

Domain-Driven Design y Arquitectura Hexagonal

Pepe Fabra Valverde

Senior Frontend Developer



Agenda

- **DDD**

- Qué es DDD (pero de verdad)
- Lenguaje de negocio > Lenguaje de negocio adaptado al técnico
- Bounded Context

Agenda

- **DDD**
 - Qué es DDD (pero de verdad)
 - Lenguaje de negocio > Lenguaje de negocio adaptado al técnico
 - Bounded Context
- **Arquitectura Hexagonal**
 - Principio de las Arquitecturas Limpias
 - Estructura de capas
 - Dominio
 - Aplicación
 - Infraestructura
 - Estructura de directorios
 - Testing



Domain-Driven

DESIGN

Tackling Complexity in the Heart of Software



Eric Evans
Foreword by Martin Fowler

Lanzamiento

Escrito por Eric Evans, publicado en 2003

Contexto

- Waterfall y sus consecuencias (entregas a destiempo)
- Agile empezando a integrarse
- Lenguaje de negocio adaptado al técnico

Dominio de Negocio

El negocio puede ser algo muy grande, qué vas a cubrir es el dominio

Es el nivel de zoom que harás en el negocio, tanto por compañía/proyecto como por equipo

Eric Evans explicando Domain-Driven Design

<https://www.youtube.com/watch?v=pMuiVlnGqjk>

Domain-Driven Design Europe

Canal de referencia

Qué propone Domain-Driven Design

Qué propone Domain-Driven Design

Usar el lenguaje de negocio como el lenguaje ubicuo del sistema

¿Cómo se consigue?

¿Cómo se consigue?

- Diccionario interno, el equipo usa un mismo lenguaje con unas mismas palabras

¿Cómo se consigue?

- Diccionario interno, el equipo usa un mismo lenguaje con unas mismas palabras
- Cada equipo puede tener su propia definición de un concepto (la definición del dominio de negocio que estén adaptando)

Negocio como core

- “Es que eso es muy difícil”, “mejor atajo por aquí”, “eso no tiene sentido, mejor hacemos esto otro”

Negocio como core

- “Es que eso es muy difícil”, “mejor atajo por aquí”, “eso no tiene sentido, mejor hacemos esto otro”
- Leer el código debería dar un entendimiento del dominio de negocio que se está implementando

Negocio como core

- “Es que eso es muy difícil”, “mejor atajo por aquí”, “eso no tiene sentido, mejor hacemos esto otro”
- Leer el código debería dar un entendimiento del dominio de negocio que se está implementando
- Somos traductores, de negocio a código, no de código a negocio

Bounded Context

Encapsulación de una parte del (dominio de) negocio

Bounded Context a código

Módulos, un módulo es el bounded context

Low coupling, High cohesion

Low coupling, High cohesion

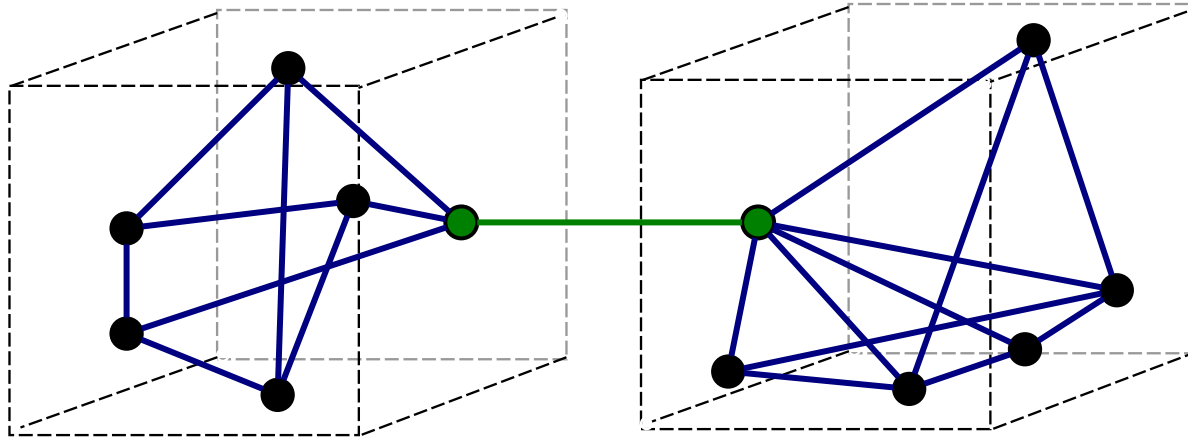
- Si se respeta este principio, entonces estás definiendo bien un bounded context

Low coupling, High cohesion

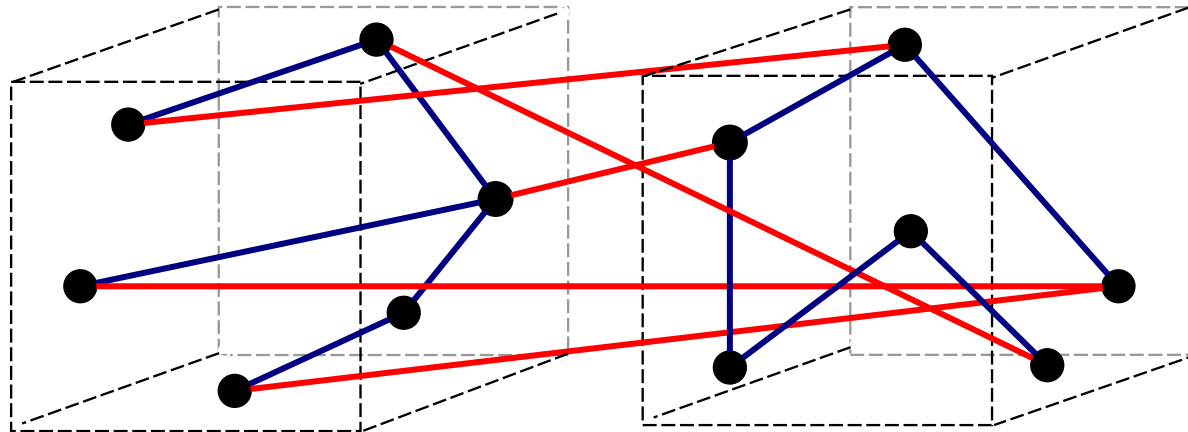
- Si se respeta este principio, entonces estás definiendo bien un bounded context
- Este principio es un principio para desarrollar módulos de software

Low coupling, High cohesion

- Si se respeta este principio, entonces estás definiendo bien un bounded context
- Este principio es un principio para desarrollar módulos de software
- Sirve de introducción a **Ports & Adapters**



a) Good (loose coupling, high cohesion)

