Mejor Framework Python OO + SOLID

aleasoluciones

Vivir sin Frameworks

Presentaciones

Alea Soluciones
Bifer Team

@eferro
@pasku1
@apa42
@nestorsalceda

Un placer...

¿Qué es un framework?

Django

Rails

Grails

Spring

AppEngine

Zope/Plone

¿Qué ventajas nos aporta?

Aporta **UNA** solución (única)

Estructura tu aplicación

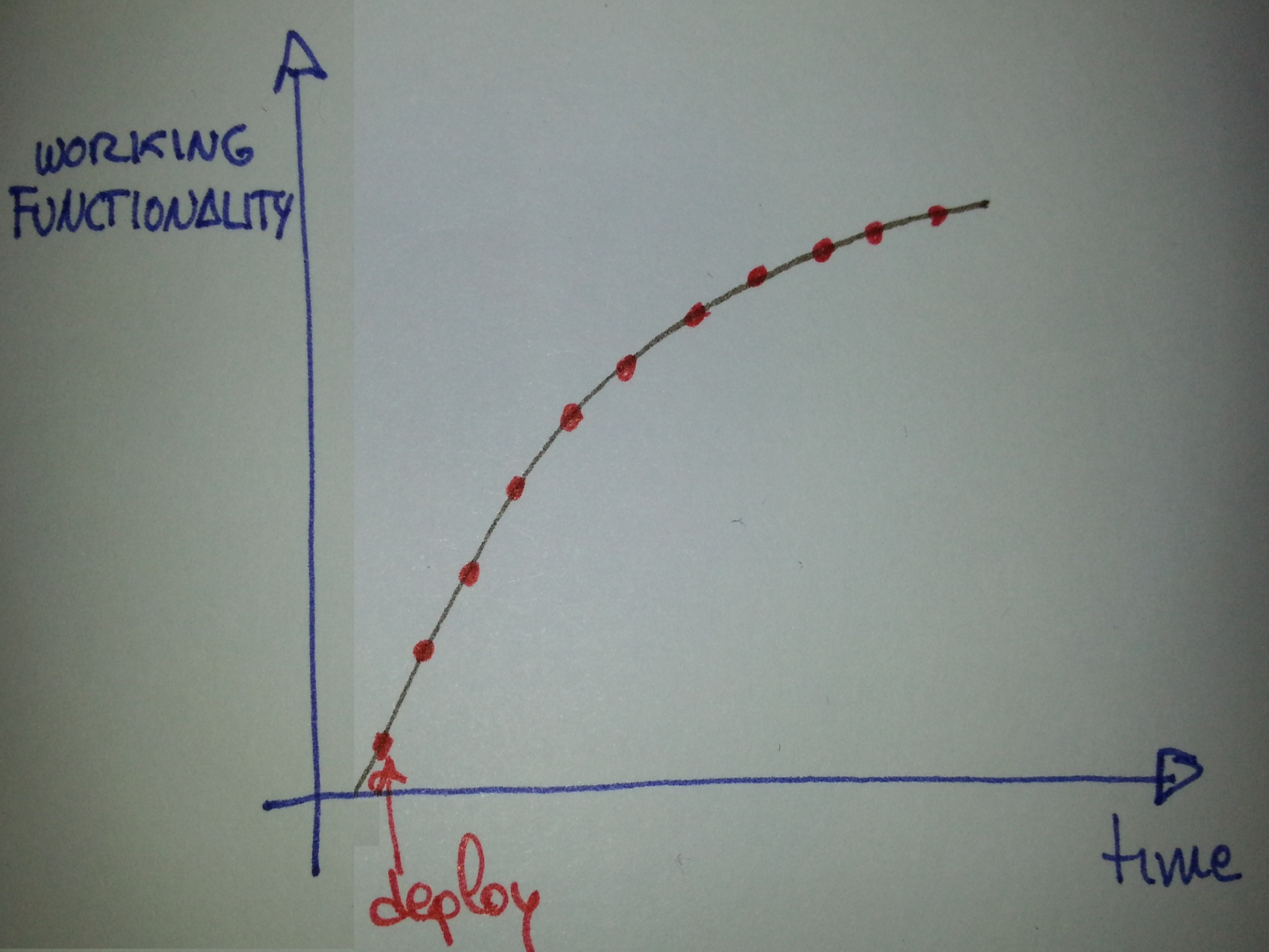
¿Qué inconvenientes tiene?

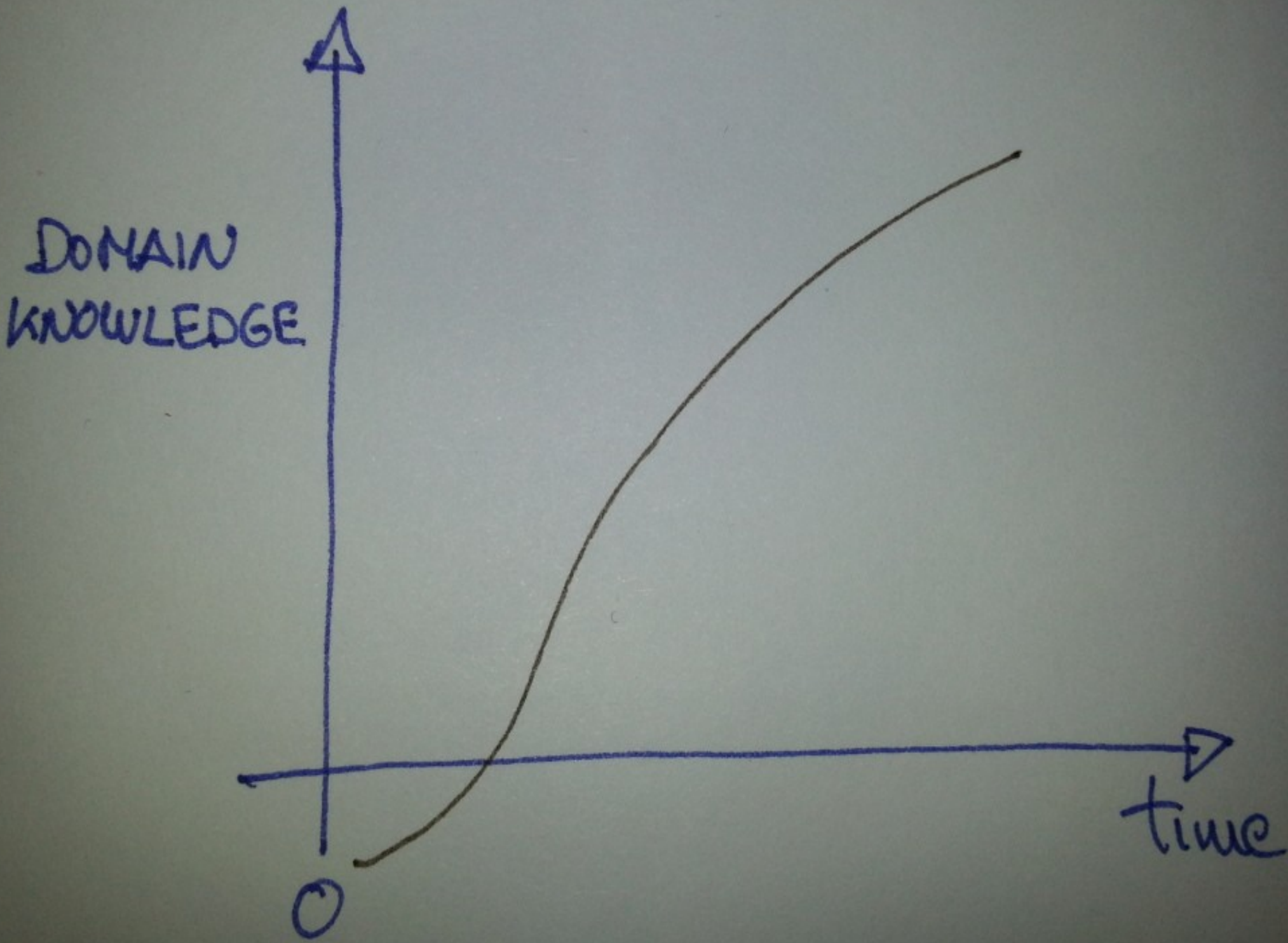
Crea **GRAN** dependencia

Limita flexibilidad negocio

Obsolescencia / Moda ???

