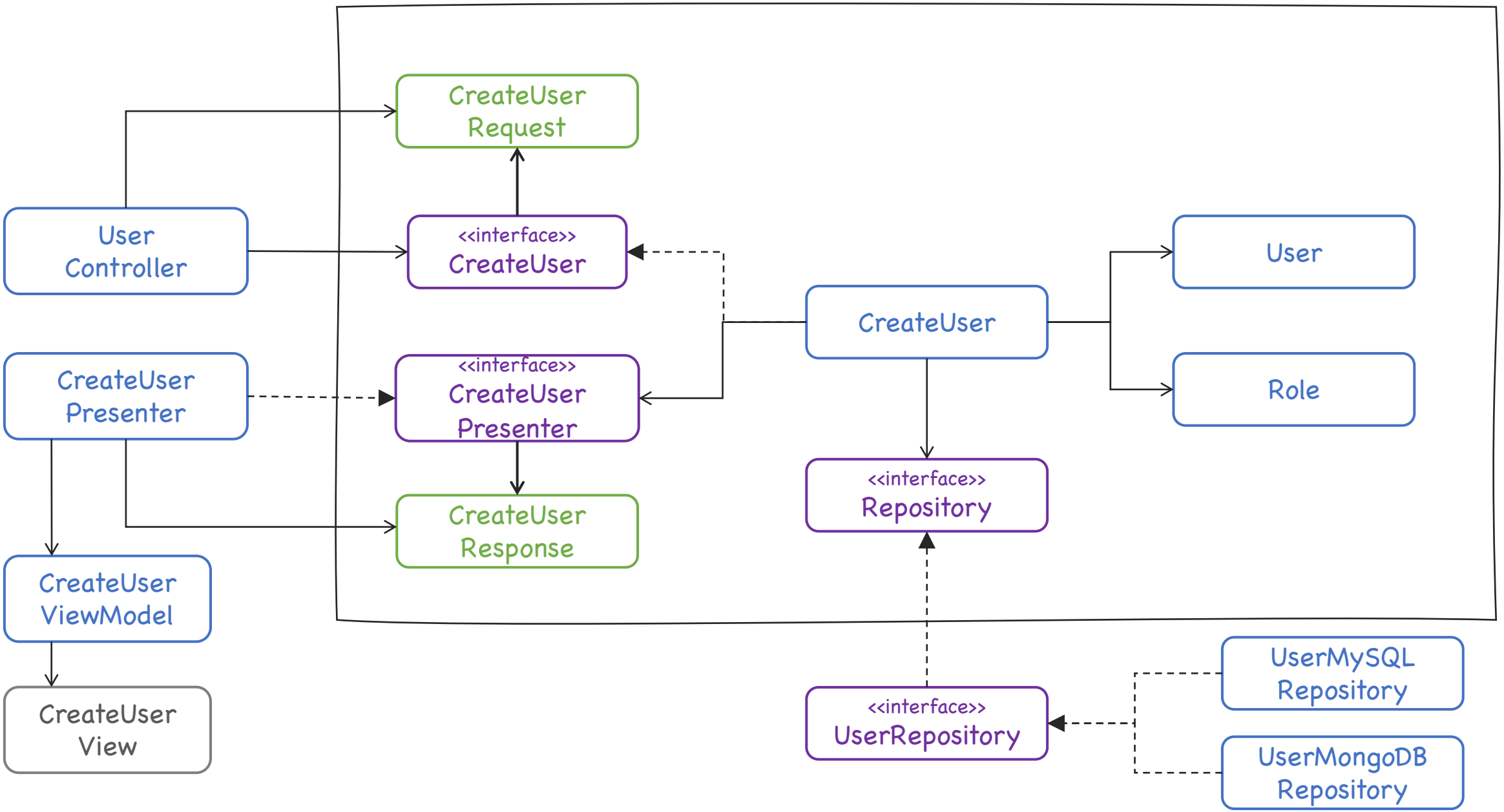


https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/

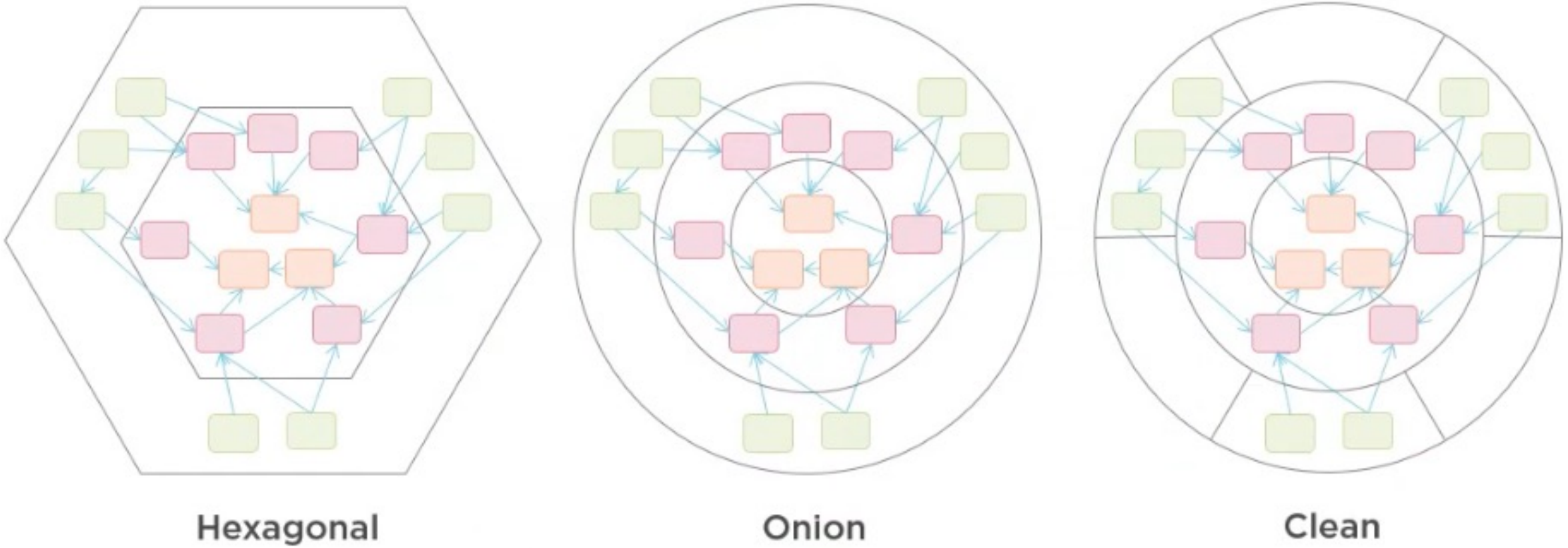
CLEAN ARCHITECTURE




REQUEST MODEL
Controller
<I>
Boundary
INTERACTOR
ENTITY
ENTITY
ENTITY
PRESENTER
<I>
Boundary
<I>
ENTITY GATEWAY
VIEW MODEL
RESPONSE MODEL
VIEW
ENTITY GATEWAY IMPLEMENTATION
DATABASE API


CreateUser
Request
User
Controller
<<interface>>
CreateUser
User
CreateUser
CreateUser
Presenter
<<interface>>
CreateUser
Presenter
Role
<<interface>>
Repository
CreateUser
Response
CreateUser
ViewModel