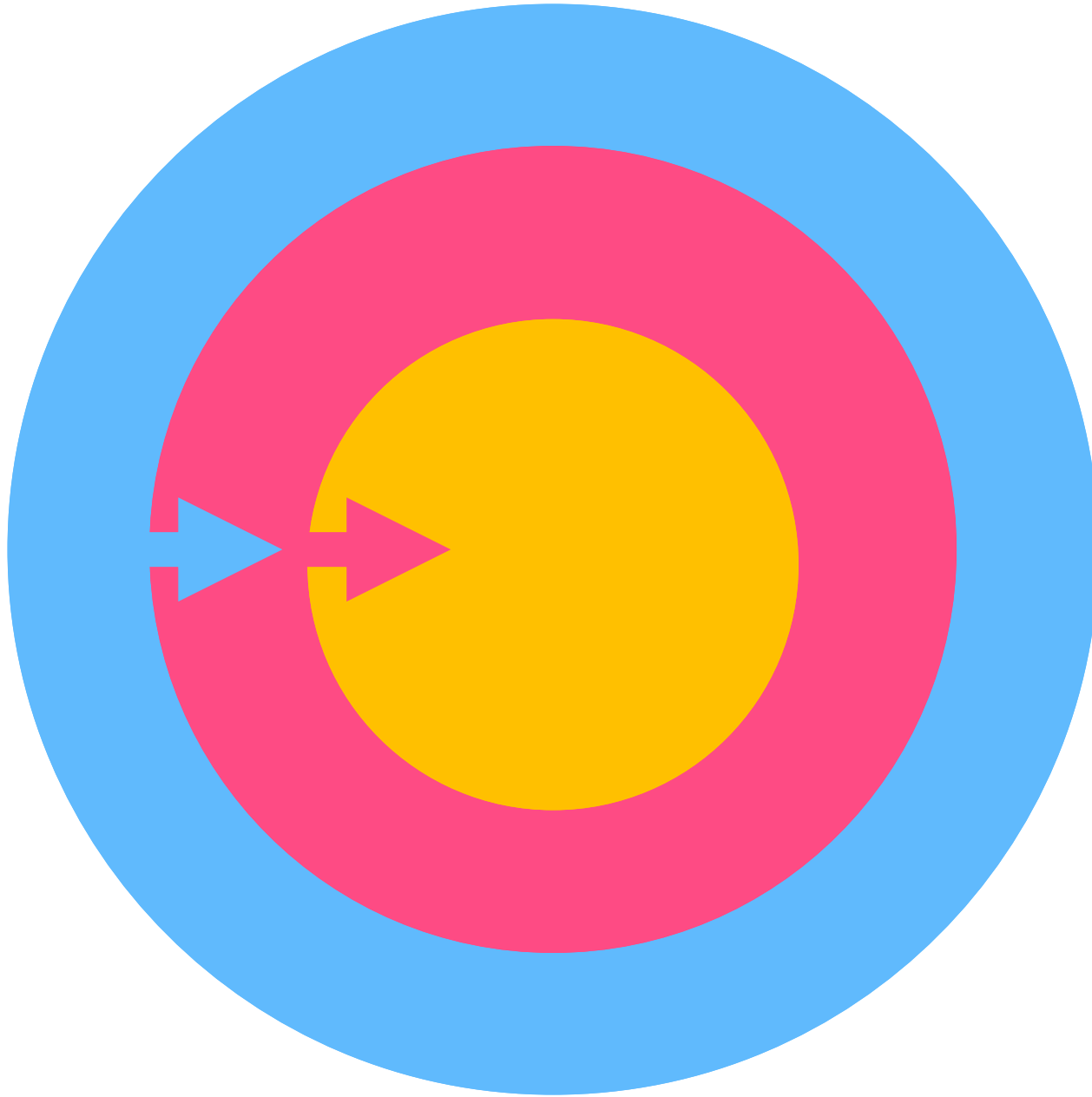


b) Bad (high coupling, low cohesion)

Arquitectura Hexagonal

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

De los peores cambios de nombre de la historia



DOMINIO

APLICACIÓN

INFRAESTRUCTURA

SOLID

SRP - Single Responsibility Principle

Los elementos del sistema tienen una única responsabilidad

- Implementan/definen una lógica
- Definen la orquestación de las distintas lógicas necesarias para cubrir otra lógica

La lógica puede separarse en lógica más pequeña

Divide y vencerás

DIP - Dependency Inversion Principle

- **Se quiere evitar que *Function* internamente se *acople* a dependencia**
- **Proponiendo que *Function* reciba por parámetro la dependencia**

Ahora sí...

Arquitectura Hexagonal

Conceptos

- **Dominio**
 - Repository
- **Aplicación (o caso de uso)**
- **Infraestructura**
 - Implementación de Repository
 - Controller

Principio de Arquitectura Limpia

Separar (desacoplar) el dominio de la infraestructura

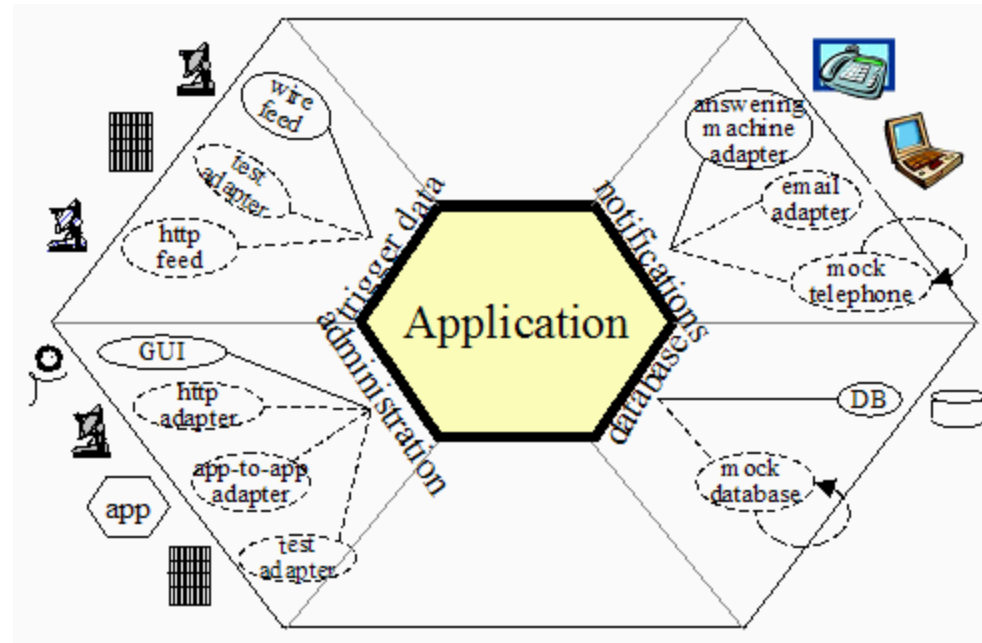
Ports & Adapters

Nombre original del patrón arquitectónico

Petición llega a infra, se aplica el caso de uso (aplicación) con la lógica de negocio (dominio), y se persiste (adapter) en BDD

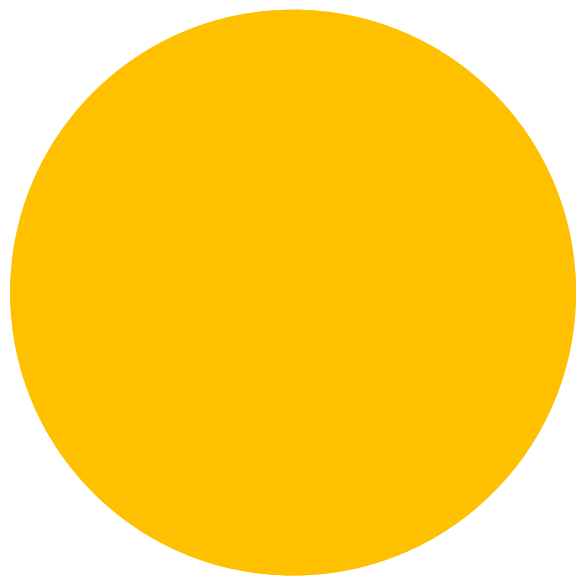
Ports & Adapters

Hexagonal es para representar la posibilidad de múltiples puertos y adaptadores, no para delimitar el número de entradas y salidas

