

Ingeniería de Aplicaciones Web

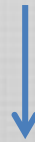
Diego C. Martínez

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

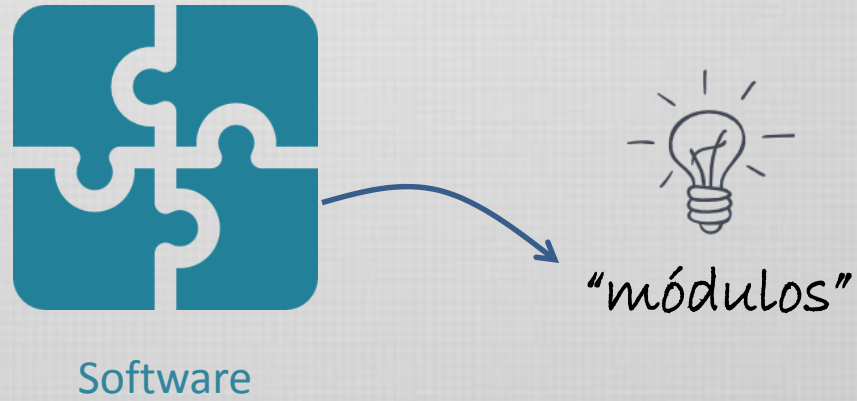
Modularidad



Software



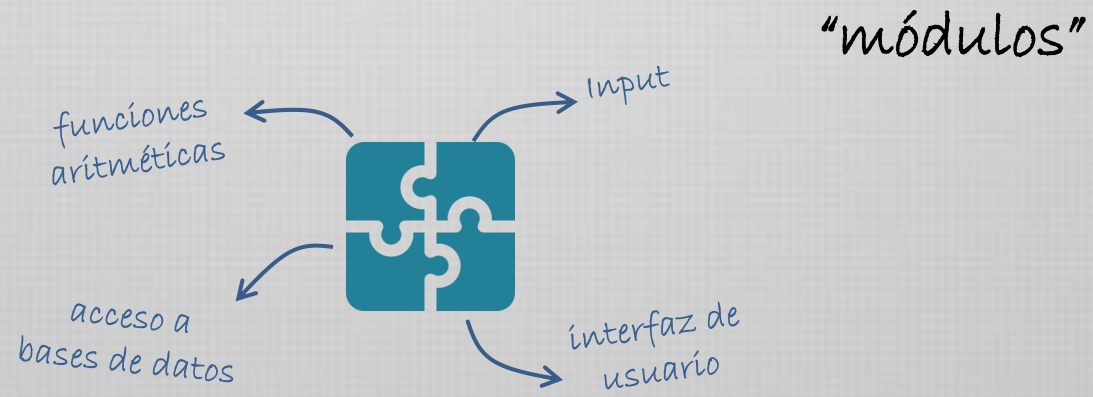
Modularidad



- Las partes se focalizan en un aspecto del sistema
- Las partes son intercambiables
- Las partes son independientes

Modularidad

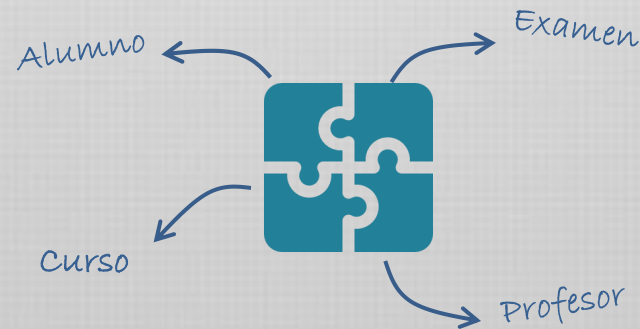
¿Cuál es el significado de un módulo?
¿Cuál es el criterio de división en módulos?



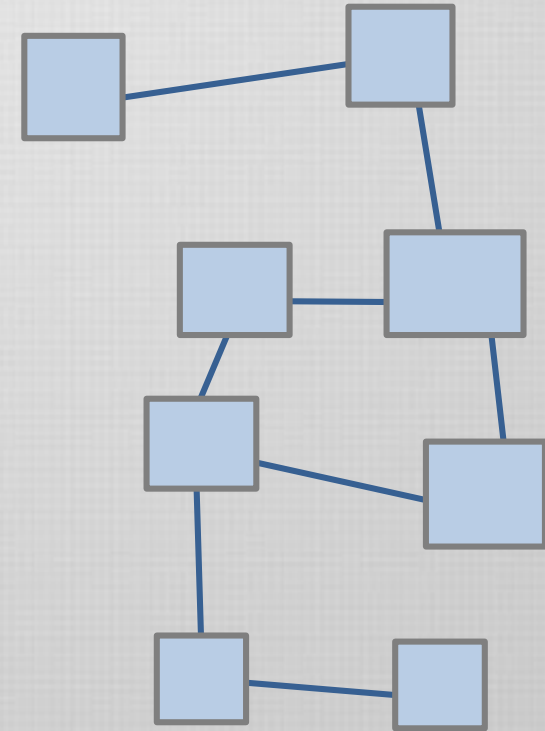
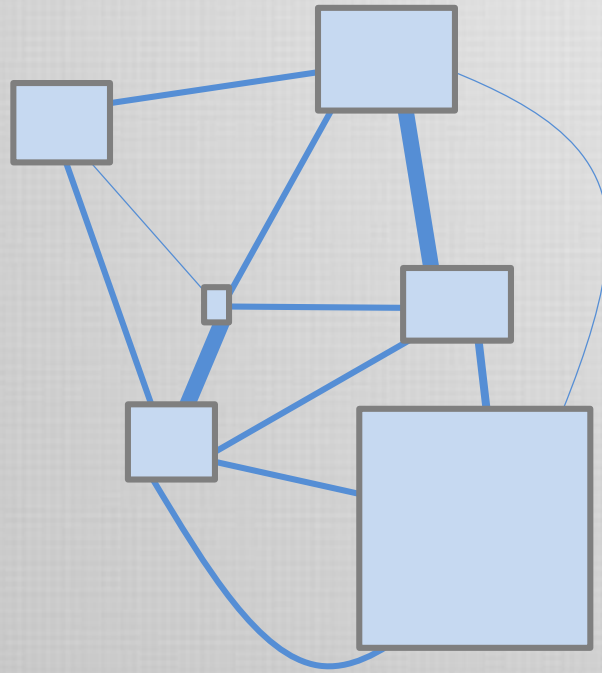
Modularidad

“módulos”

¿Cuál es el significado de un módulo?
¿Cuál es el criterio de división en módulos?



Diseño modular



Señales de un mal diseño modular

Subjetivamente, para algunos la regla para identificar un mal diseño es TNTWIWHD
“That’s not the way I would have done it”

Objetivamente

El software es **difícil de cambiar** pues afecta a muchas otras partes del sistema

Cuando se realiza un cambio, **partes inesperadas** del sistema fallan

Es **difícil reusar** las partes en otra aplicación porque están muy enredadas en la aplicación actual