Desarrollo Ágil
Crecimiento Orgánico
Arquitectura Emergente


WORKING
FUNCTIONALITY
deploy
time


DOMAIN
KNOWLEDGE
O
time


I SEE FRAMEWORKS
EVERYWHERE
memegenerator.net

Lo único seguro
Todo Cambia / Evoluciona

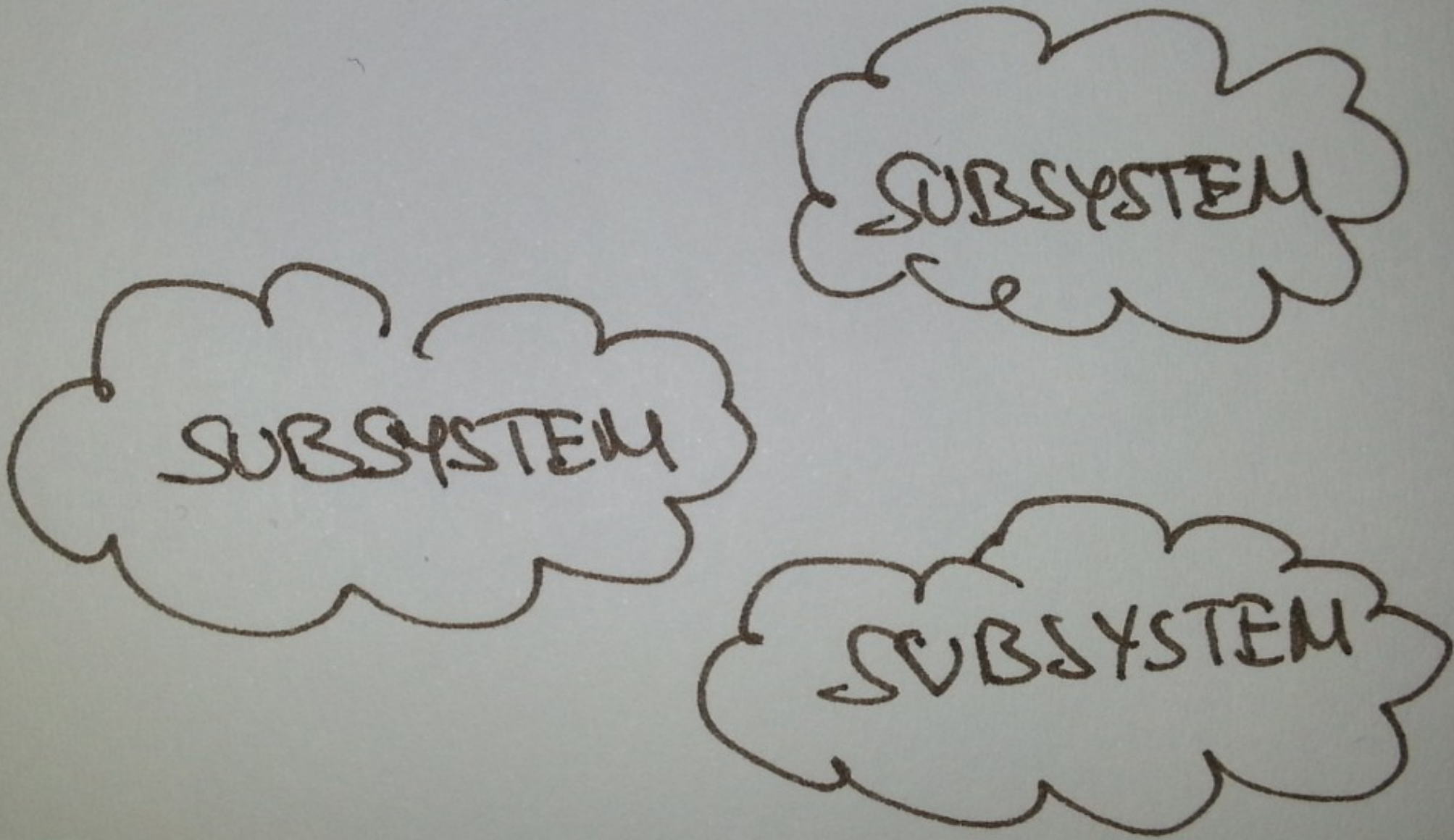
Existe una alternativa

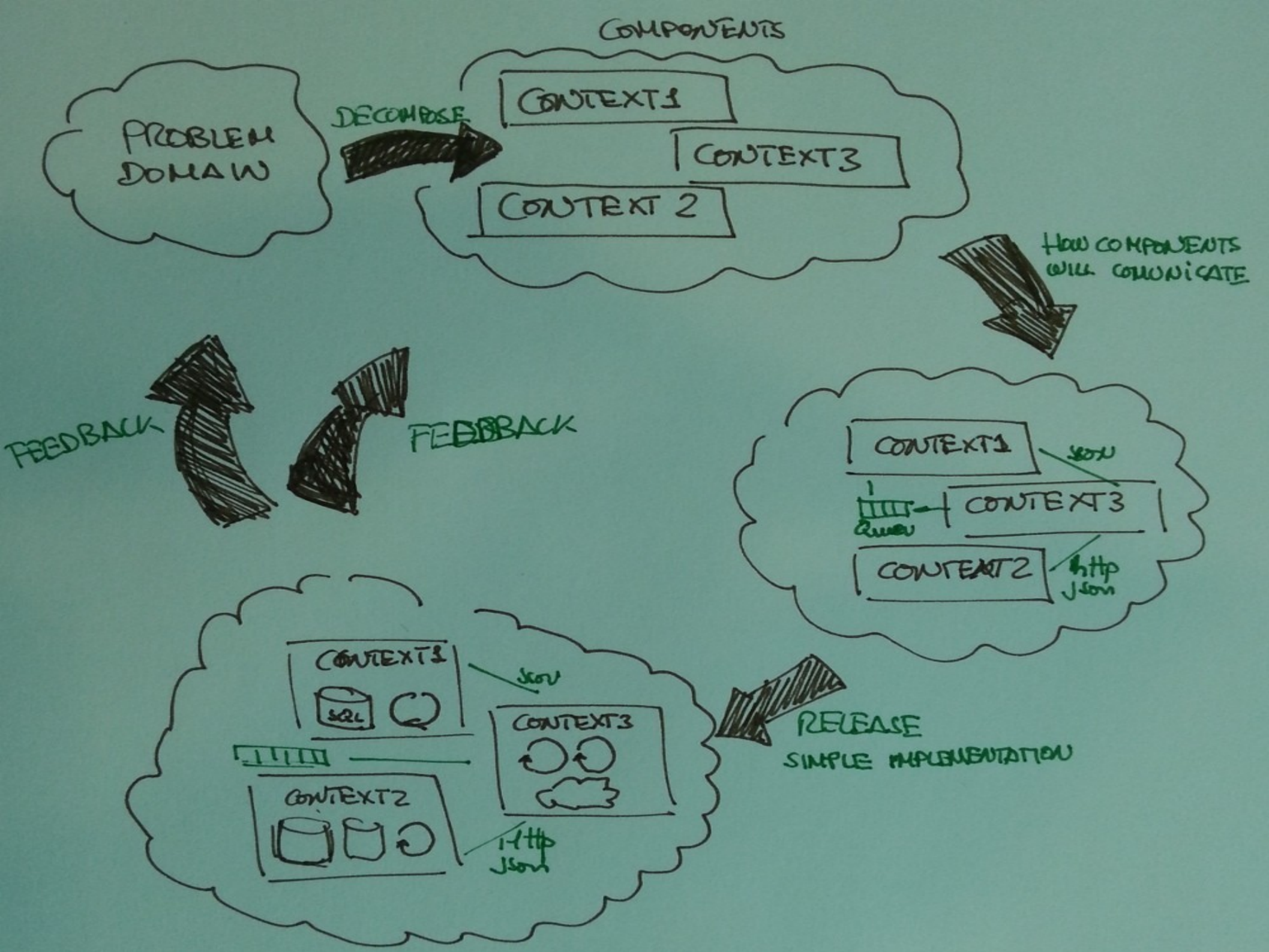
Centrarnos en el Negocio
y
Desarrollo ágil / evolutivo

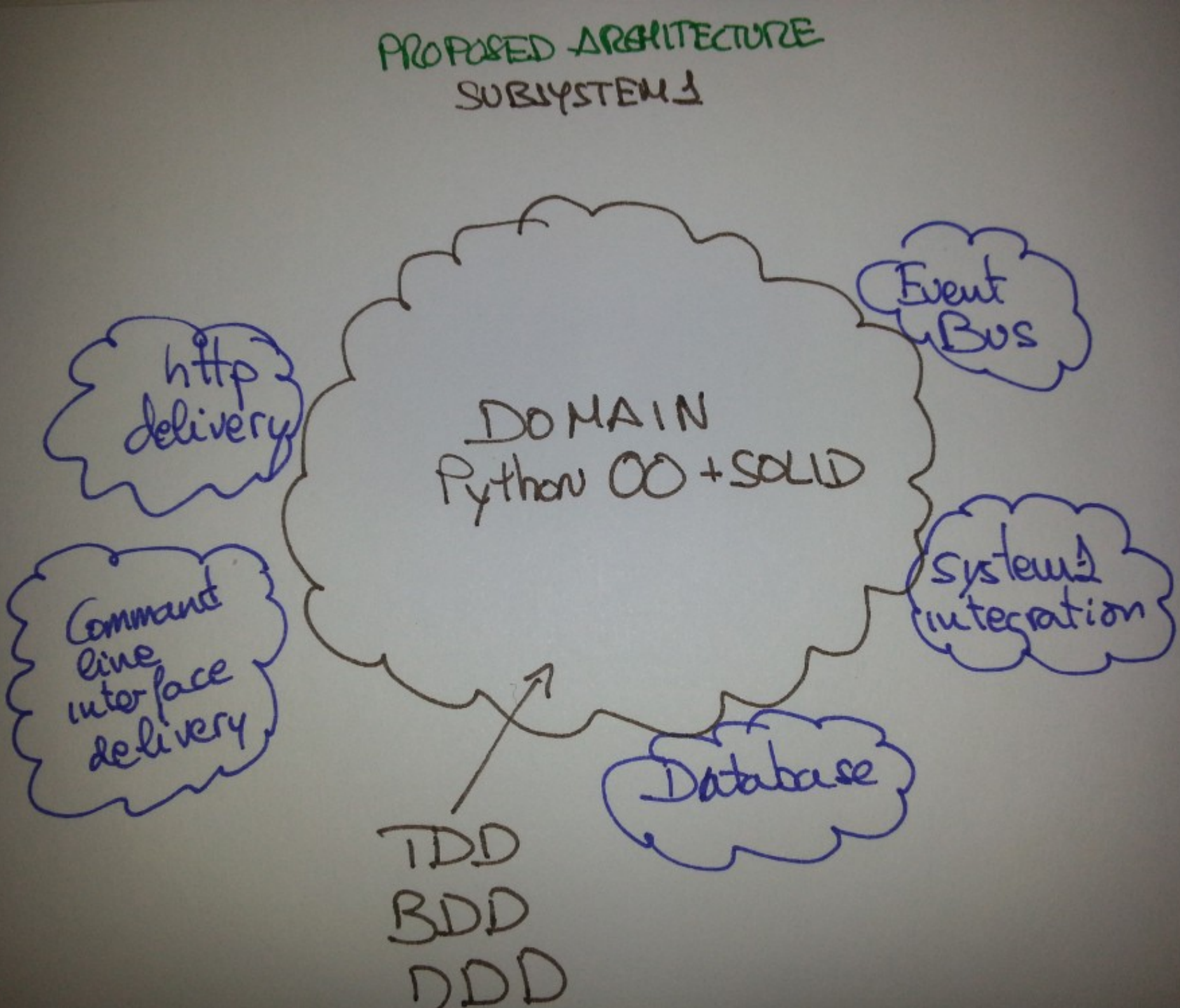
Poder postponer decisiones

Tomar decisiones conscientemente

PROPOSED ARCHITECTURE




COMPONENTS
PROBLEM DOMAIN
DECOMPOSE
CONTEXT1
CONTEXT3
CONTEXT 2
HOW COMPONENTS WILL COMMUNICATE
FEEDBACK
FEEDBACK
CONTEXT1
JSON
CONTEXT3
CONTEXT2
http
JSON
CONTEXT1
SQL
JSON
CONTEXT3
CONTEXT2
http
JSON
RELEASE
SIMPLE IMPLEMENTATION


PROPOSED ARCHITECTURE
SUBSYSTEM 1
DOMAIN
Python OO + SOLID
http delivery
Event Bus
Command line interface delivery
System integration
Database
TDD
BDD
DDD

¿Cómo desarrollamos negocio?