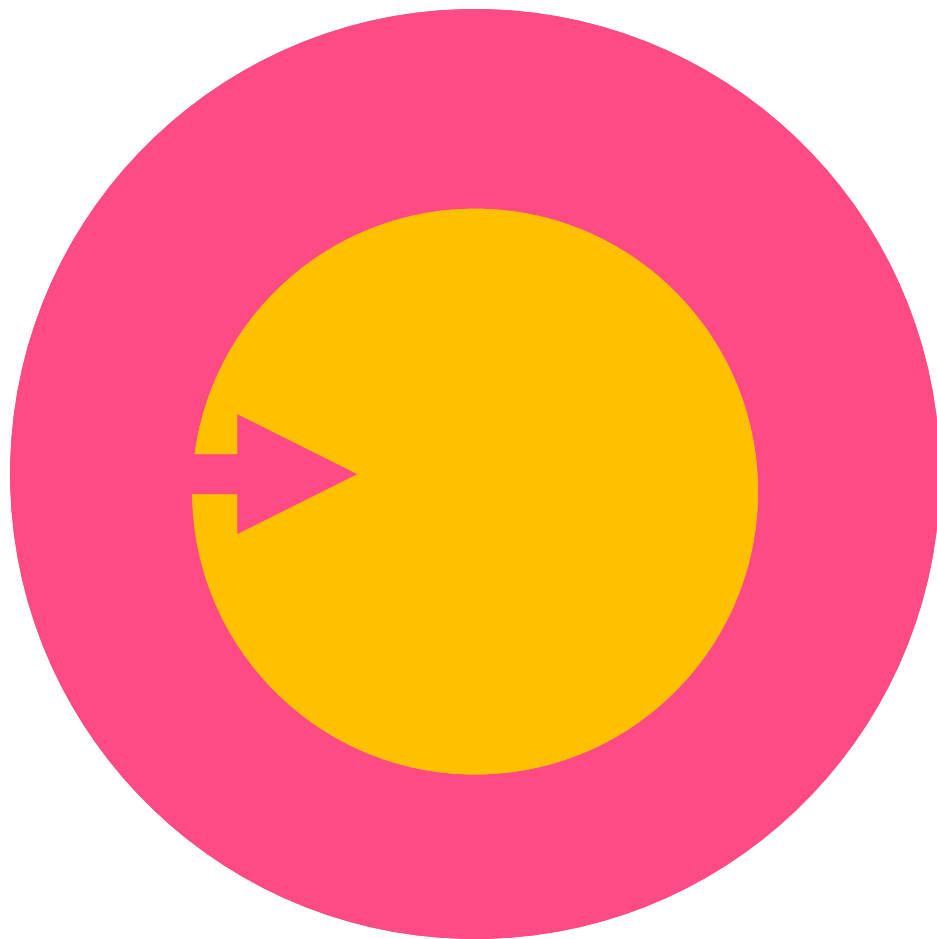


Regla de dependencias

Infraestructura -> Aplicación -> Dominio

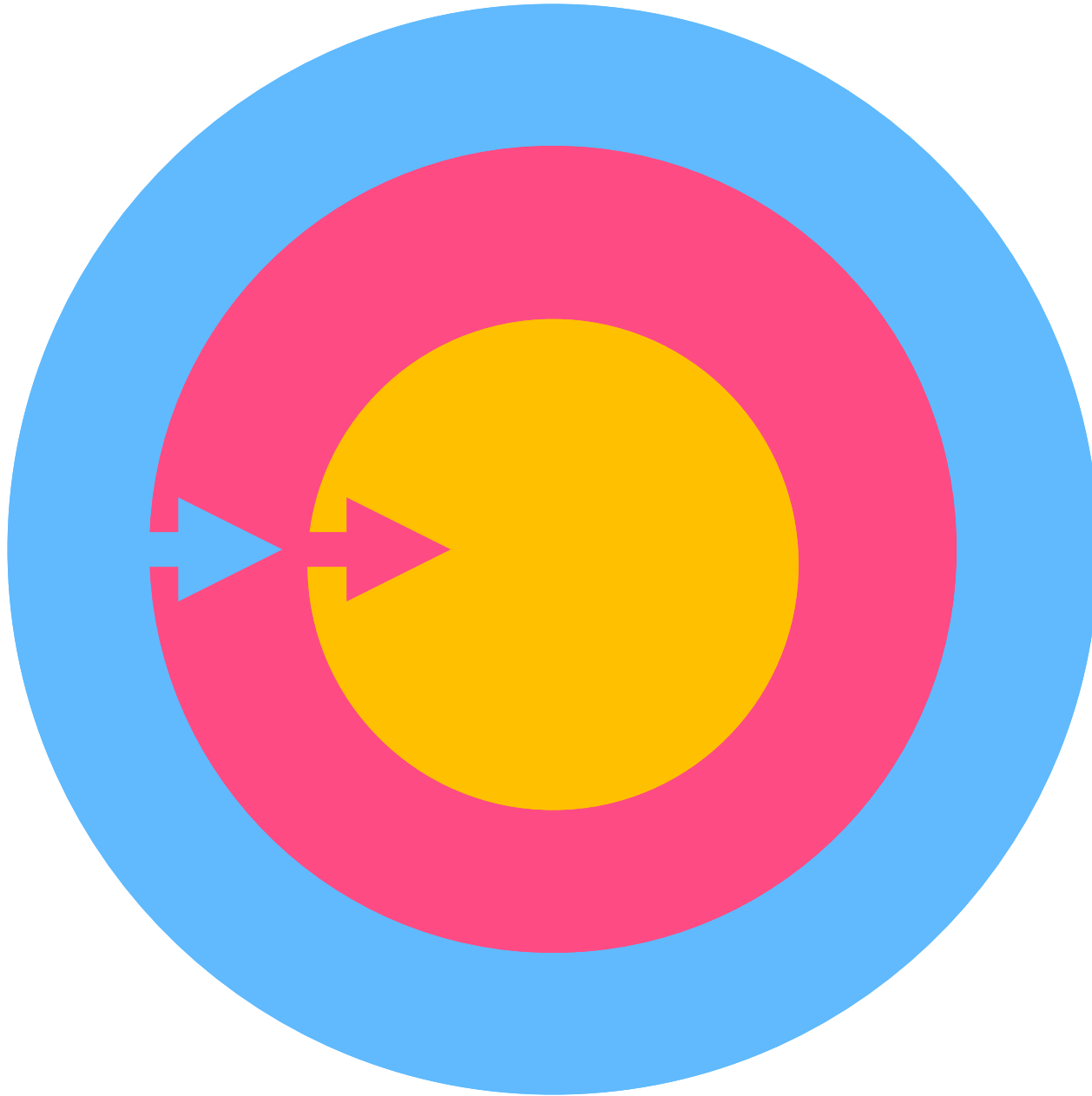


DOMINIO



DOMINIO

APLICACIÓN



DOMINIO

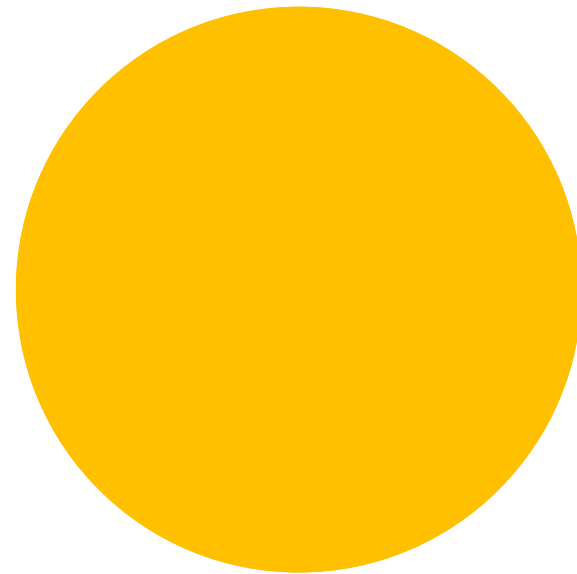
APLICACIÓN

INFRAESTRUCTURA

Dominio

Model, Repository, DomainService, ValueObjects

DomainService para casos de uso repetidos



Repository

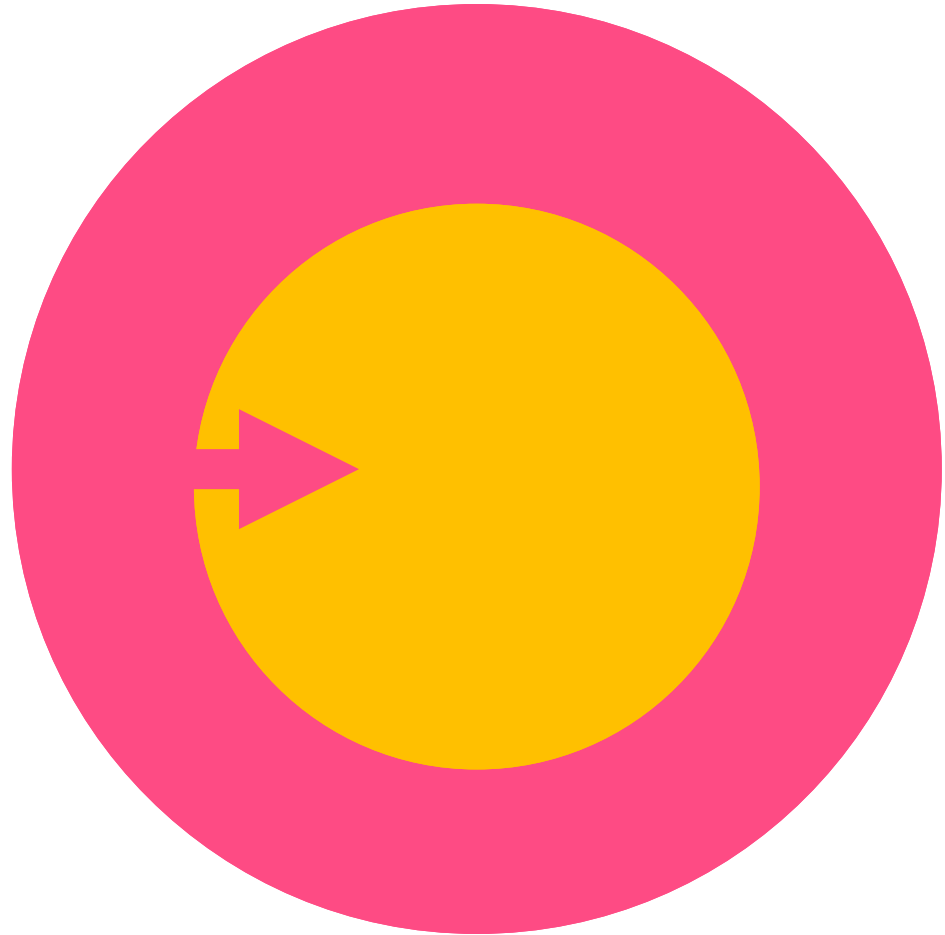
Es una interface