Para procurar un buen diseño lo mejor es:

comprender los elementos del diseño

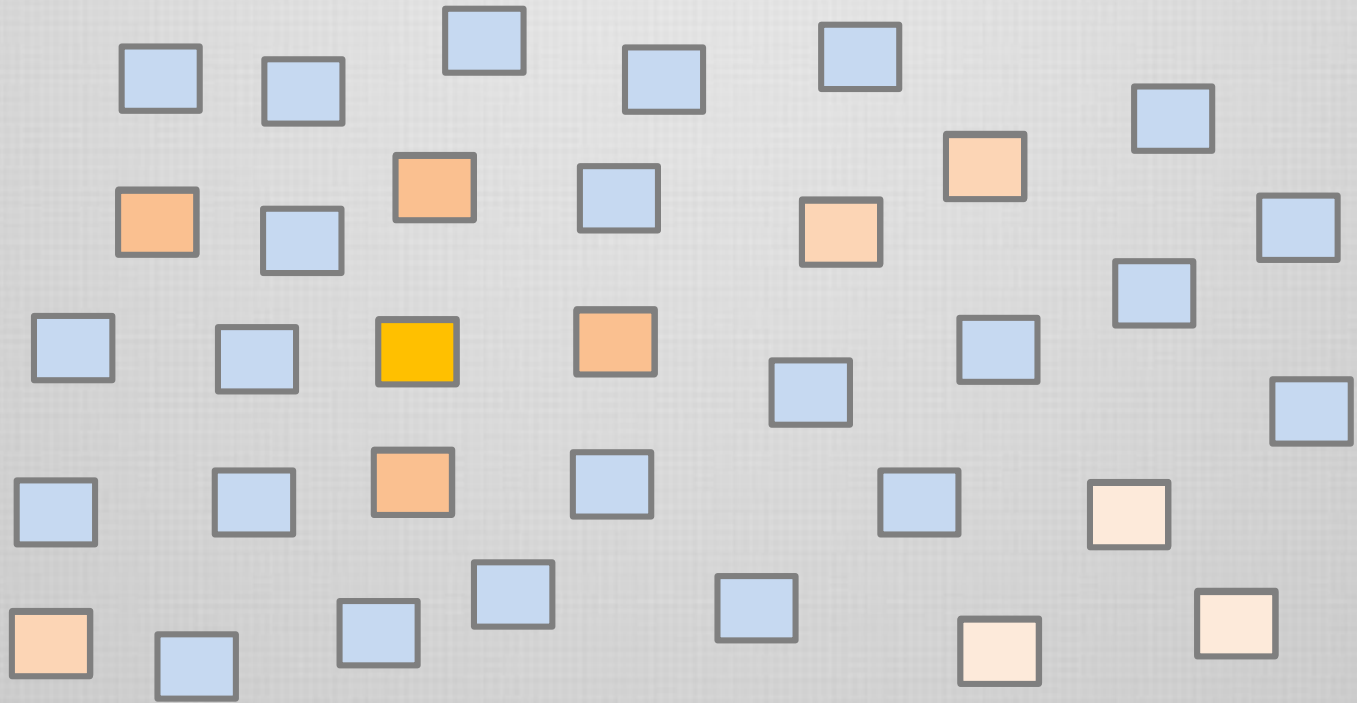
conocer buenos diseños

Principios de buen diseño modular



costo del cambio

Principios del buen diseño modular



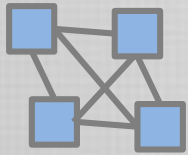
Un buen diseño ayuda a minimizar esfuerzos
en la **modificación** de un módulo

especificaciones

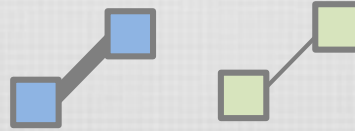
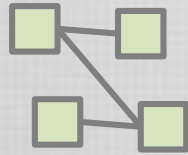
errores

actualizaciones

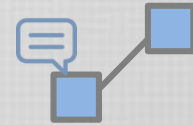
Algunas reglas en pos de un buen diseño modular



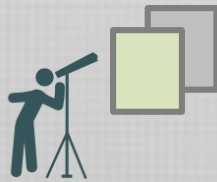
Pocas Interfaces



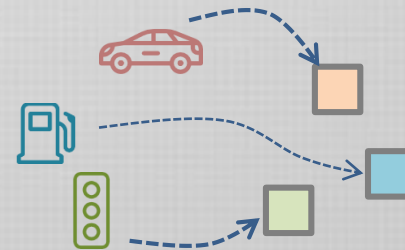
Interfaces Pequeñas



Interfaces explícitas



Ocultamiento de
Información

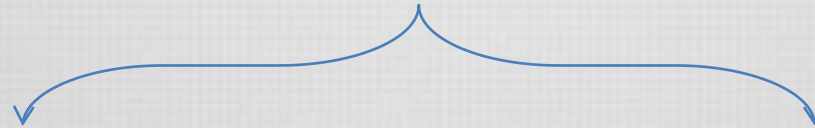


Mapeo Directo

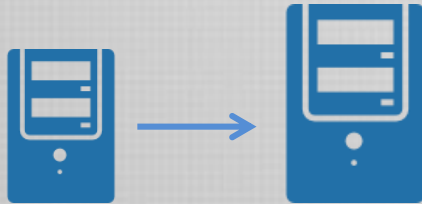
Escalabilidad

Un aspecto importante de la arquitectura

La habilidad de adaptar el sistema para permitir mayores cargas de trabajo



Escalabilidad Vertical



+ recursos para un nodo

*de poca complejidad
solución trivial*

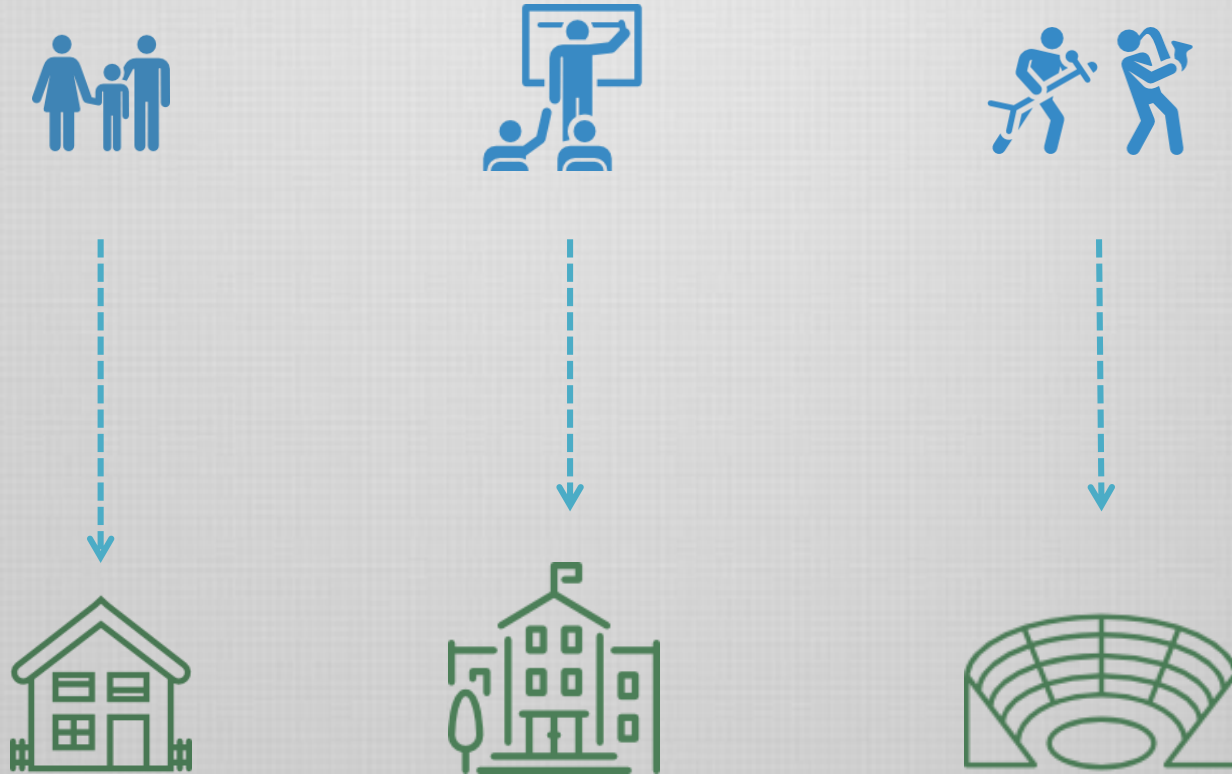
Escalabilidad Horizontal



+ nodos

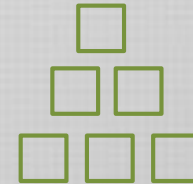
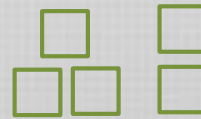
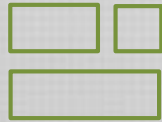
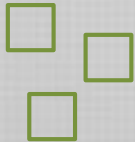
*requiere preparación
implementación no trivial*

Arquitectura



La arquitectura de un edificio es la solución estructurada que satisface los requerimientos iniciales

Arquitectura de Software



Arquitectura de Software



Ralph Johnson

“...architecture is the
important stuff (whatever
that is)”

The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both.