DDD Domain Driven Design

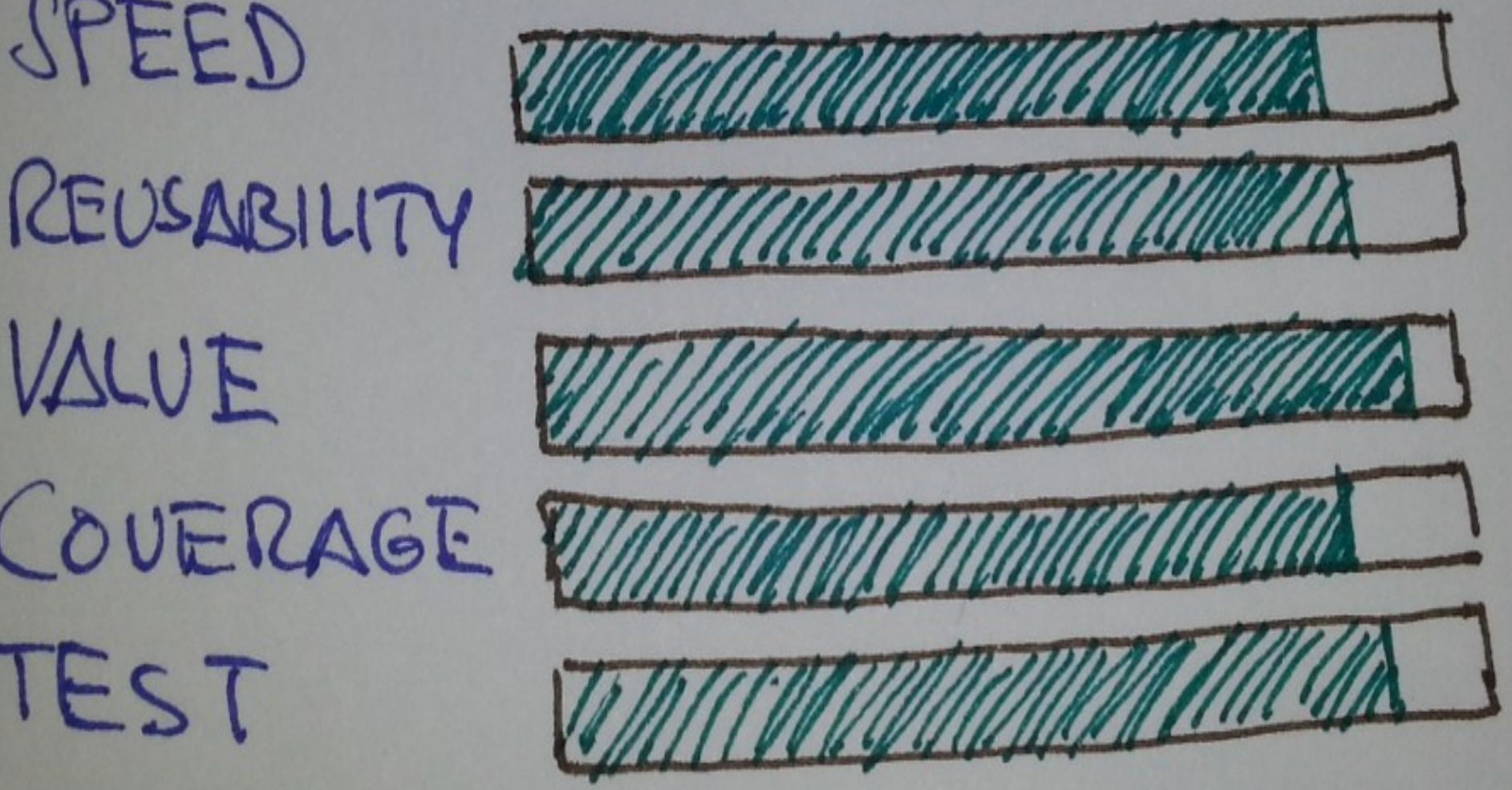
Clean Code / S.O.L.I.D

TDD Test Driven Development

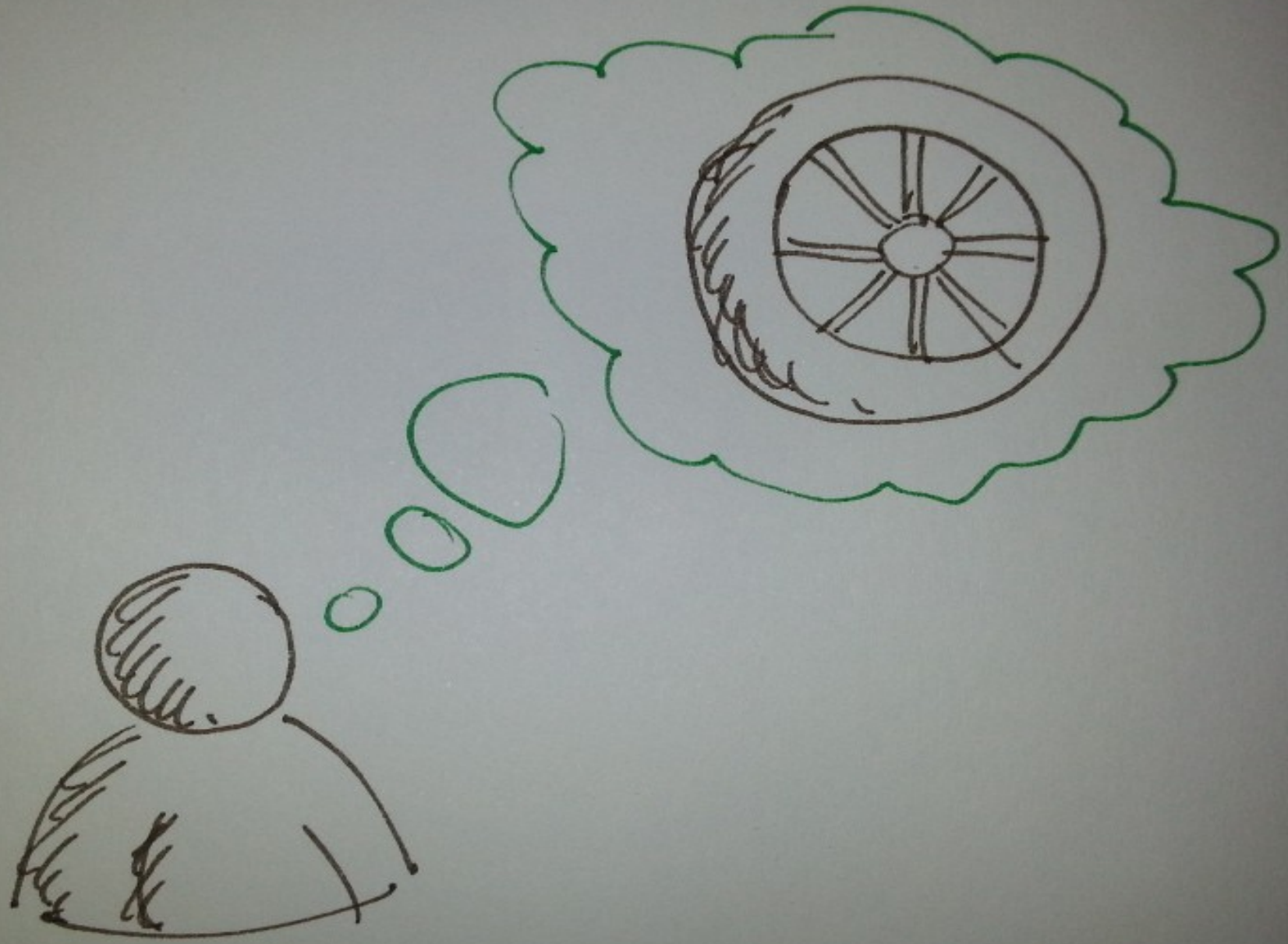
Pure Object Oriented Python

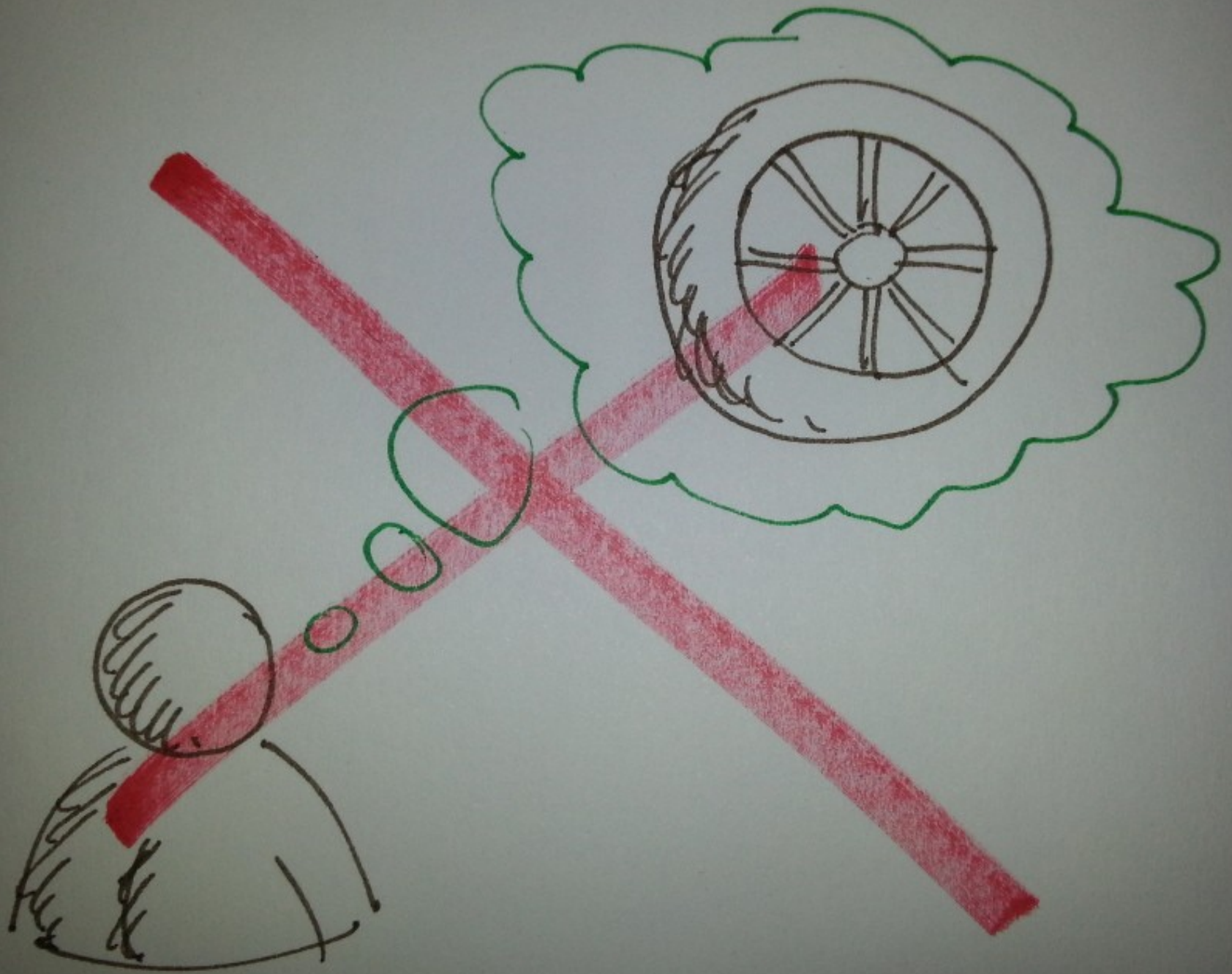
Independiente de IO
(No net, No BD, No files, ...)

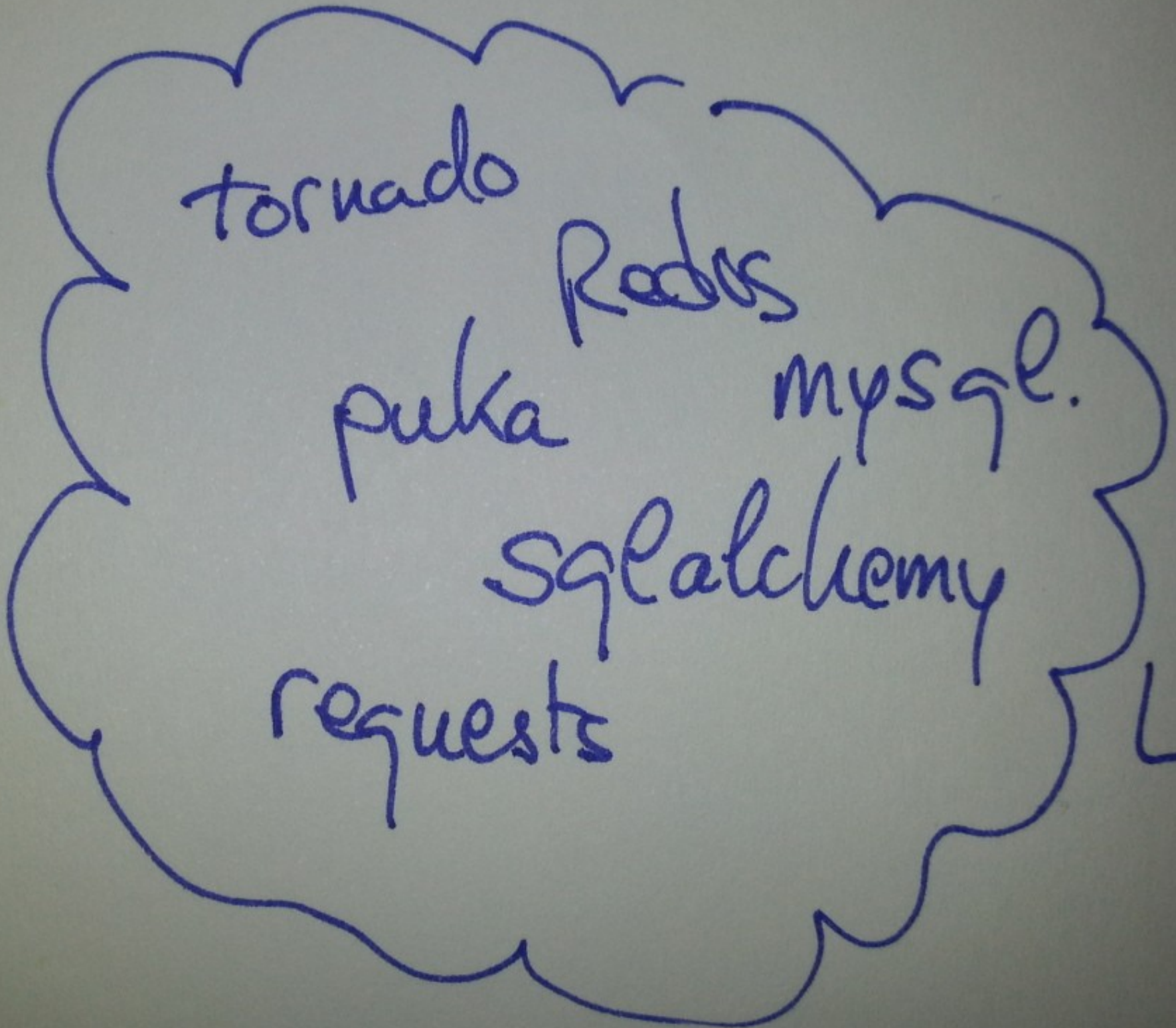
DOMAIN PYTHON OO + SOLID



¿Cómo hacemos el resto?