<https://thinkinginobjects.com/2012/08/26/dont-use-dao-use-repository/>

Aplicación o Caso de uso

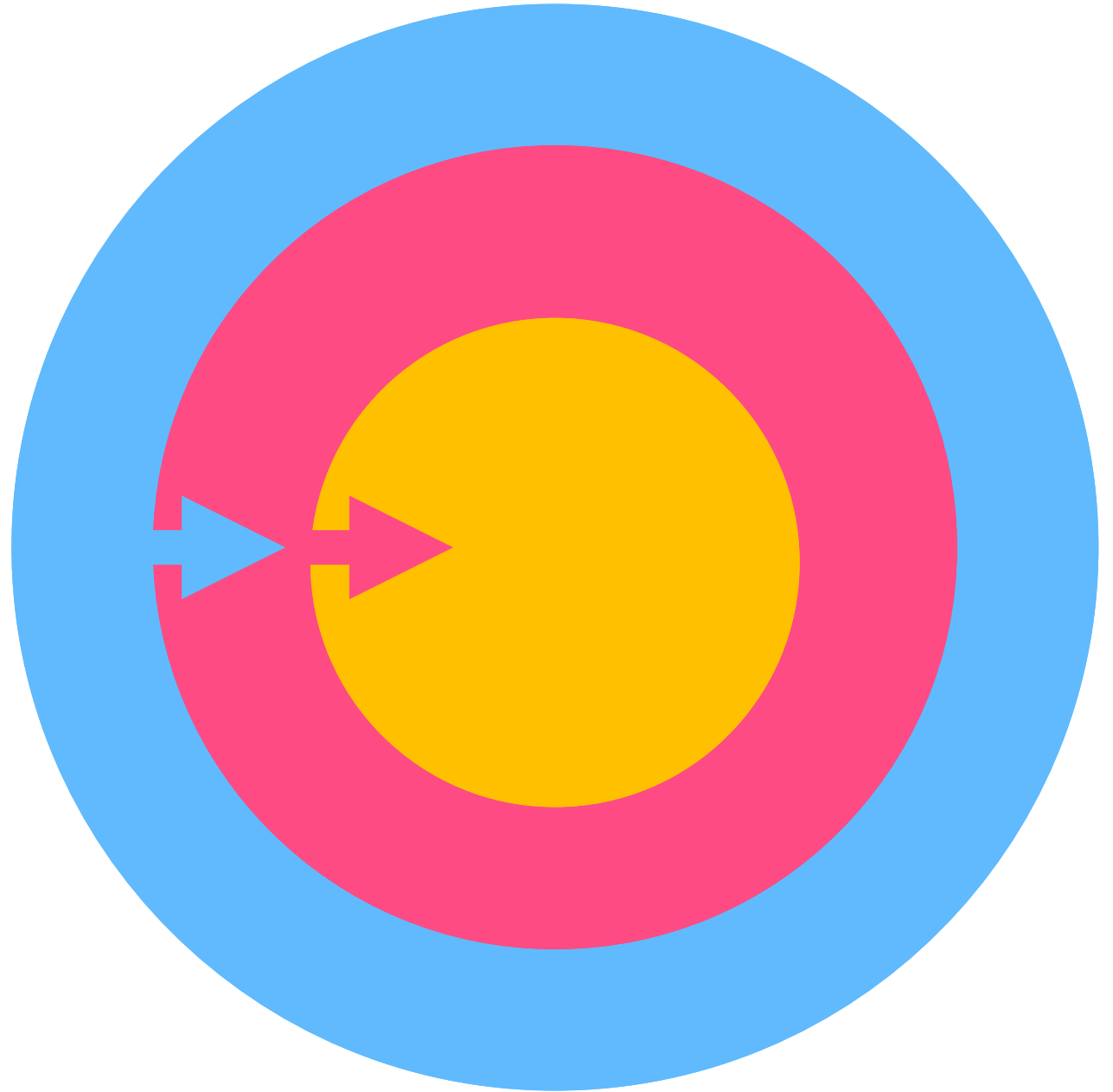
ApplicationService, NamedConstructor

Inicia/cierra transacciones, publica eventos



Infra- estructura

Controller, RepositoryImpl, DIP



Implementación de los Repository

- Adapter

Controller

- Port

Trade-off Analysis

Trade-off Analysis

Pros

- Cambiabilidad
- Testabilidad
- Mantenibilidad
- Fiable (siempre igual)