[Documenting Software Architectures: Views and Beyond](#)

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

[Software Architecture in Practice](#)

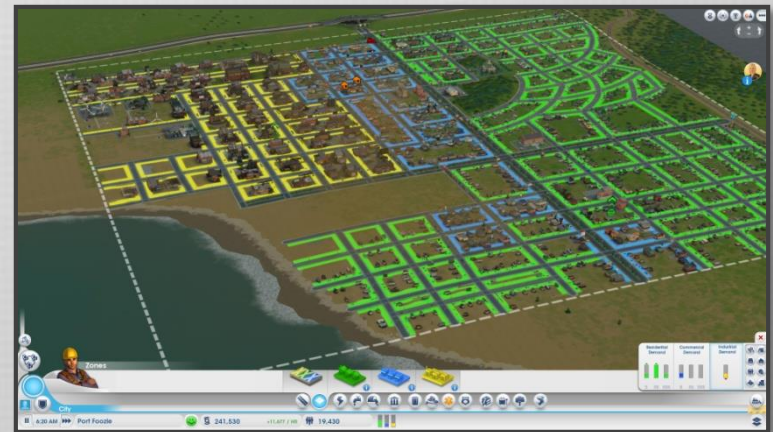
the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

[EEE Std 1471-2000](#)

Arquitecto



¿analogía adecuada?



El arquitecto de software es mas urbanista que arquitecto



Arquitecto

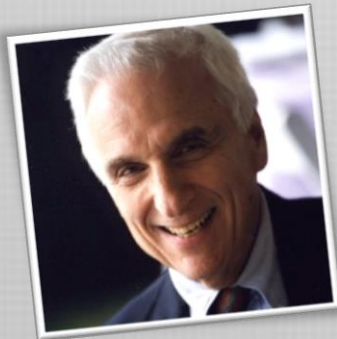
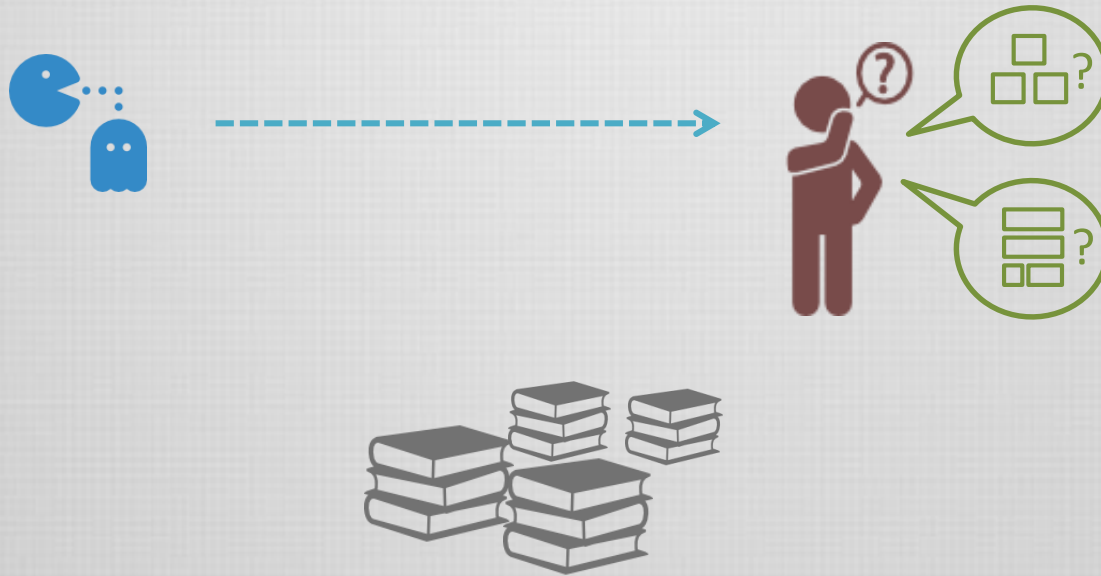


Delinea *zonas* con objetivos y responsabilidades
Se asegura que *el todo* cumpla con su propósito

Más preocupado por lo que ocurre *entre* las zonas,
que lo que ocurre *dentro* de las zonas

Debe planificar para el cambio

Arquitecturas de software

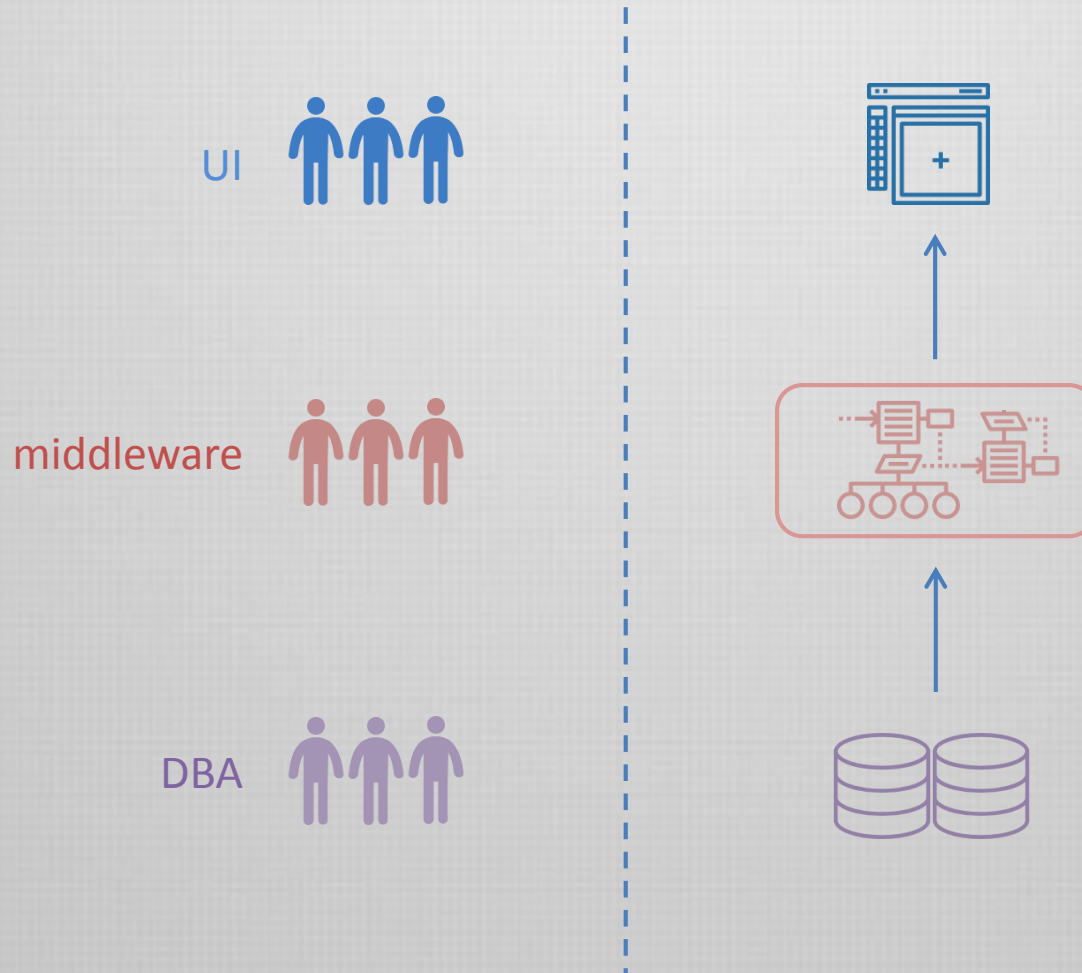


Melvyn Conway

“Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure” (1967).