tornado
Redis
puka
mysql.
sqlalchemy
requests
Libs

EXTERNAL
LIB
WRAPPER
APLICATION
CONTRACT
TEST

Infraestructura y Entrega

Usamos las abstracciones sobre libs

Clean Code / S.O.L.I.D

TDD Test Driven Development

Conclusiones

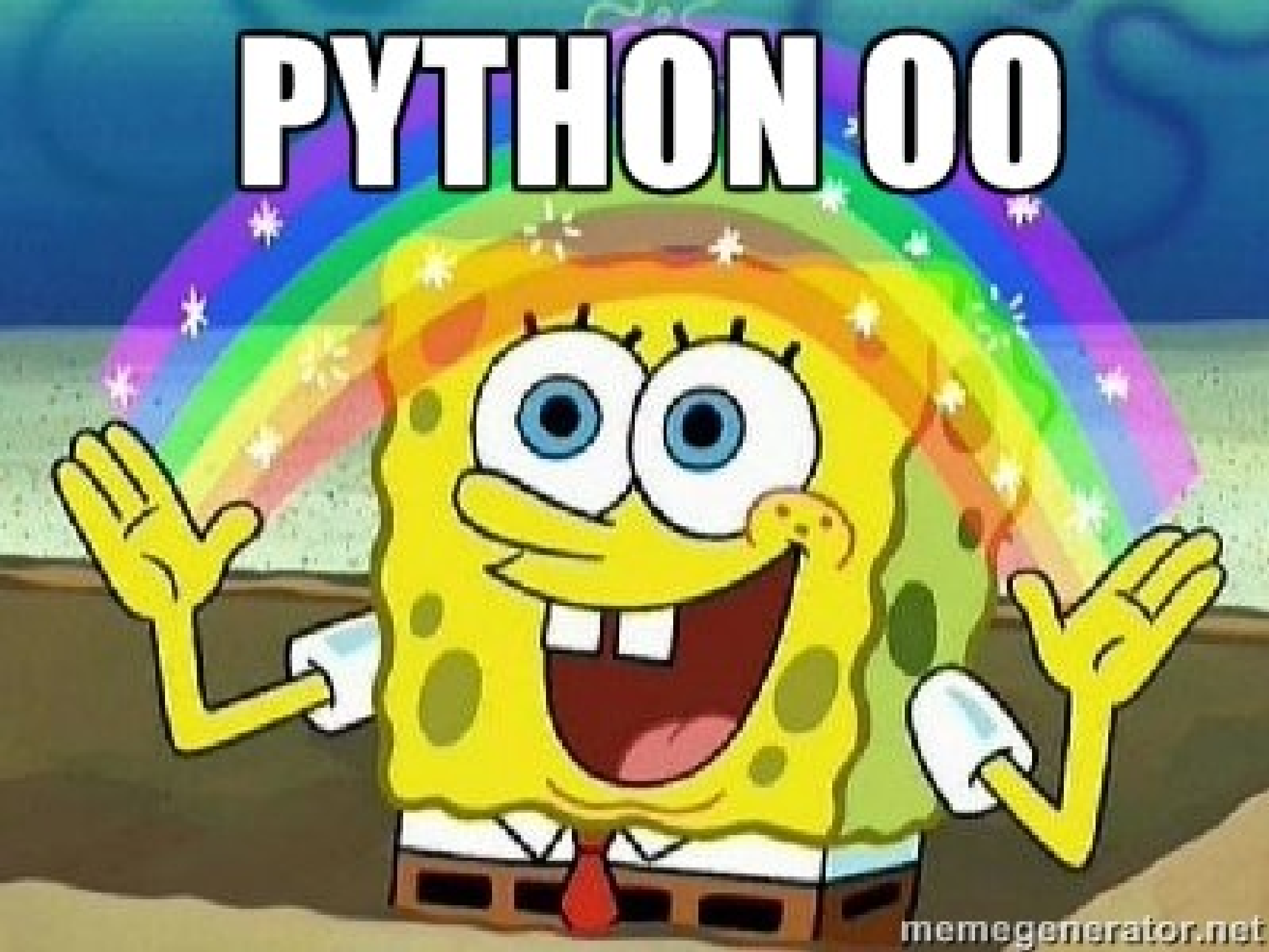
- Los frameworks son una decisión muy importante
    - Las decisiones tienen que ser conscientes