Trade-off Analysis

Pros

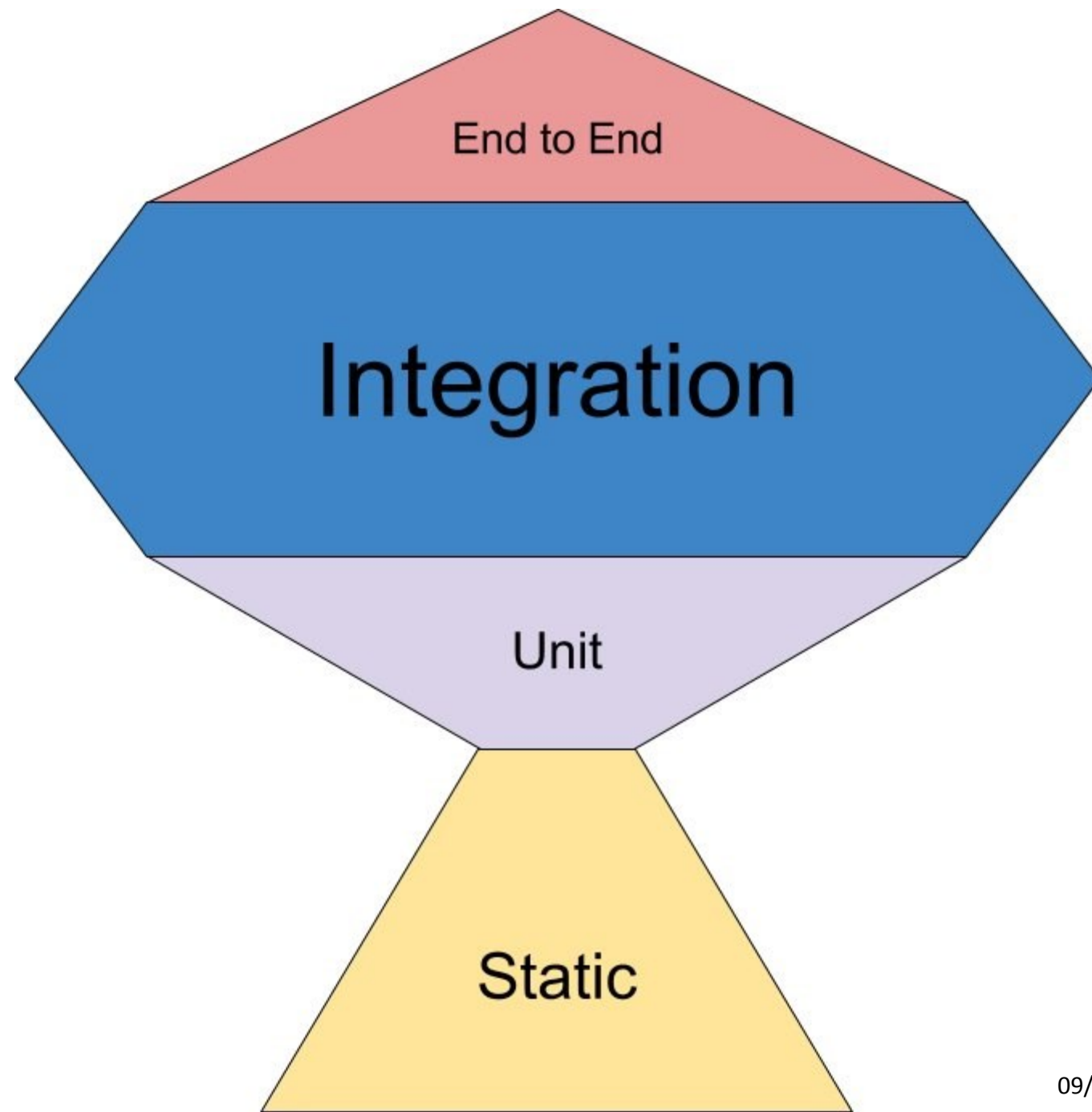
- Cambiabilidad
- Testabilidad
- Mantenibilidad
- Fiable (siempre igual)