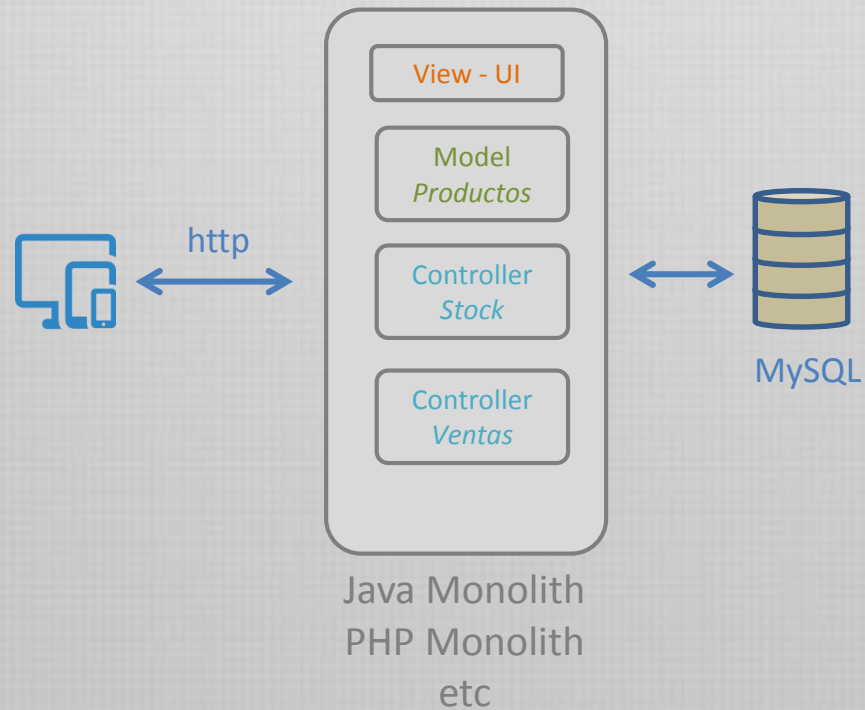
TIP: *Structure teams to look like your target architecture, and it will be easier to achieve it.*

Arquitecturas Monolíticas

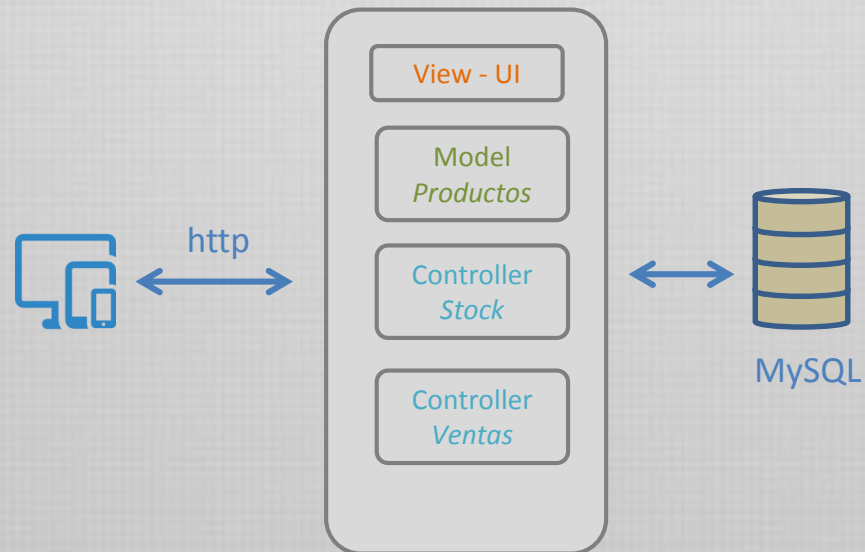


Arquitecturas Monolíticas

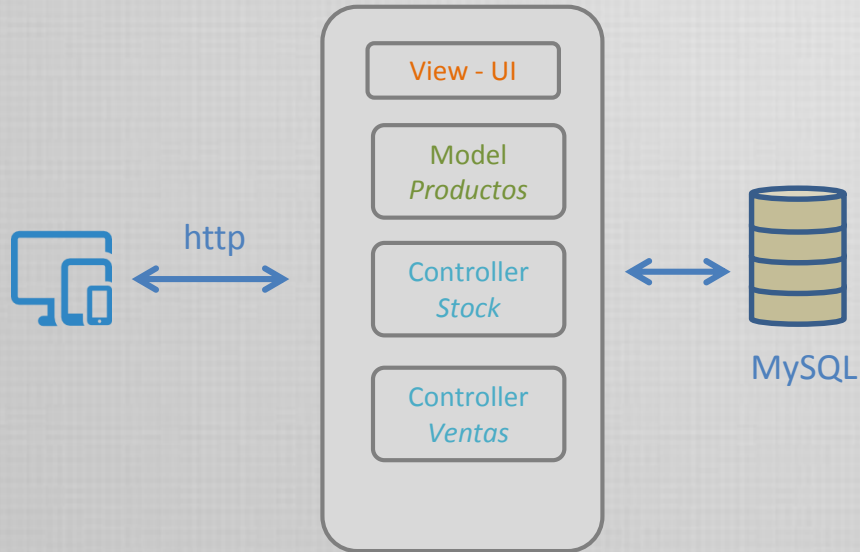
Shopping Cart



Arquitecturas Monolíticas



Arquitecturas Monolíticas



Proceso de desarrollo simple

Sustentado por IDEs, enfocados en el sistema completo



Testing de integración simple

Las pruebas se aplican a un sólo sistema en conjunto



Deployment simple

Copia en el servidor y configuraciones locales



Escalabilidad horizontal simple

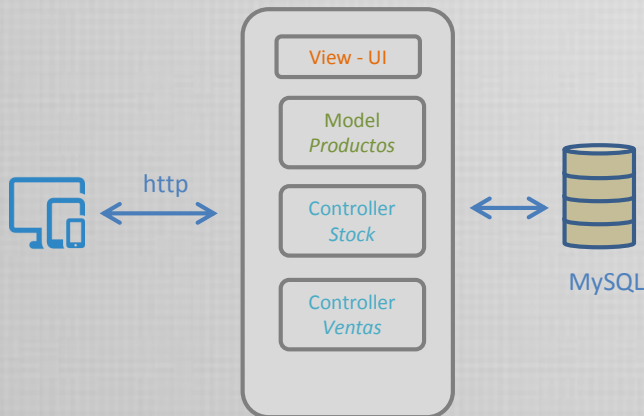
El sistema escala con múltiples instancias en el servidor

Arquitecturas Monolíticas



Dificultad en el desarrollo

*Para sistemas complejos requiere conocimiento completo
Si crece el proyecto, disminuye la productividad
Difícil análisis de cambios*