Nuestra forma de trabajo y arquitectura debe permitirnos:

- Hacer un sistema que evolucione con las necesidades del cliente
- Postponer decisiones
- Aplicar distintas soluciones técnicas

Existen alternativas (Reales)

Que permiten centrarse en negocio
Que facilitan el desarrollo ágil
Que evolucionan a largo plazo
Que evolucionan a buena velocidad


PYTHON OO
memegenerator.net

Referencias

DDD

http://en.wikipedia.org/wiki/Behavior-driven_development

TDD

http://en.wikipedia.org/wiki/Test-driven_development

Hexagonal Architecture

http://alistair.cockburn.us/Hexagonal+architecture

Clean Architecture

http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html


PREGUNTAS?
memegenerator.net

Gracias !!!

aleasoluciones

Mejor Framework Python OO + SOLID

aleasoluciones

Vivir sin Frameworks

Presentaciones

Alea Soluciones
Bifer Team

Hacemos producto
Telecomunicaciones
Sistemas + Software
Extreme Programming
Aportamos valor

@eferro
@pasku1
@apa42
@nestorsalceda

Un placer...

¿Qué es un framework?

Django

Rails

Grails

Spring

AppEngine

Zope/Plone

Framework (tal y como lo entendemos)
- Intenta solucionar un problema concreto:
    - el problema que tenia el que lo creo
    - No hay dos problemas iguales
    - los problemas crecen (o por lo menos cambian)
- Llama a tu código
- Te hace depender de él (tanto como pueda)
(herencia, callbacks, estructura fuentes, tipo de BD)
- Estructura tu aplicación
- Define una forma de desarrollar / testear
- Aporta muchas soluciones prehechas

La mayor parte se centra en:
Aplicaciones centradas en datos con interface usuario Web

El framework te suele “empujar” a usarlo de cierta forma

¿Qué ventajas nos aporta?

Aporta **UNA** solución (única)

Estructura tu aplicación

Ventajas
No necestias pensar demasiado inicialmente
Al principio pareces ir muy rápido

¿Qué inconvenientes tiene?

Crea **GRAN** dependencia

Limita flexibilidad negocio

Obsolescencia / Moda ???

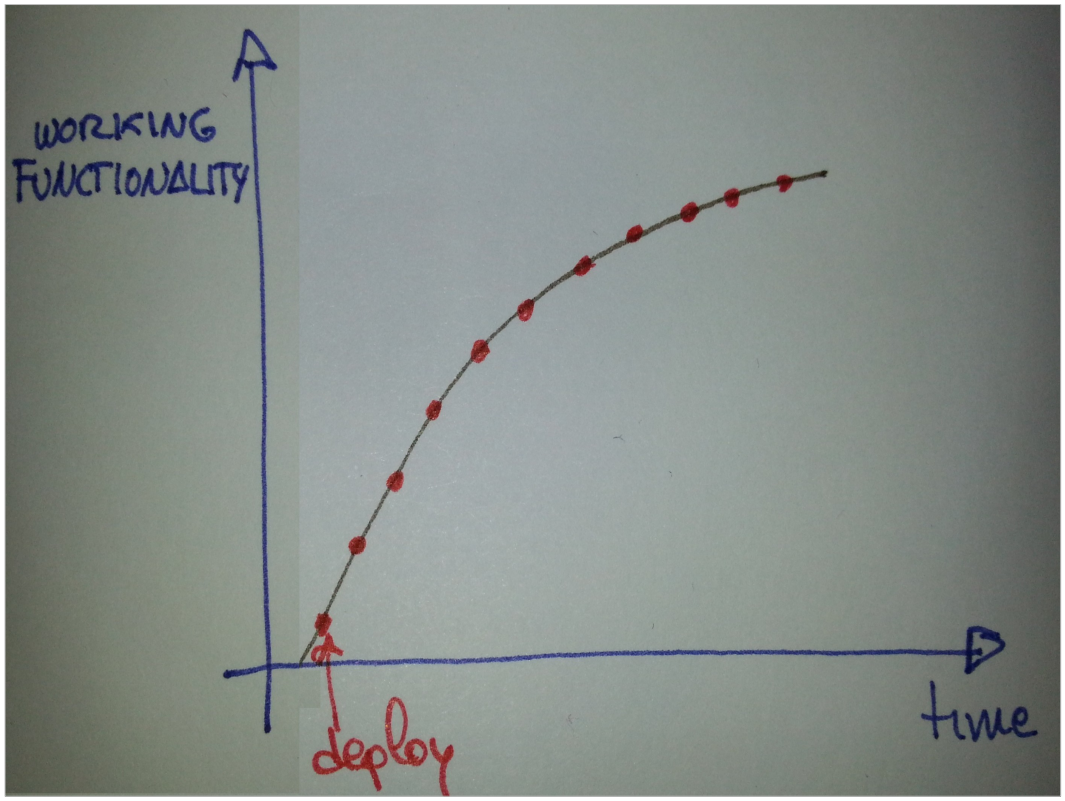
Desventajas
Vendes tu “alma”
Te terminas centrandote en la solución técnica en vez de en el negocio del cliente
Pierdes flexibilidad

Decisiones técnicas siempre son TradeOffs
Corto plazo vs Largo plazo
Flexibilidad vs Dirección marcada
Centrarse en tecnologia vs Centrarse en negocio
Crecimiento organico/evolutivo vs Desarrollo en Fases/Implantación

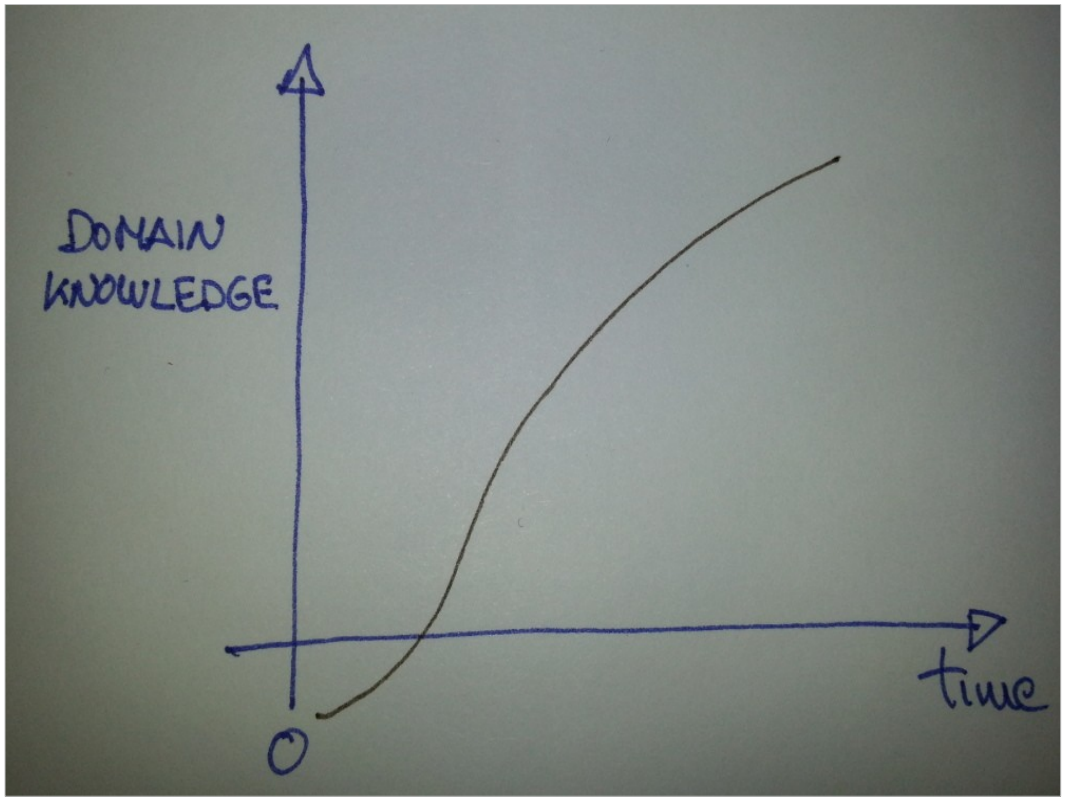
Usar un framework es un GRAN trade-off

El Cliente te paga por solucionar SU problema (Negocio / Dominio …) no por usar un framework (aunque esté de moda)

Desarrollo Ágil
Crecimiento Orgánico
Arquitectura Emergente



La evolución es:
Sabemos muy poco
Desarrollamos lo mínimo posible / Desplegamos
Conseguimos feedback
Repetimos



Nuestros conocimientos de negocio crece
Las funcionalidades se acumulan
El necesidades cambian

Al principio del proyecto no sabemos nada.
Cómo coño elegimos el Framework !!!

Lo eliges porque:
es lo que conoces???
está de moda???

Si inicialmente no tenemos conocimiento de negocio
Cómo vamos a saber lo que necesitamos técnicamente ?
Cómo va a saber el creador de framework lo que necesitamos ?
Nos va a aportar productividad o va a secuestrar la evolución futura ?

Somos conscientes del compromiso que es usar un framework?