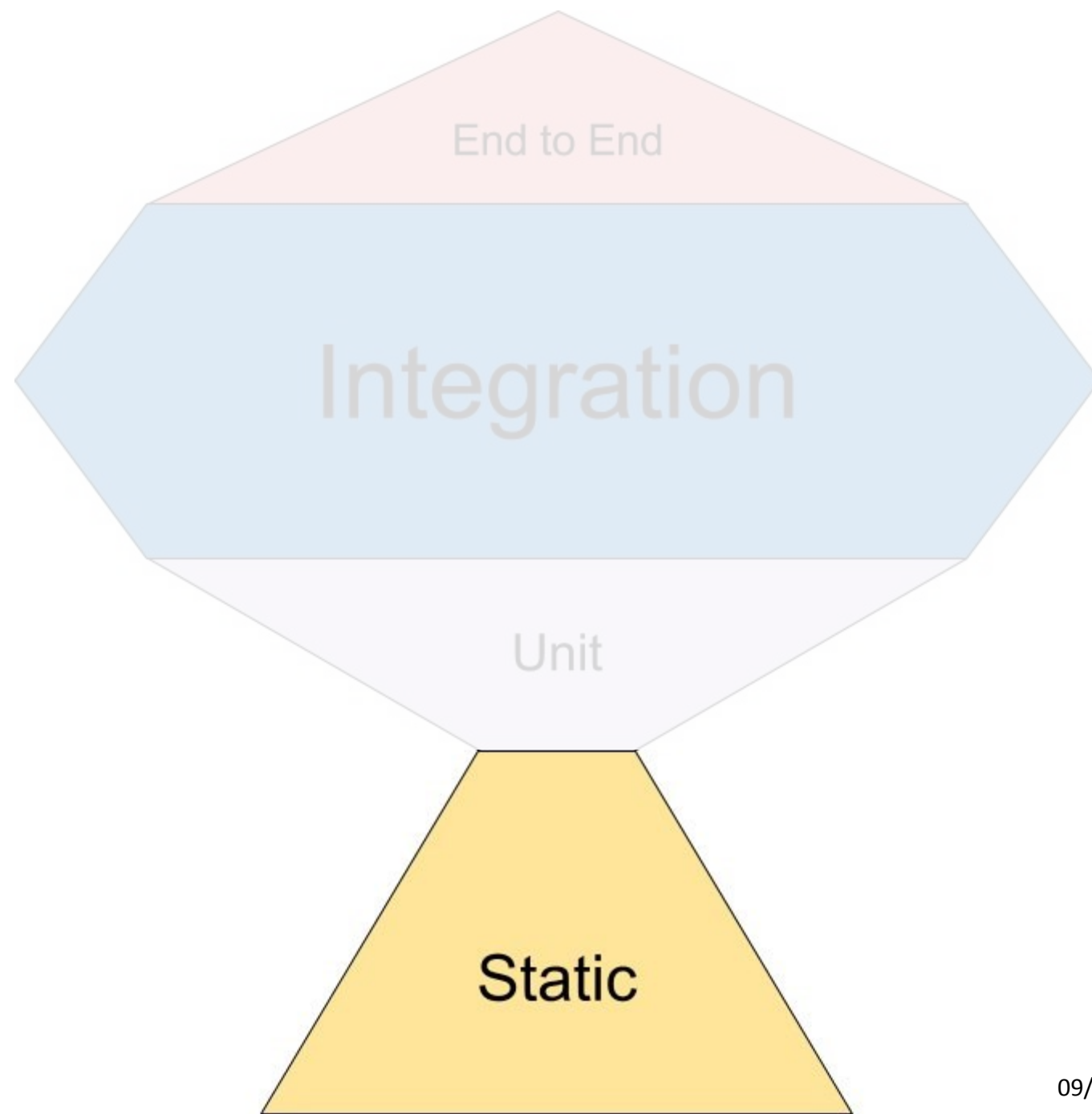
Cons

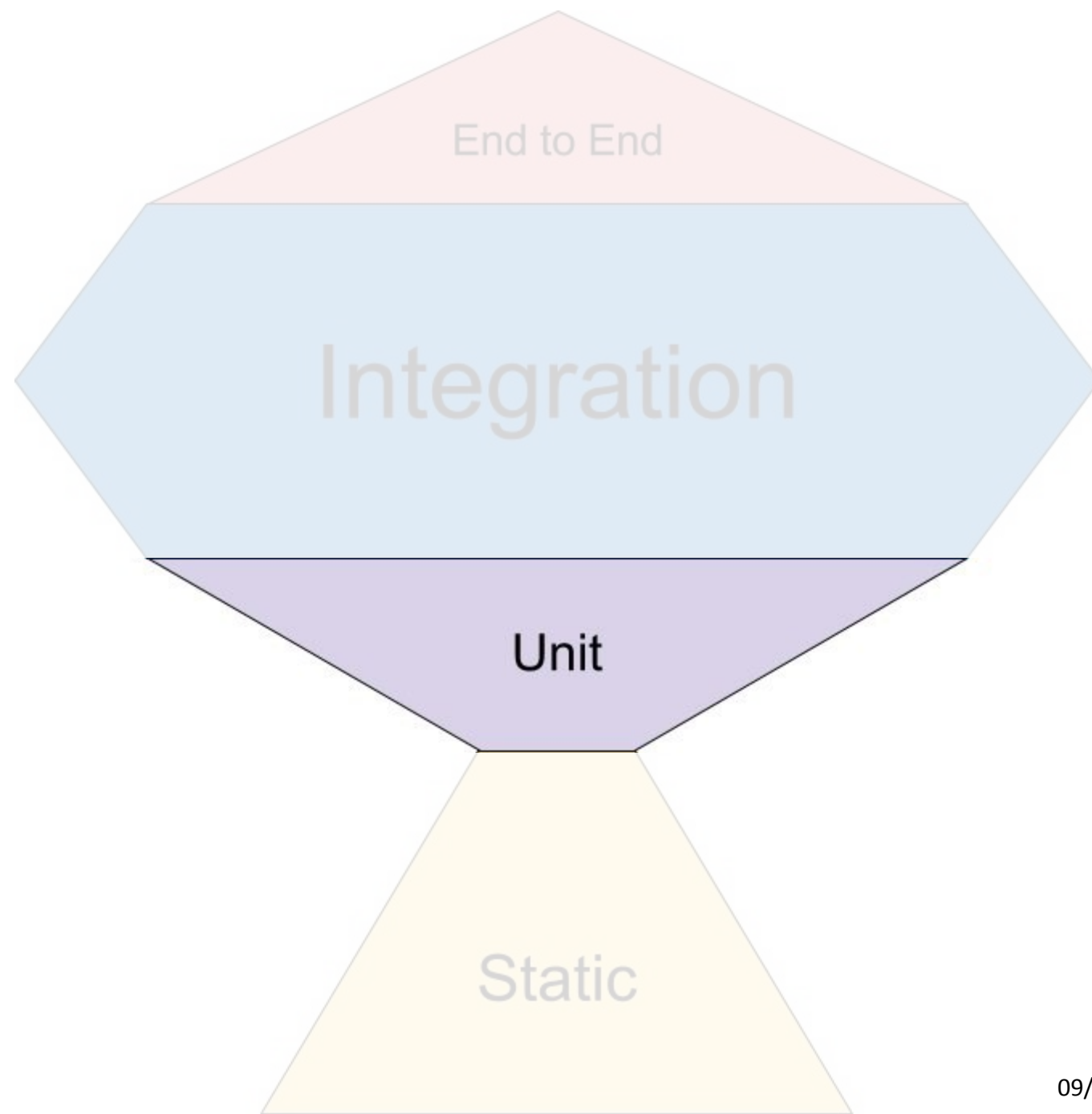
- Complejidad
 - inmediata
- Time-to-market reducido
 - Pero luego escala mejor
- Peor para prototipos