Dificultad en el equipo de desarrollo

Es difícil dividir en equipos independientes



Compromiso global con ciertas tecnologías

*Resulta difícil integrar tecnologías nuevas
Hay que apegarse al stack elegido*



Overloaded IDE

Se trabaja con IDEs complejos que administran todo el código

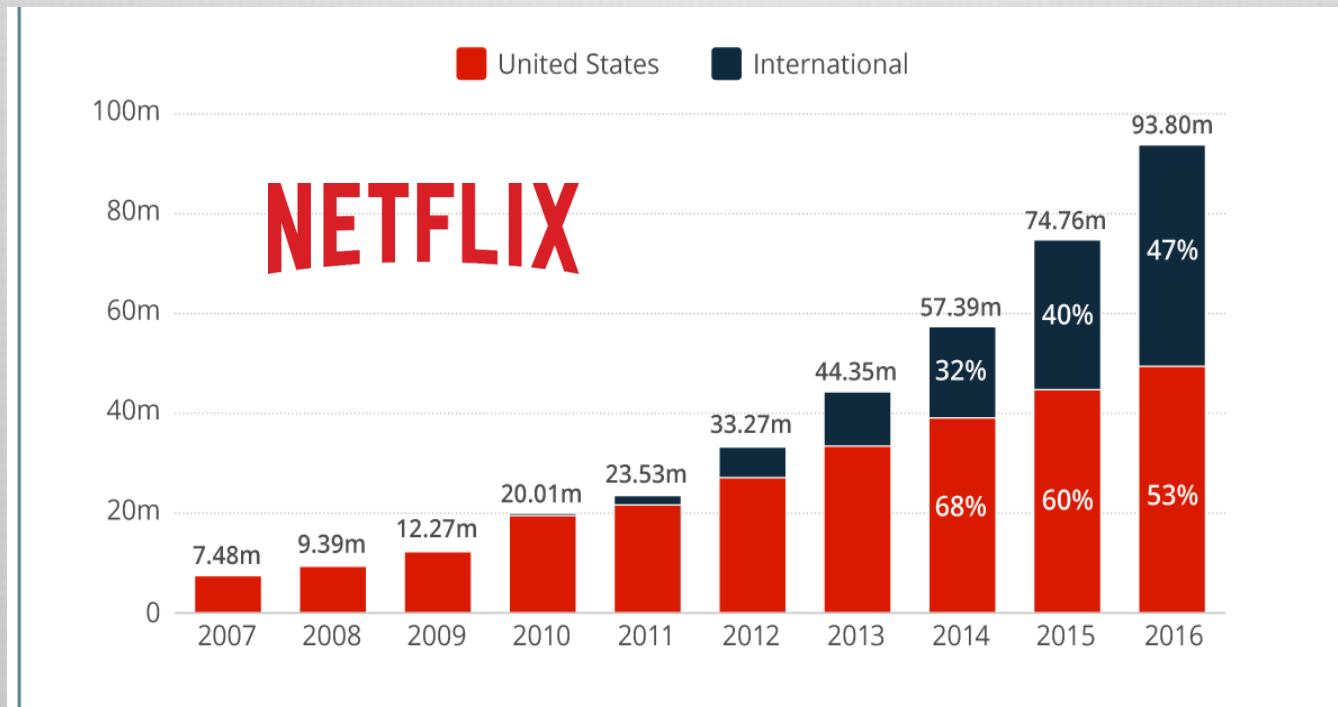


Difícil Escalabilidad

*Todo el sistema se ve afectado
No pueden escalarse partes individuales*

Escalabilidad en arquitecturas monolíticas

Para muchas aplicaciones, el factor de escalabilidad es esencial
Las arquitecturas monolíticas no son fácilmente escalables



Arquitecto



En estos contextos el arquitecto debe necesariamente ser **evolucionario**

Visión

Asegurarse de que existe una visión técnica global que cumpla con los requerimientos establecidos