CreateUser
View
<<interface>>
UserRepository
UserMySQL
Repository
UserMongoDB
Repository

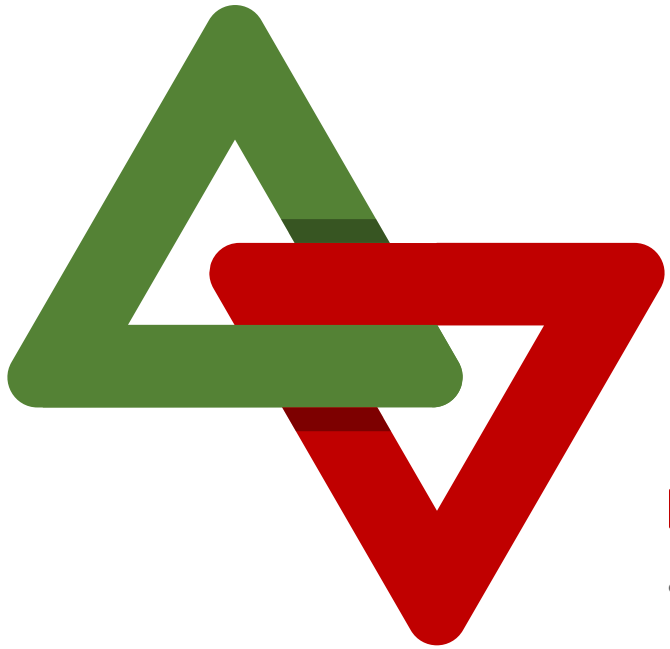
ARQUITECTURAS CENTRADAS EN EL DOMINIO



https://blog.ploeh.dk/2013/12/03/layers-onions-ports-adapters-its-all-the-same/

VENTAJAS

- Enfoque en el dominio.
- Menos acoplamiento, dirigido fuera del dominio.
- Compatible con un diseño guiado por el dominio
- La interfaz de usuario, librerías externas y persistencia de datos se pueden intercambiar por otra implementación.
- Promueve la generación de código que es fácil de probar.