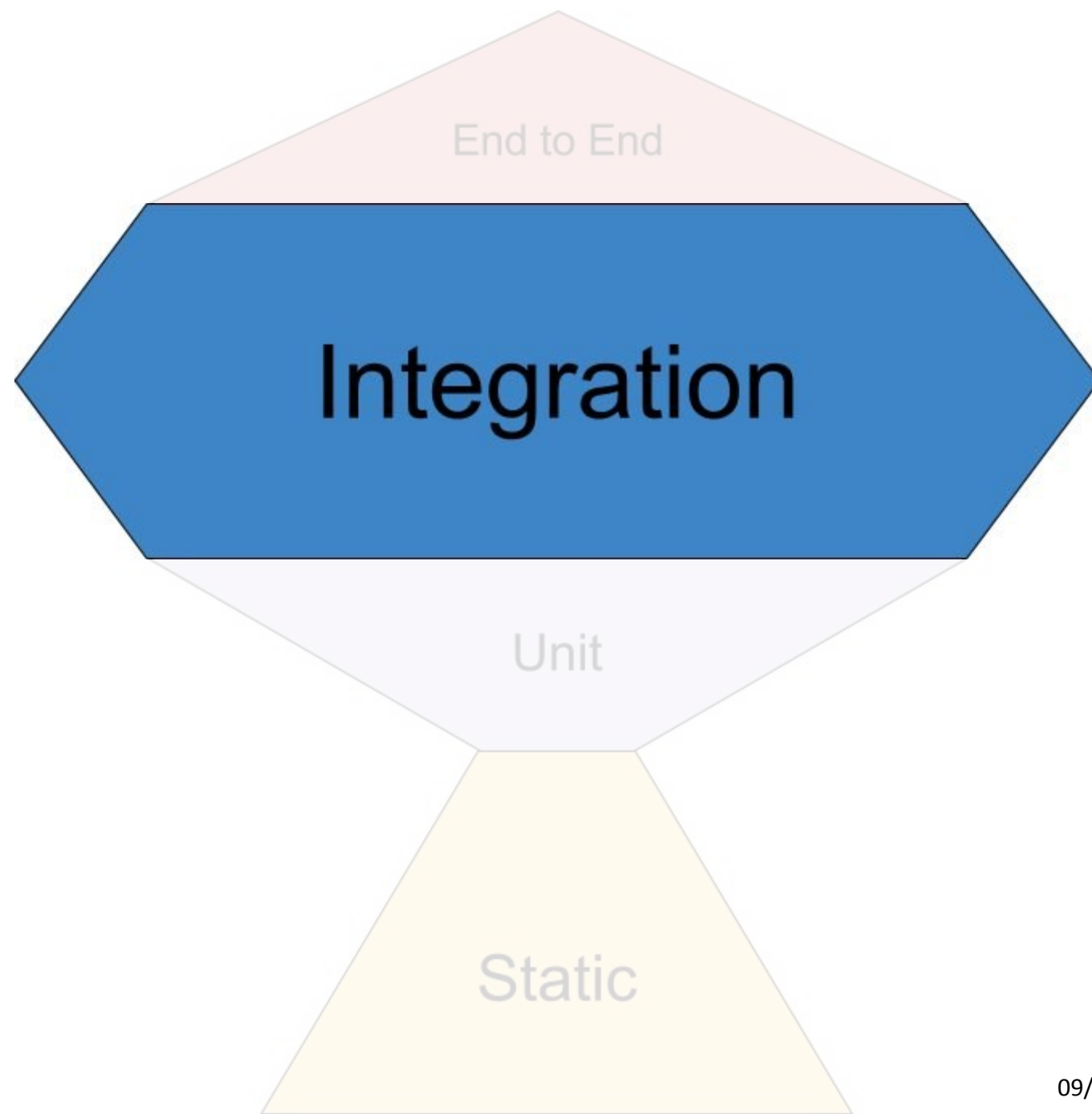
Testing

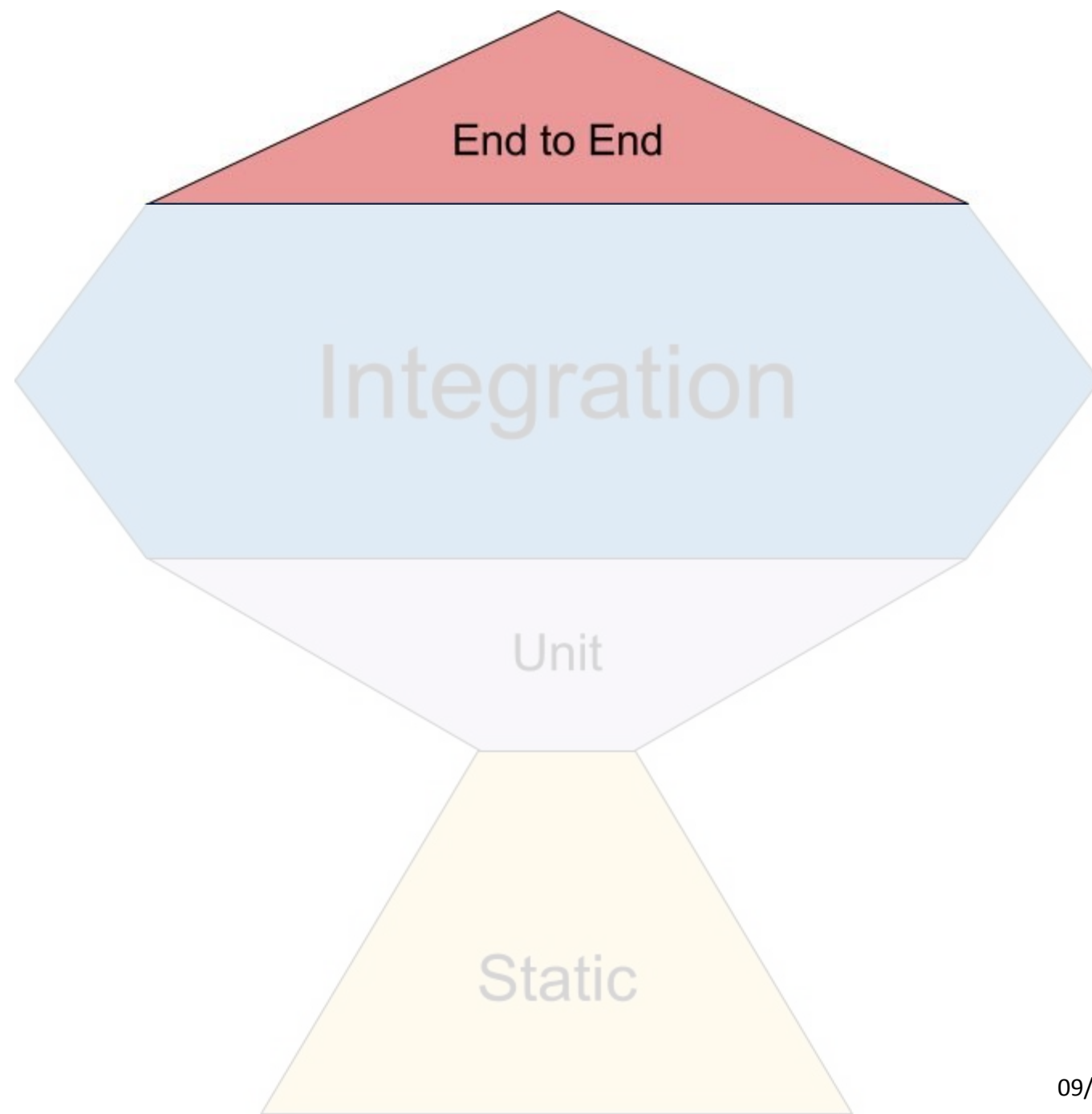
- Qué tipos de testing hay
- Propósitos y reparto de esfuerzo











Enlaces de interés

- <https://kentcdodds.com/blog/the-testing-trophy-and-testing-classifications>
- <https://kentcdodds.com/blog/write-tests>

Qué capas se testean

- **Aplicación**
 - Dominio implícitamente para no acoplarnos
- **Infraestructura**

Aceptación

Capa: Aplicación

Scope: Testear casos de uso, el happy path

Aceptación

Capa: Aplicación

Scope: Testear casos de uso, el happy path

Ejemplo: Publicar vídeo comprobará que el vídeo sí que se publica y se está validando el modelo de dominio (ValueObjects)

Unit

Capa: Aplicación

Scope: Testear casos límite, mockeando

Unit

Capa: Aplicación

Scope: Testear casos límite, mockeando

Ejemplo: Se comprueba si el vídeo es válido (si no lo es lanzará excepción)

Integración

Capa: Infraestructura

Scope: Sin mockear, el path completo, contra infra de verdad

Integración

Capa: Infraestructura

Scope: Sin mockear, el path completo, contra infra de verdad

Ejemplo: En el Docker se comprobará que se sube el vídeo a BDD, sin errores, y el vídeo luego existe

Estructura de archivos

Particiones

- Técnica
 - Un nivel
 - Por feature
- Negocio
 - Por bounded context
 - Por feature
 - O incluso ambas

Técnica - Un nivel

- application
 - PublishVideoApplicationService
- domain
 - Video
 - PublishVideoRepository
- infrastructure
 - YouTubePublishVideoRepositoryAdapter

Técnica - Por feature

- application
 - PublishVideo
 - PublishVideoApplicationService
- domain
 - Video
 - PublishVideo
 - PublishVideoRepository
- infrastructure
 - PublishVideo
 - YouTubePublishVideoRepositoryAdapter

Negocio - BoundedContext

- Video
 - application
 - Publish
 - PublishVideoApplicationService
 - domain
 - PublishVideoRepository
 - infrastructure
 - YouTubePublishVideoRepositoryAdapter

Negocio - Feature

- PublishVideo
 - application
 - PublishVideoApplicationService
 - domain
 - PublishVideoRepository
 - infrastructure
 - YouTubePublishVideoRepositoryAdapter

Negocio - BoundedContext y Feature

- Video
 - Publish
 - application
 - PublishVideoApplicationService
 - domain
 - PublishVideoRepository
 - infrastructure
 - YouTubePublishVideoRepositoryAdapter

Gustos colores

Pero es preferible no perder el foco en el negocio...

- boundedContext
 - Feature
 - application
 - UseCase
 - UseCaseApplicationService
 - Command
 - Handler
 - domain
 - AdapterRepository
 - UseCaseDomainService
 - infrastructure
 - controller
 - UseCaseController
 - mapper
 - InfraToDomainModelMapper
 - EventPublisherImpl
- shared - compartido entre bounded contexts
 - application
 - domain
 - infrastructure
 - persistence
 - AdapterRepositoryImpl

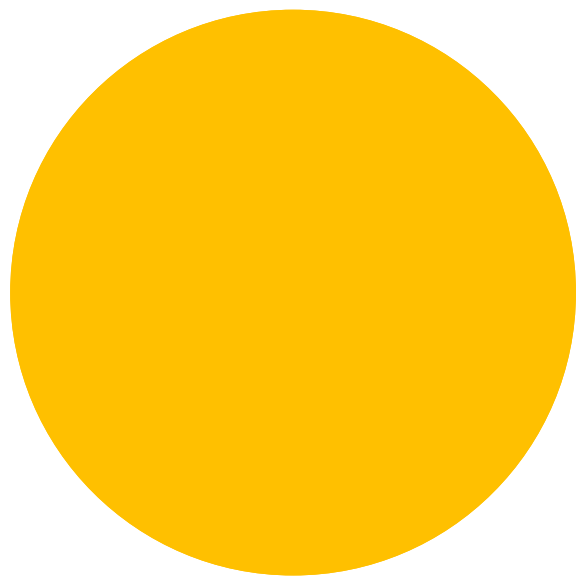
Hemos visto...

Hemos visto...