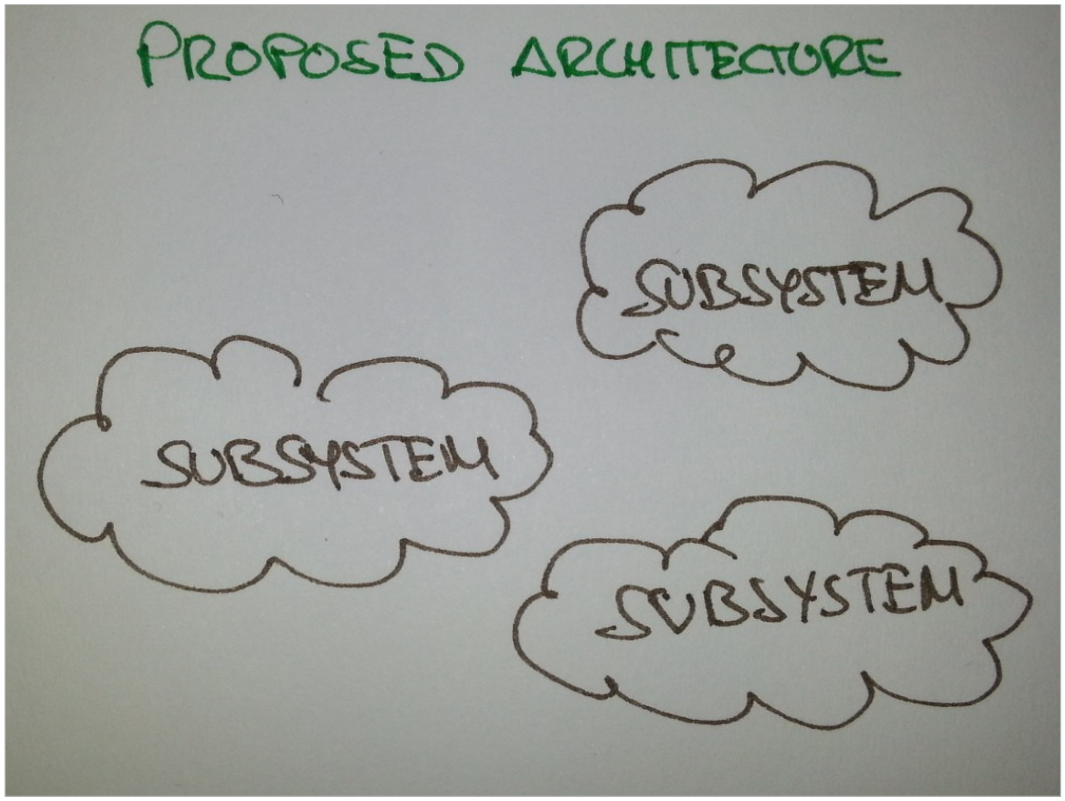
Lo único seguro
Todo Cambia / Evoluciona

Existe una alternativa

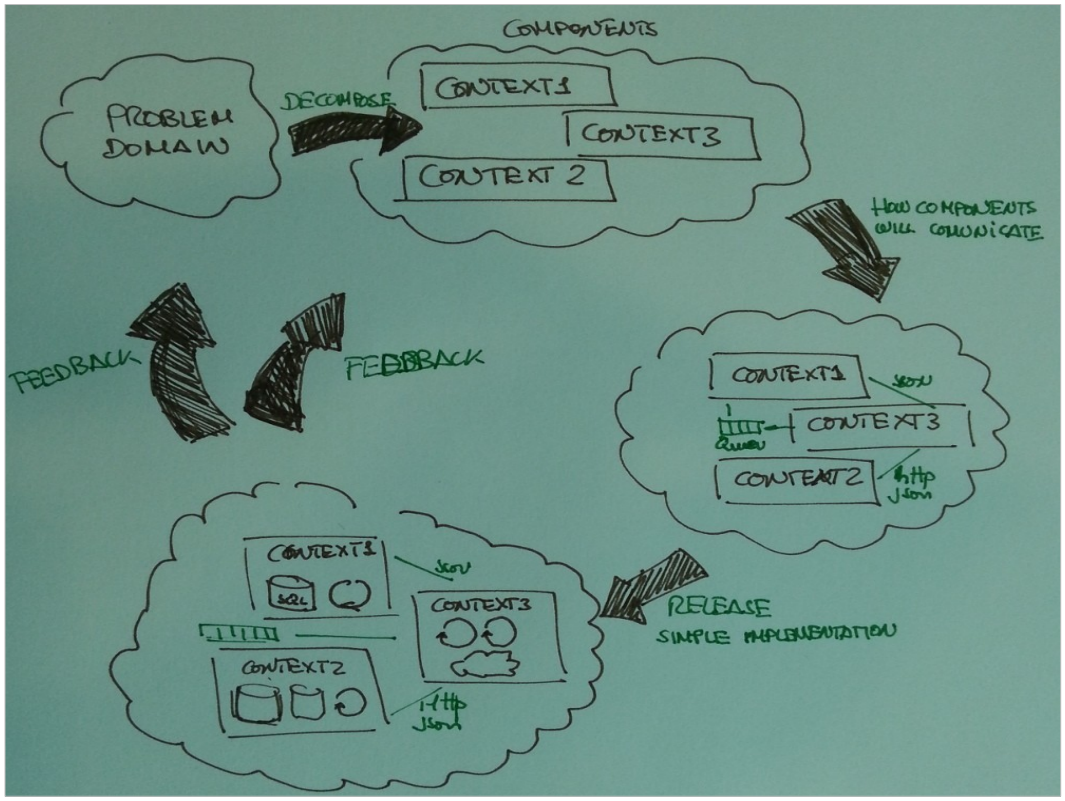
Centrarnos en el Negocio
y
Desarrollo ágil / evolutivo

Poder postponer decisiones

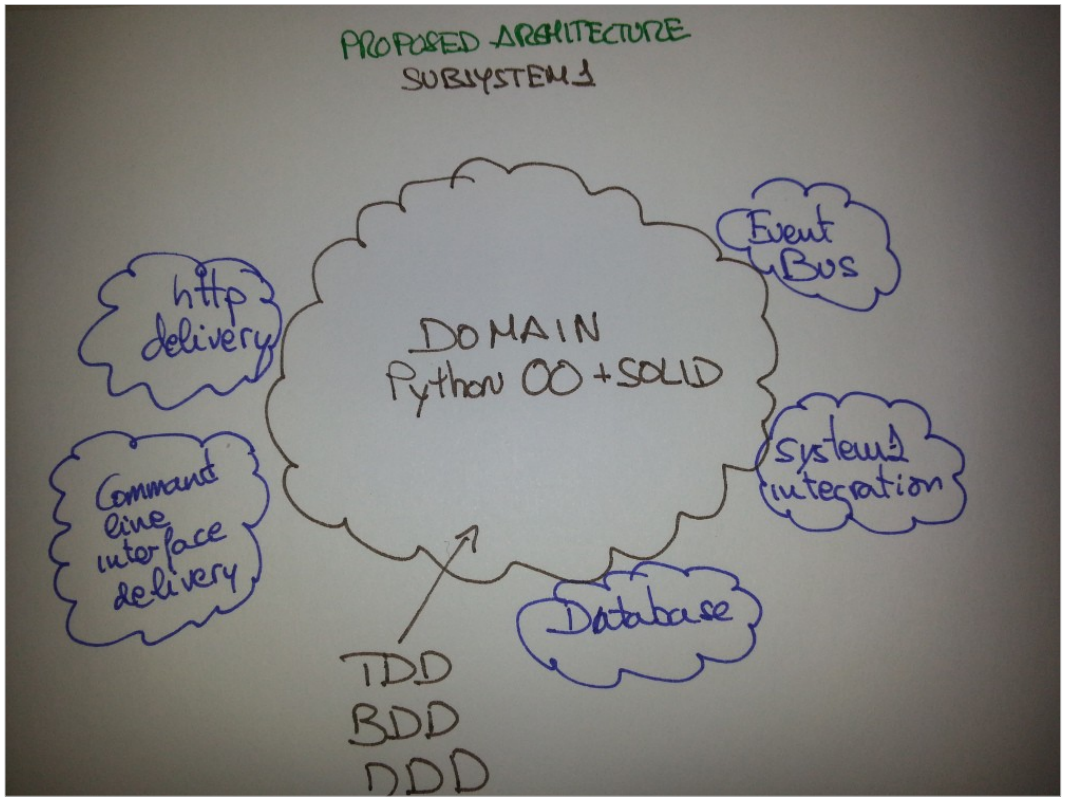
Tomar decisiones conscientemente



Un sistema está compuesto por subsistemas (excepto si es trivial)
Diferentes subsistemas NO es diferentes módulos usando el mismo esquema de BD y compartiendo los datos



Objetivo
Ciclo desarrollo / release / feedback muy rápido
Cada implementación debe ser mínima
Flexible
Para que no cueste cambiar decisiones técnicas
Para adaptarnos a Cualquier requisito



Cada subsistema lo desarrollamos con este esquema
Nos centramos en Dominio de cliente / Lógica de Negocio

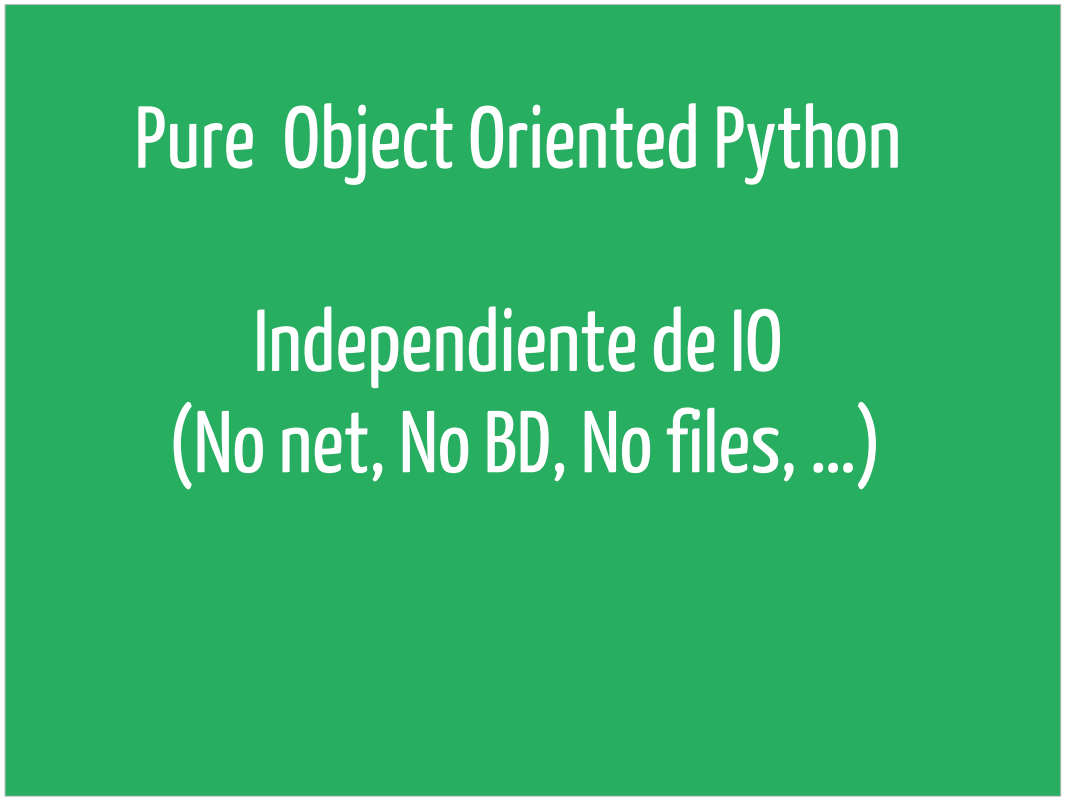
Arquitectura
Limpias
Desacoplada
Delivery

¿Cómo desarrollamos negocio?