Empatía

Comprender el impacto de sus decisiones en el cliente y en los colegas

Colaboración

No trabajar aisladamente

Adaptabilidad

Cambiar su visión técnica según evoluciona el cliente y sus colegas

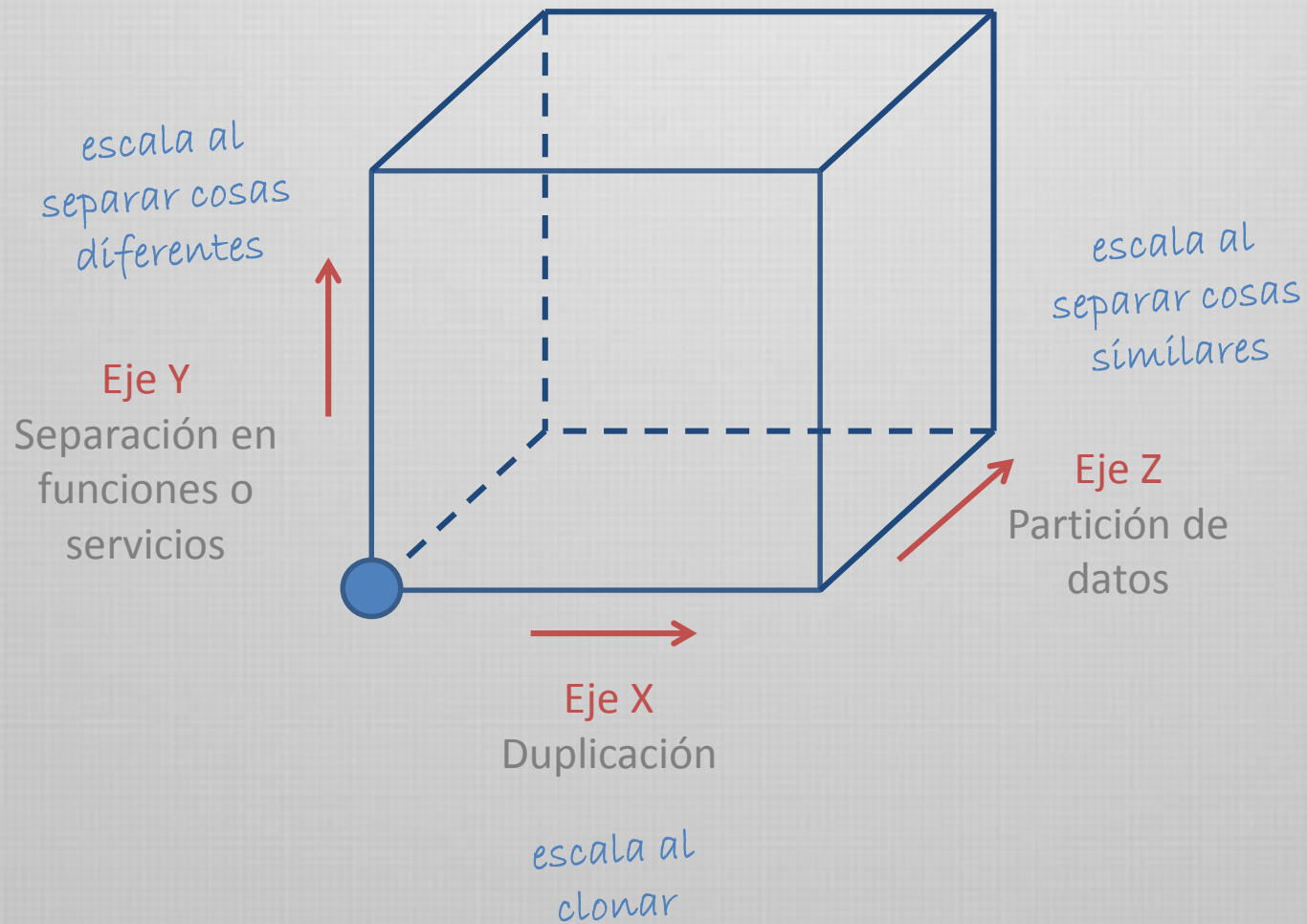
Autonomía

Encontrar balance entre autonomía y estandarización

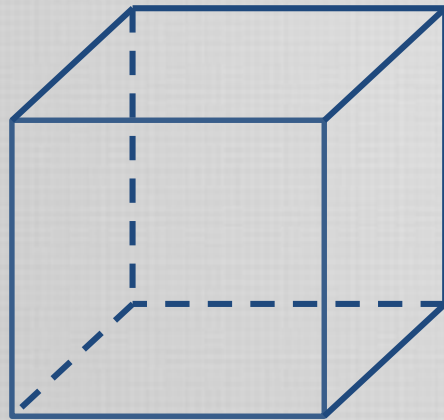
Governanza

Asegurarse que el sistema que se está implementando cumple con su visión

Cubo de escalabilidad

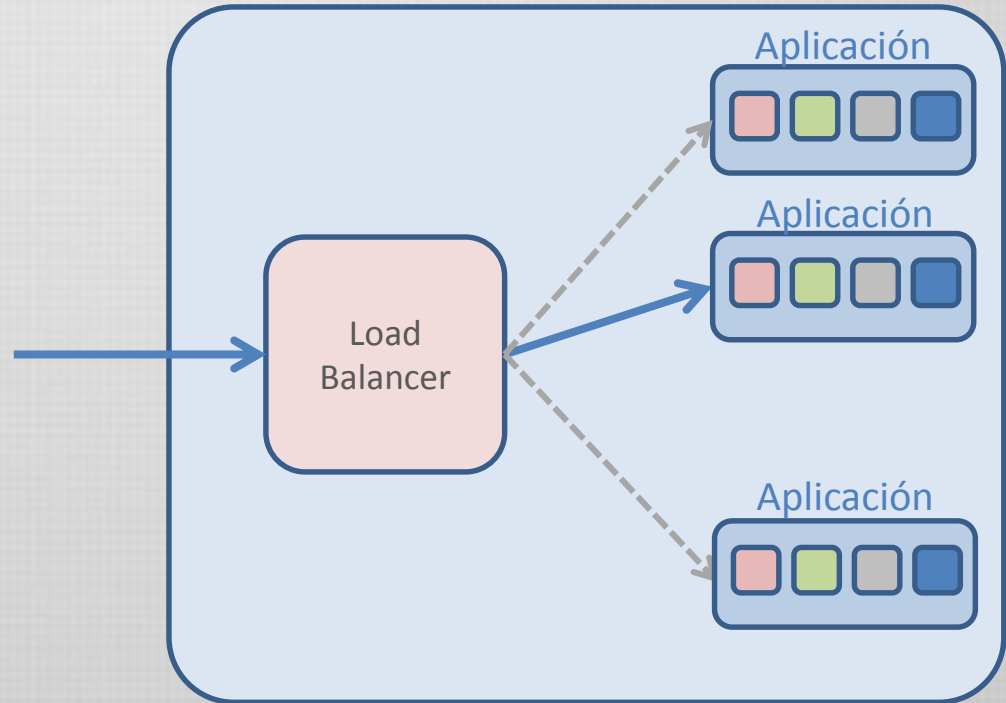


Cubo de escalabilidad



→
Eje X
Duplicación

escala al
clonar

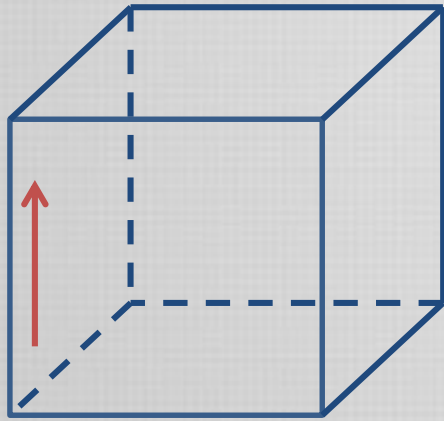


N sistemas replicados íntegramente
Cada uno de los sistemas se comporta igual al resto

Implementación simple
(**stateless** mas simple que **stateful**)

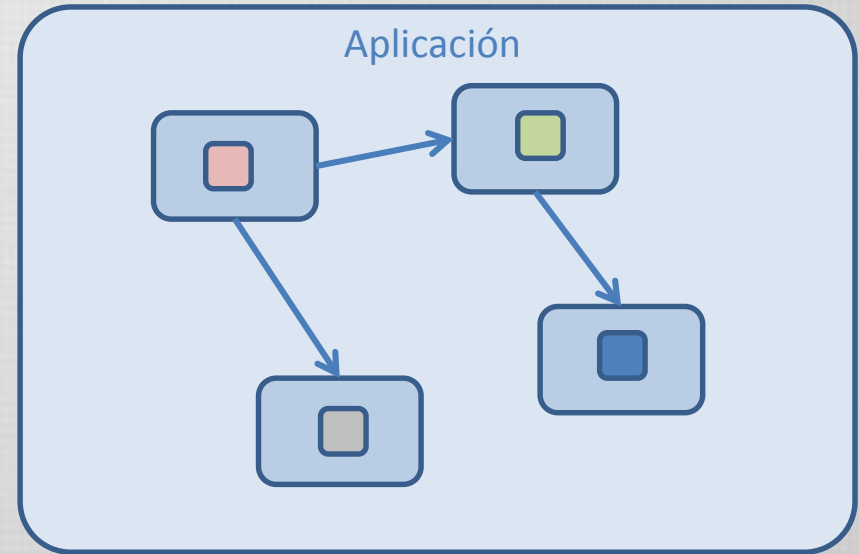
No escala bien si la aplicación crece mucho
en tamaño de código y datos

Cubo de escalabilidad



Eje Y
Separación en
funciones o
servicios

*escala al
separar cosas
diferentes*



Separación de las responsabilidades
dentro de la aplicación

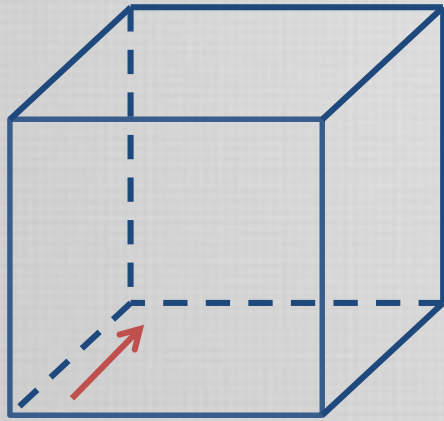
*Puede implementarse clonando y restringiendo
funcionalidad en los clones*

Ideal para el crecimiento del código y los datos

El equipo de desarrollo se divide en grupos
enfocados en los servicios individuales

Más costoso de implementar

Cubo de escalabilidad

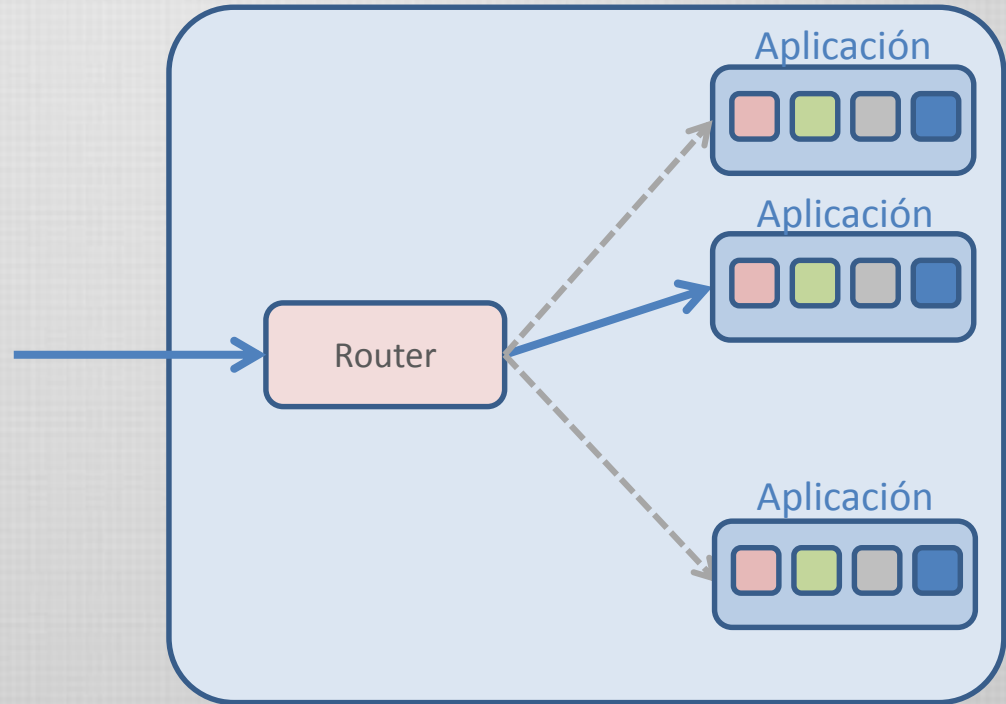


Eje Z

Partición de
datos

escala al
separar cosas
similares

Es el más costoso de implementar
Debe estudiarse el request, su
originador y la carga asociada al
trabajo puntual