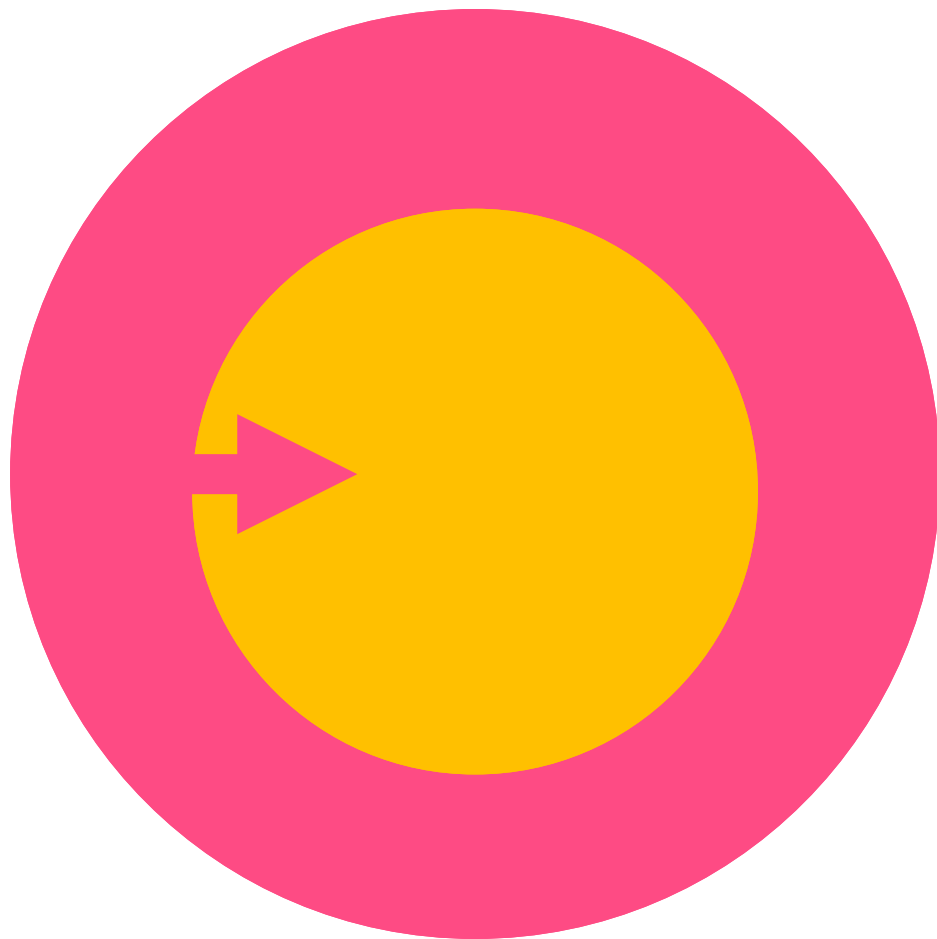
- DDD -> lo técnico se adapta al negocio
 - Se utiliza lenguaje de negocio para describir lo técnico, no al revés

Hemos visto...

- DDD -> lo técnico se adapta al negocio
 - Se utiliza lenguaje de negocio para describir lo técnico, no al revés
- Arquitectura Hexagonal
 - Infraestructura -> Aplicación (caso de uso) -> Dominio
 - Testing

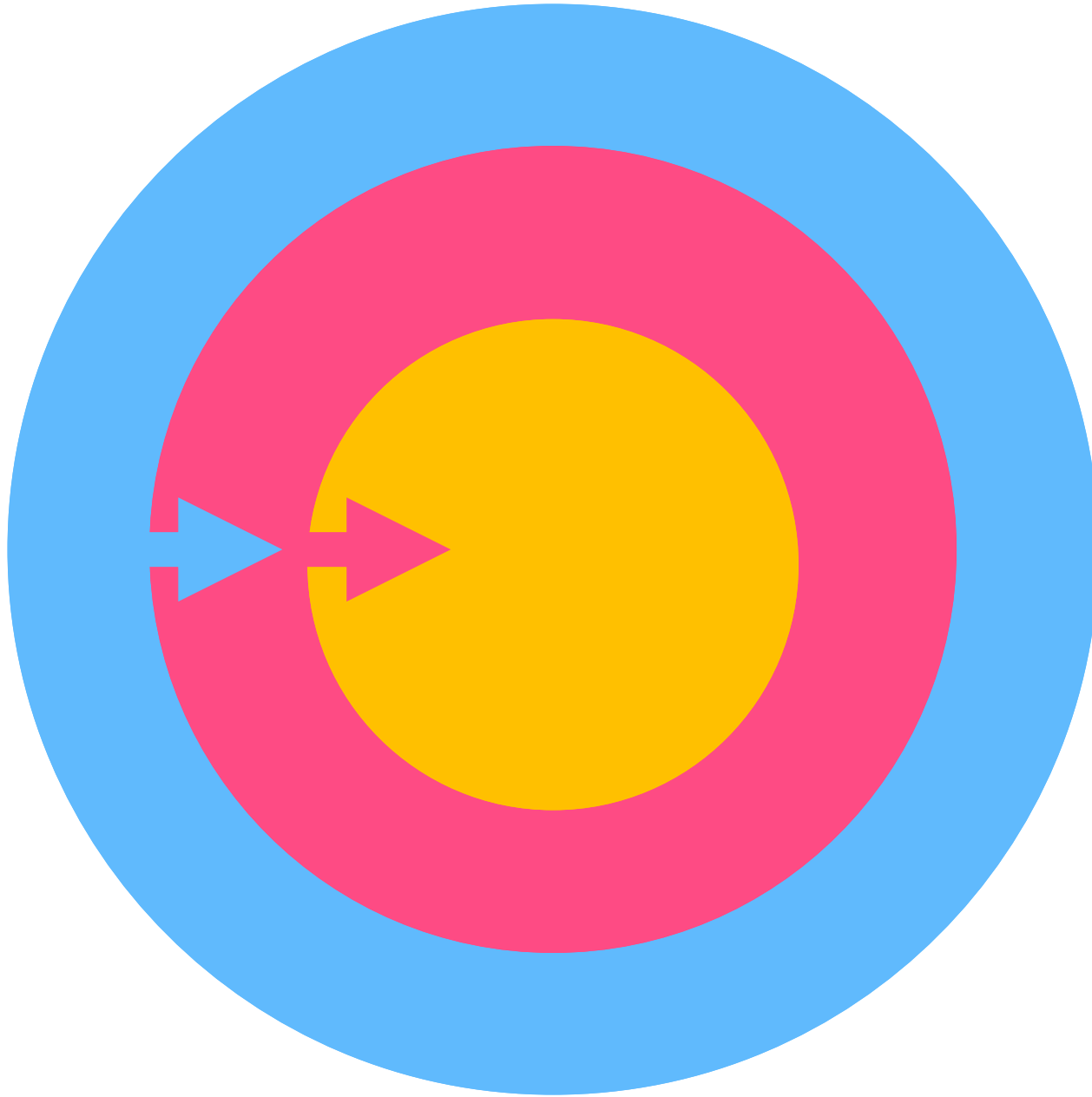


DOMINIO



DOMINIO

APLICACIÓN



DOMINIO

APLICACIÓN

INFRAESTRUCTURA

Literatura Técnica

- Domain-Driven Design
 - Implementing y distilled
- Clean Architecture
- CodelyTV

Elementos de arquitectura limpia en Java

Mapper, Handler, Resolver, Circuit Breakers, Adapters, Response, ValueObjects, NamedConstructors, Publisher, Mocking y muchos más

Lecturas al respecto:

- Implementing Domain-Driven Design
- Design Patterns of Reusable Elements of Object-Oriented Software
- Patterns of Enterprise Application Architecture

¡¡Gracias por tu tiempo!!