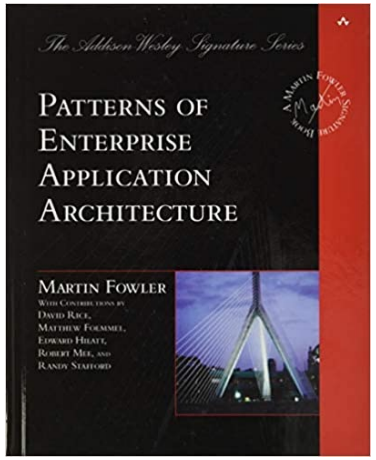
DESVENTAJAS

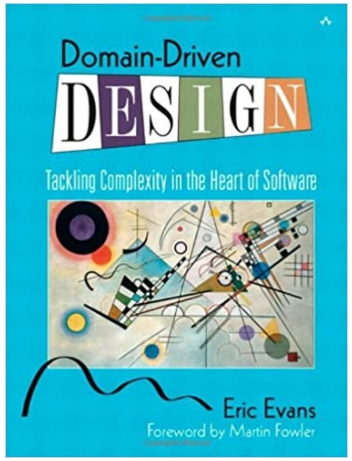
- Excesivo para aplicaciones pequeñas.
- Muchas interfaces
- Costo de desarrollo más elevado
- Requiere un diseño más intencional
- Pueden perderse ciertas optimizaciones

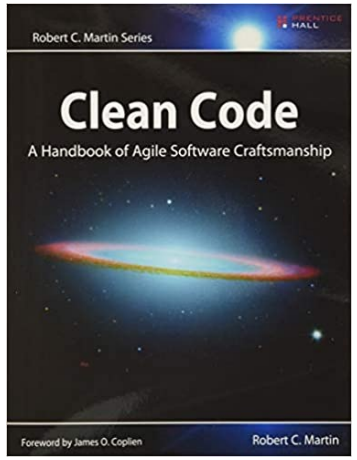
CONCLUSIONES

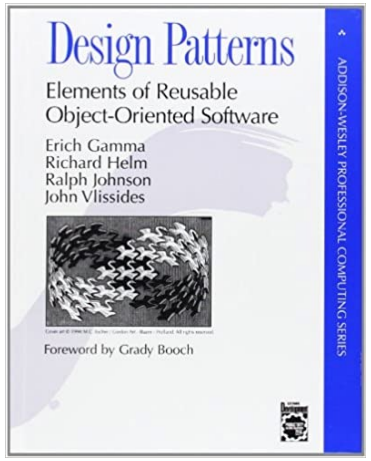
- El énfasis al diseñar la arquitectura debe dirigirse a lograr una implementación exitosa y mantenible de los requerimientos de negocio y de operación.
- La arquitectura de software no son las herramientas y frameworks.
- El arquitecto debe maximizar la cantidad de decisiones no tomadas
- El reto es lograr que el sistema sea fácil de cambiar, en todas las maneras en que deba cambiar, dejando las opciones abiertas.

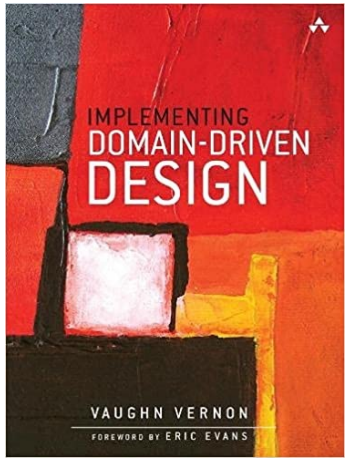
REFERENCIAS (LIBROS)

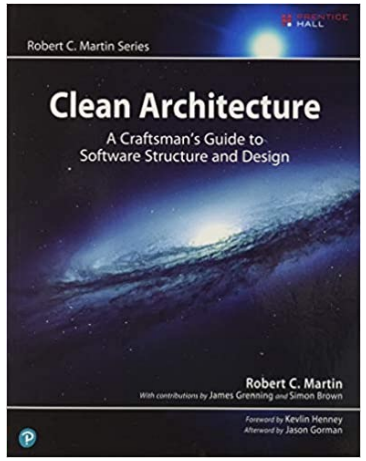












REFERENCIAS (PÁGINAS WEB)

- Hexagonal architechture, Alistair Cockburn
  https://alistair.cockburn.us/hexagonal-architecture/
- The onion architecture, Jeffrey Palermo
  https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/
- Clean architecture, Robert Martin
  https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
- “Dependency Inversion Principle”, Robert Martin
  http://www.objectmentor.com/resources/articles/dip.pdf
- “Dependency Injection” pattern por Martin Fowler
  http://www.martinfowler.com/articles/injection.html
- “The Software Architecture Chronicles”
  https://herbertograca.com/2017/07/03/the-software-architecture-chronicles/

REFERENCIAS (VIDEOS)

- Eric Evans (2016). **Tackling Complexity in the Heart of Software**.
  https://www.youtube.com/watch?v=dnUFEg68ESM
- Robert Martin (2014). **Clean architecture**
  https://www.youtube.com/watch?v=Nltqi7ODZTM
- Alistair Cockburn. "**Hexagone**", 3 partes:
  https://www.youtube.com/watch?v=th4AgBcrEHA
  https://www.youtube.com/watch?v=iALcE8BPs94
  https://www.youtube.com/watch?v=DAe0Bmcyt-4

Karla A. Fonseca Márquez
karlafm@unam.mx