Cada instancia es responsable de un
subconjunto de los datos

La división se hace en función de
la naturaleza transaccional

*e.g, clientes, proveedores, usuarios locales,
usuarios internacionales, etc*

Microservicios

Los **microservicios** son un **estilo arquitectónico** que organiza una aplicación como una **colección de servicios poco acoplados** que implementan lógica de negocios

“...the microservice architectural style is an approach to developing a **single application as a suite of small services**, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”



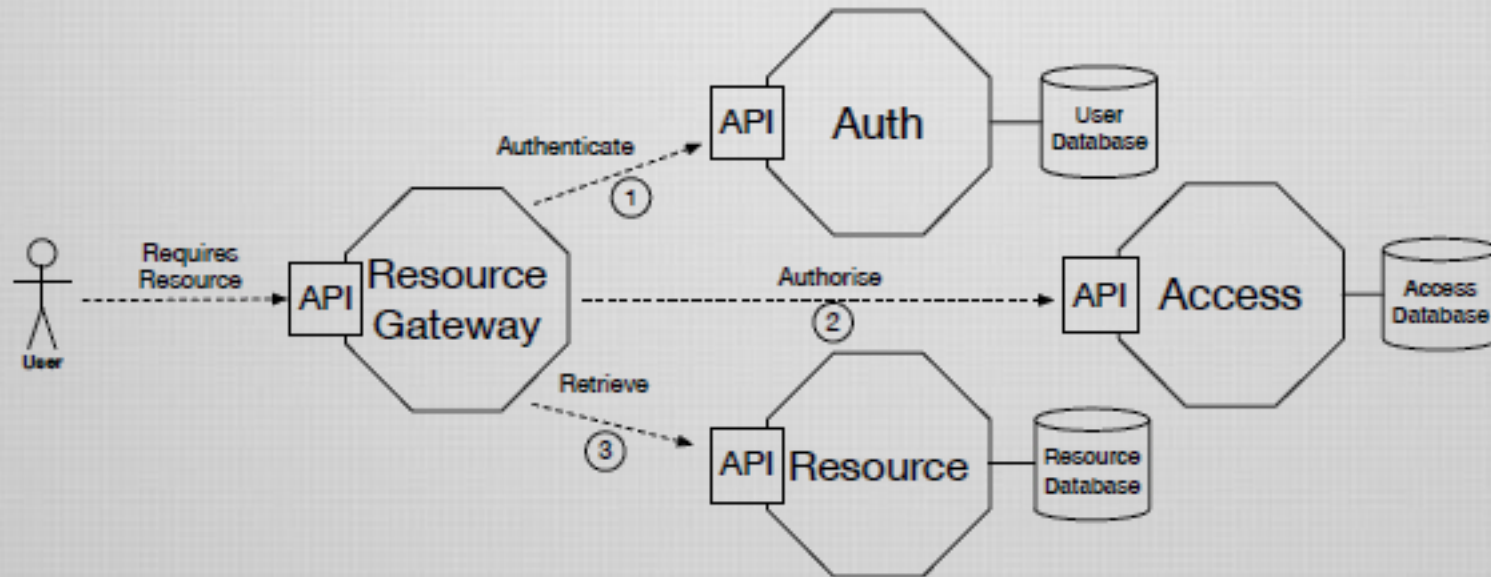
Martin Fowler



Adrian Cockcroft

“..fine grained SOA”

Microservicios



Microservices: yesterday, today, and tomorrow. Nicola Dragoni et al

Microservices vs SOA

SOA

Service Oriented Architecture

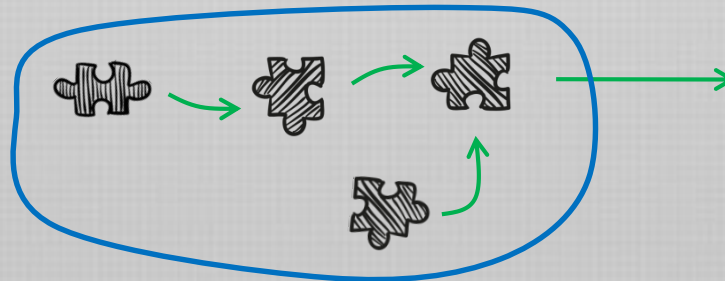


una forma de diseñar, implementar y ensamblar servicios
(para sustentar "business functions")

primeras SOAs en los 90's (DCOM, CORBA)

El software es particionado en función de **servicios**
poco acoplados
que ocultan su implementación

El software ofrece **funcionalidad** ensamblando servicios



Principios de Arquitecturas de Servicios



Don Box

“Four Tenets in Service Architecture”

Fronteras explícitas

Los servicios interactúan por mensajes a través de fronteras explícitas que protegen/ocultan las implementaciones internas

Los servicios son autónomos

El servicio reacciona ante un mensaje. No depende del contexto de otros servicios. Son independientes del sistema subyacente.

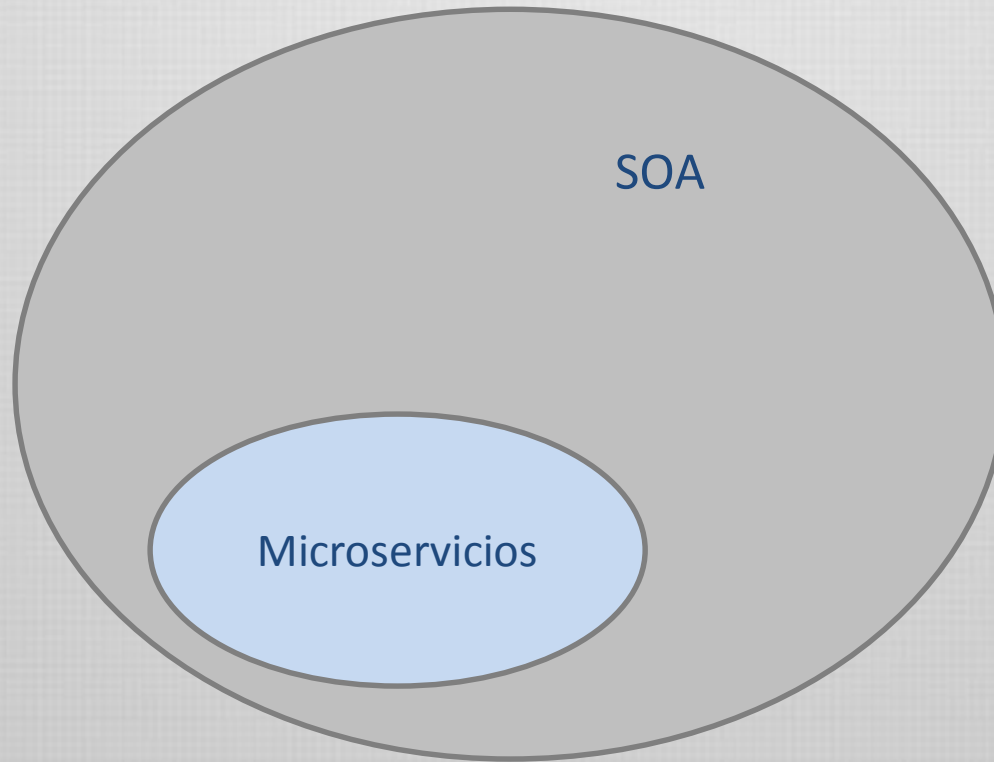
Los servicios comparten schema y contratos, no clases

*Agnosticismo de tecnologías de programación subyacente.
Sólo tecnologías como XML, WSDL*

La compatibilidad de servicios se basa en políticas.