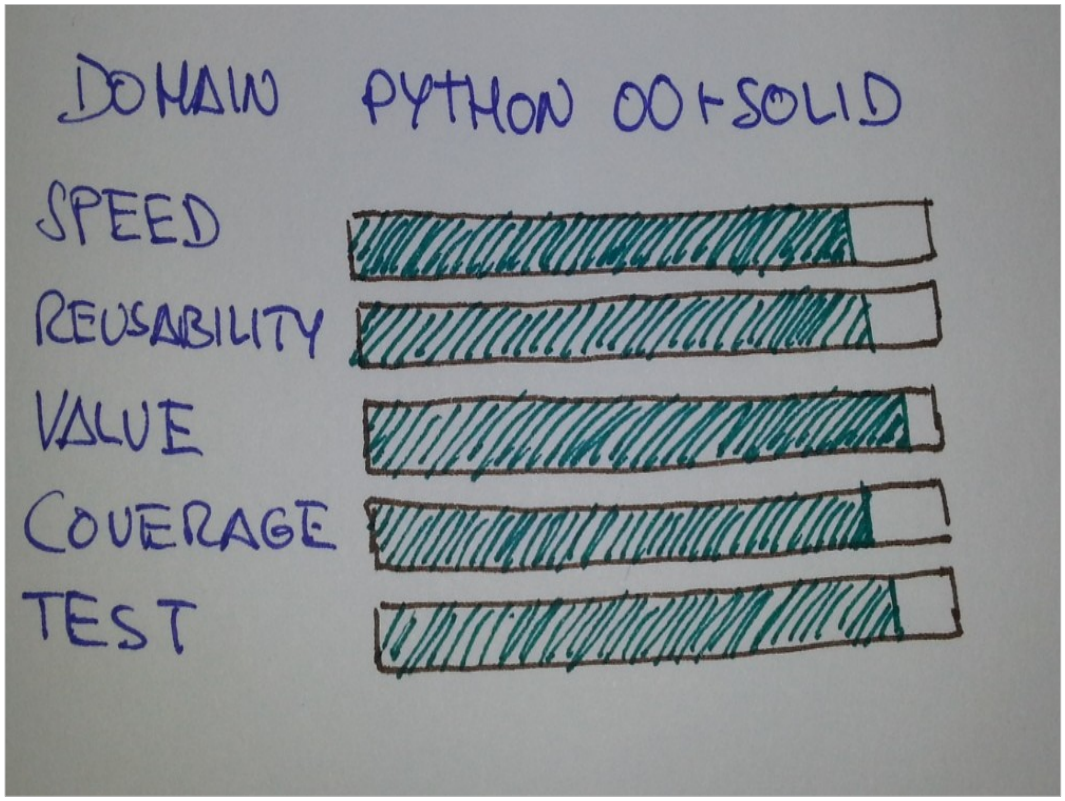
DDD Domain Driven Design

Clean Code / S.O.L.I.D

TDD Test Driven Development



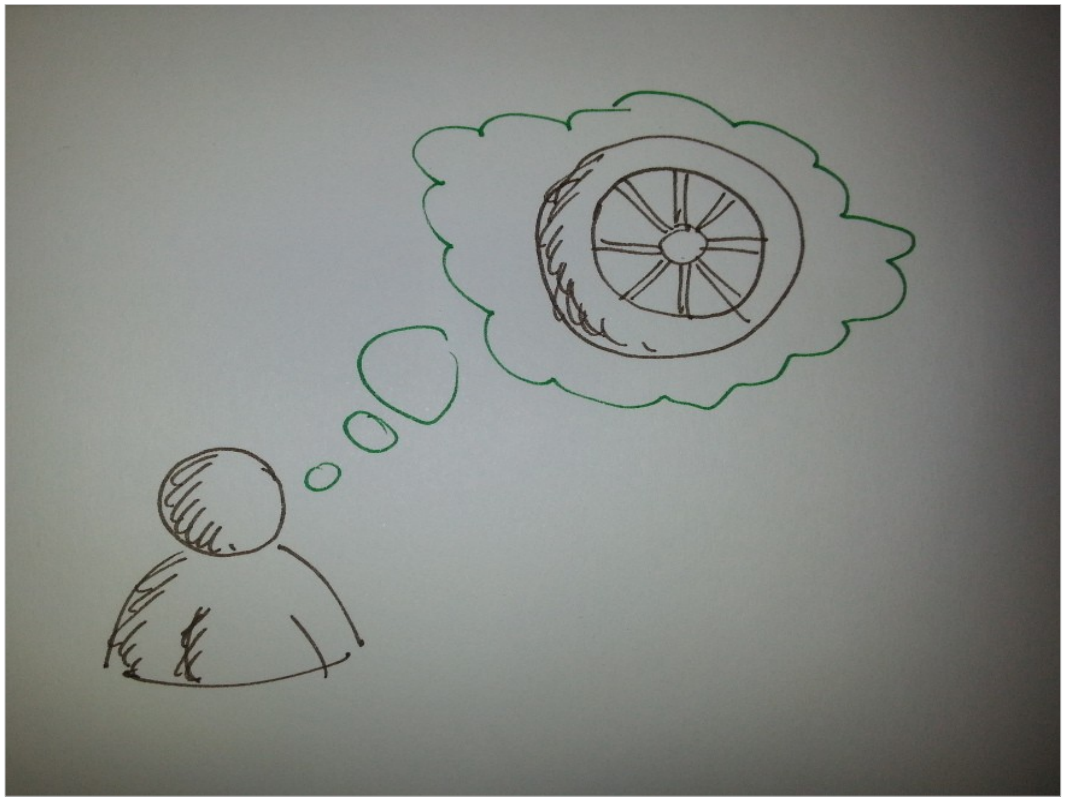
Ese código:
Es independiente de IO
(No net, No BD, No files, …)
Puede depender de abstracciones
(Cliente rest, Persistencia agregado, ….)



Velocidad desarrollo buena
Reusabilidad buena
Valor (desde el punto de vista del cliente) buena

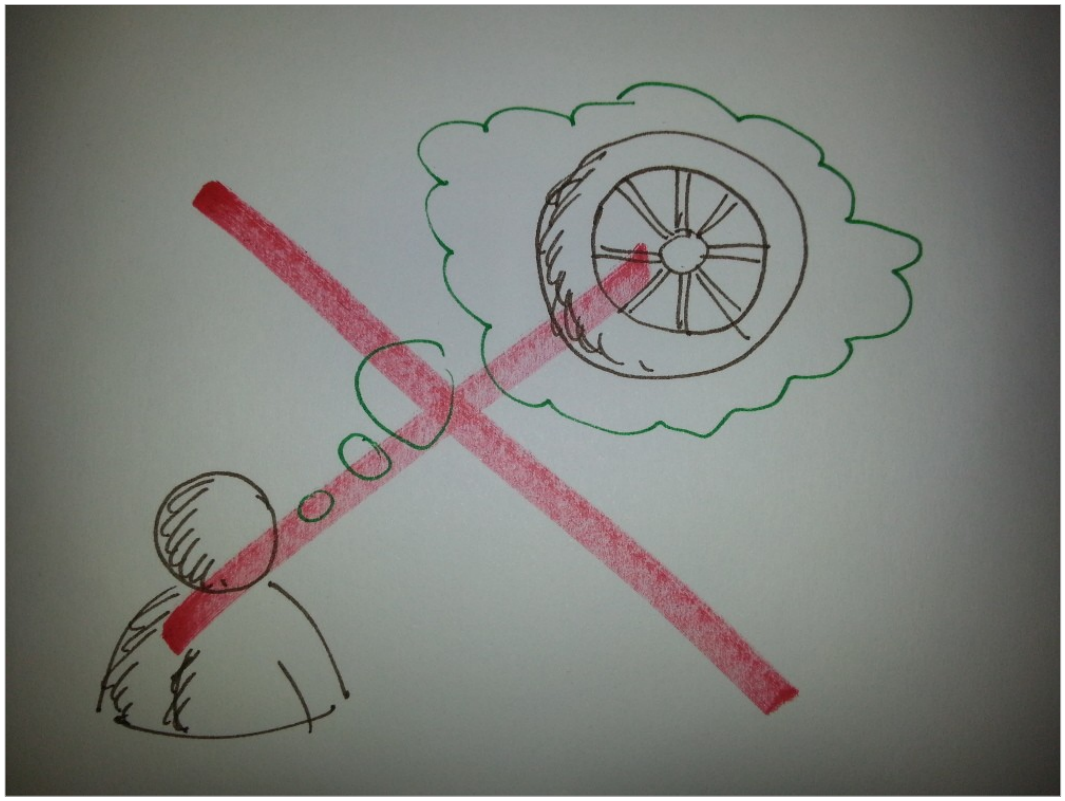
Esto es por lo que nos pagan

¿Cómo hacemos el resto?

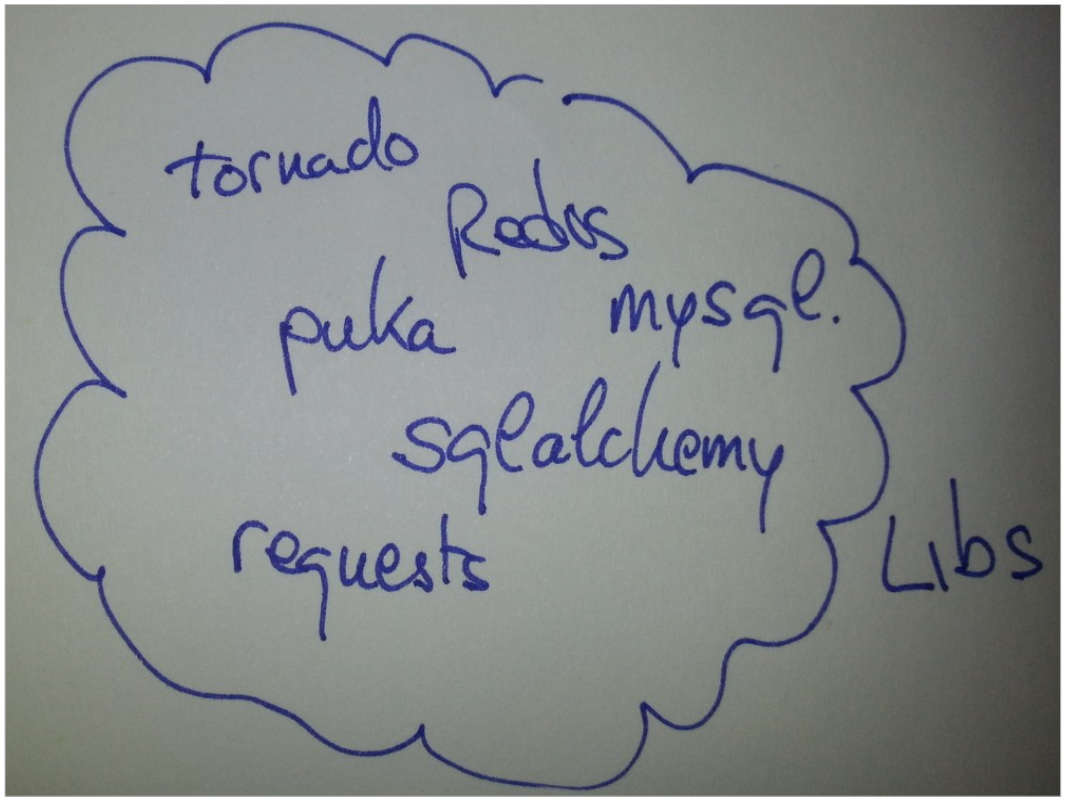


y el resto de código, el que se integra con el mundo exterior, el que permite comunicarse, el que permite persistir….
Para ese código…

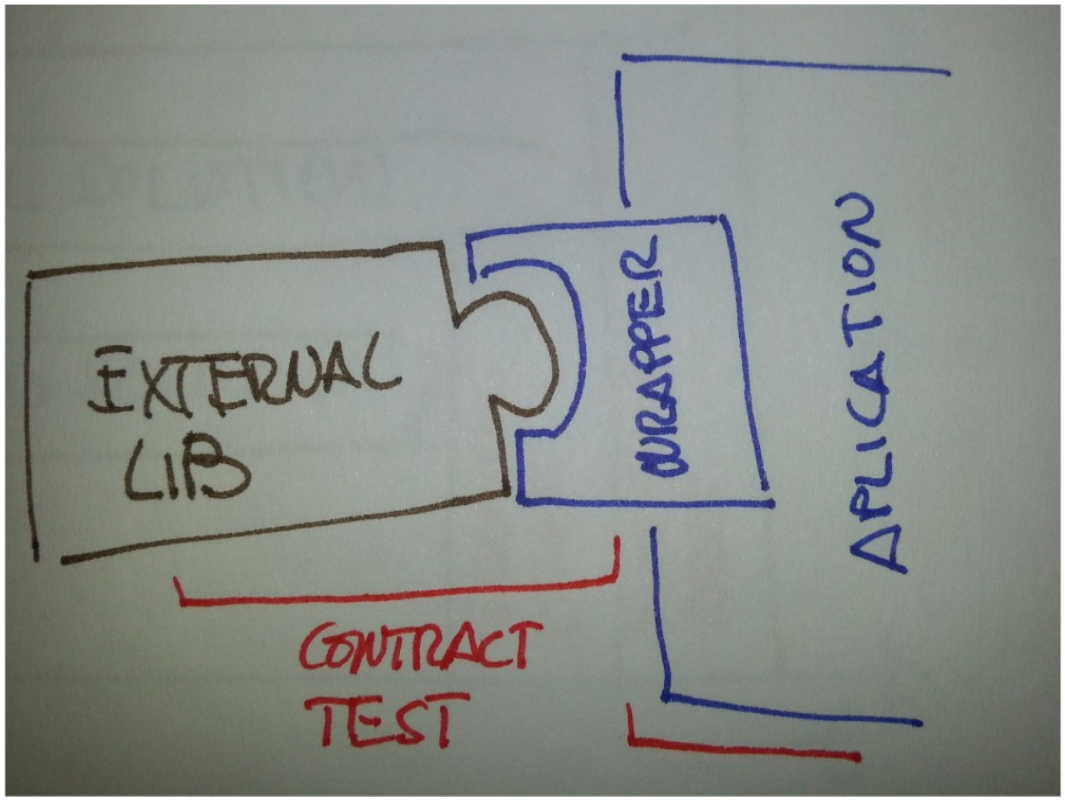
Reinventamos la rueda ????



Ni de coña



Usamos librerías…
Muchas



Cada librería la integramos
Creando una abstracción de la parte que vamos a usar (que será lo mínimo posible)
La parte de negocio usa esta abstracción (nunca la librería)
Creamos un test de contrato para la parte que usamos (verificamos que podemos enviar un mensaje, escribir una tabla o lo que sea)

Infraestructura y Entrega

Usamos las abstracciones sobre libs

Clean Code / S.O.L.I.D

TDD Test Driven Development

Conclusiones

- Los frameworks son una decisión muy importante
- Las decisiones tienen que ser conscientes

Nuestra forma de trabajo y arquitectura debe permitirnos:

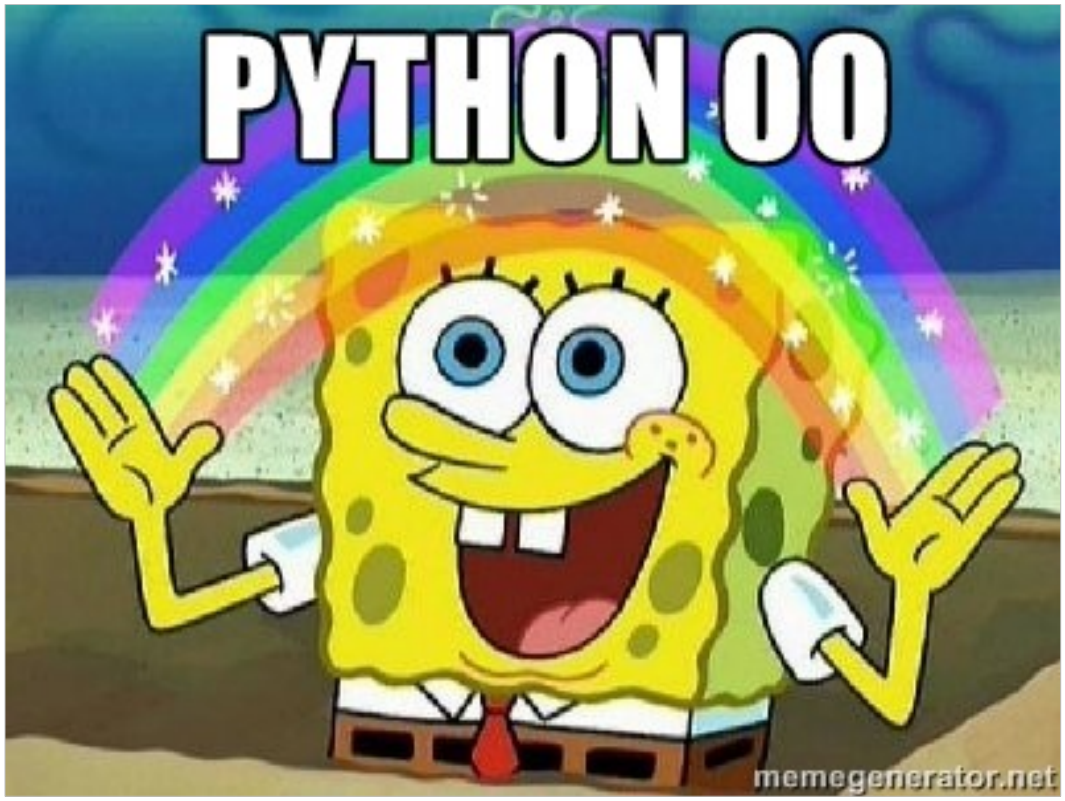
- Hacer un sistema que evolucione con las necesidades del cliente
- Postponer decisiones
- Aplicar distintas soluciones técnicas

Aplicar distintas soluciones técnicas (según se vayan necesitando, nunca antes)

Existen alternativas (Reales)

Que permiten centrarse en negocio
Que facilitan el desarrollo ágil
Que evolucionan a largo plazo
Que evolucionan a buena velocidad

Aplicar distintas soluciones técnicas (según se vayan necesitando, nunca antes)



Desarrollar en Python OO (Puro) es sencillo y divertido

Referencias

DDD

http://en.wikipedia.org/wiki/Behavior-driven_development

TDD

http://en.wikipedia.org/wiki/Test-driven_development

Hexagonal Architecture

http://alistair.cockburn.us/Hexagonal+architecture

Clean Architecture

http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html


PREGUNTAS?
memegenerator.net

Desksurfing