*policy = descripción de capacidades y requerimientos de cada web service.
WSDL no siempre es suficiente.*

Microservices vs SOA

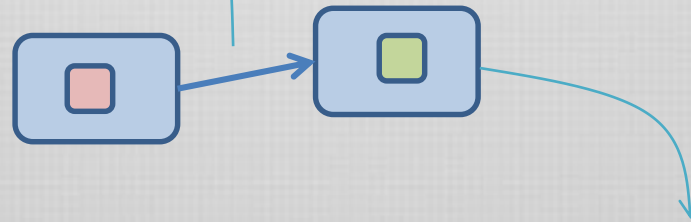


Microservicios

Dos aspectos son importantes al considerar un **buen servicio**

*Bajo Acoplamiento
(loose coupling)*

*Un cambio en un servicio
no implica un cambio en otro
Un servicio conoce lo menos posible de
los otros servicios con los que colabora*



*Los servicios relacionados
se mantienen juntos
Debe encontrarse el límite entre los
comportamientos similares
para delinear los servicios*

*Alta Cohesión
(high cohesion)*

Microservicios - características

Más allá de las definiciones, los **microservicios** se identifican mejor según ciertas **características** que deben poseer



Microservicios - características

Microservicios



- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

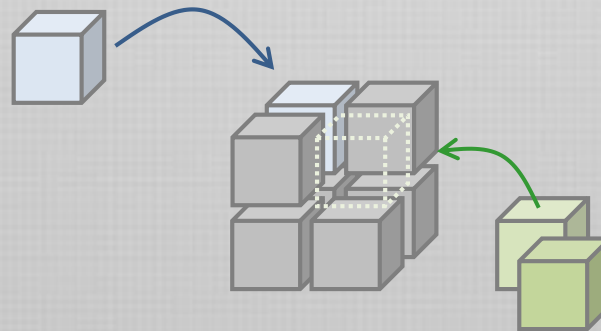
Microservicios - características

Microservicios

Componentization via Services

- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Los **componentes** son unidades de software que son **reemplazables** y **actualizables** con independencia del resto



Microservicios - características

Microservicios

Componentization via Services

- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Los **componentes** son unidades de software que son **reemplazables** y **actualizables** con independencia del resto

En una arquitectura microservicios los **componentes** son **servicios**



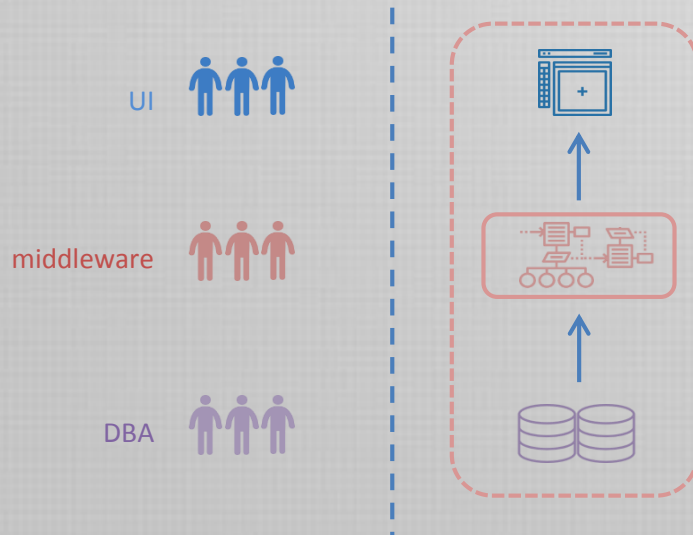
- Desplegables de manera independiente
- Corren en procesos separados
- Poseen una interfaz de componente clara (RPC, HTTP, etc)

Microservicios - características

Microservicios

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Los servicios se basan en las funcionalidades de la organización



Microservicios - características

Microservicios



- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Usualmente se aplica un **modelo de proyecto** en el desarrollo



un equipo que construye y entrega el producto
el equipo se disuelve luego de la entrega
el mantenimiento es problema de otro equipo

Los procursores de los microservicios proponen otro modelo:
el equipo se adueña del producto en todo su ciclo de vida

amazon

"you build it, you run it"

Microservicios - características

Microservicios

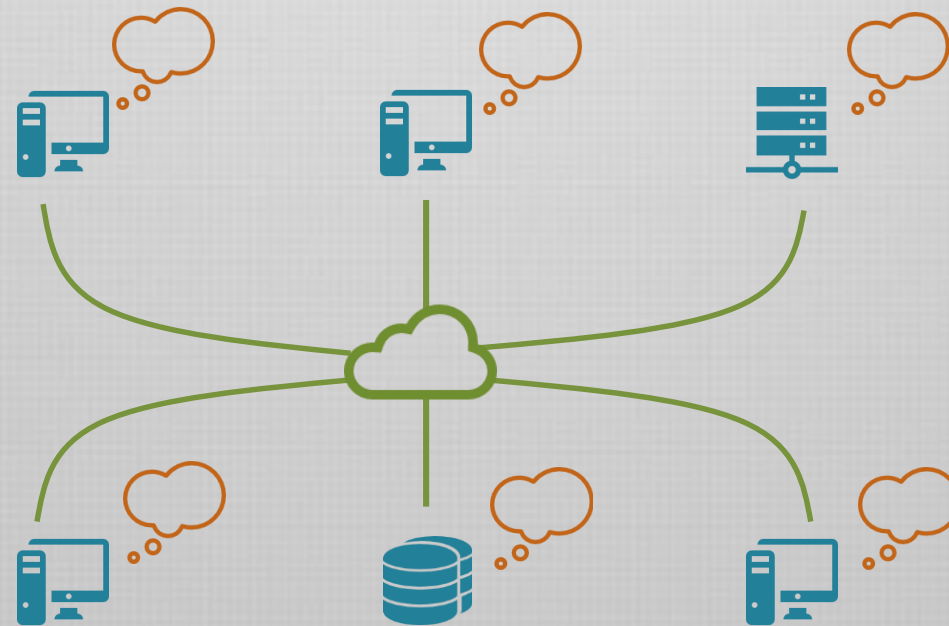
- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design



Microservicios - características

Microservicios

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design



Microservicios - características

Microservicios

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance**
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

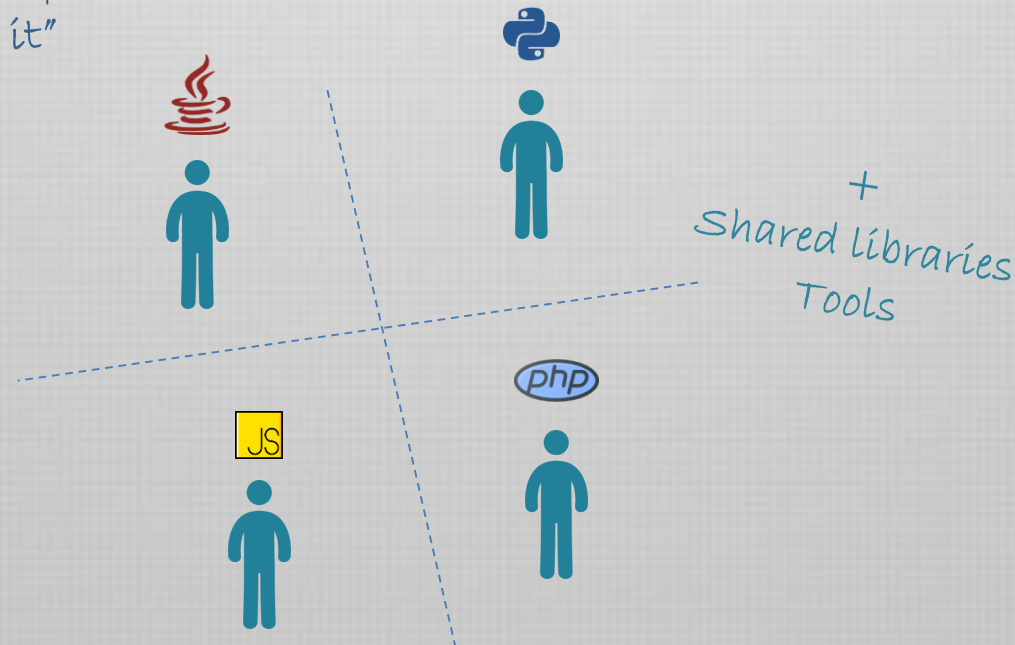


Microservicios - características

Microservicios

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance**
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Consistente con el principio
"you build it, you run it"

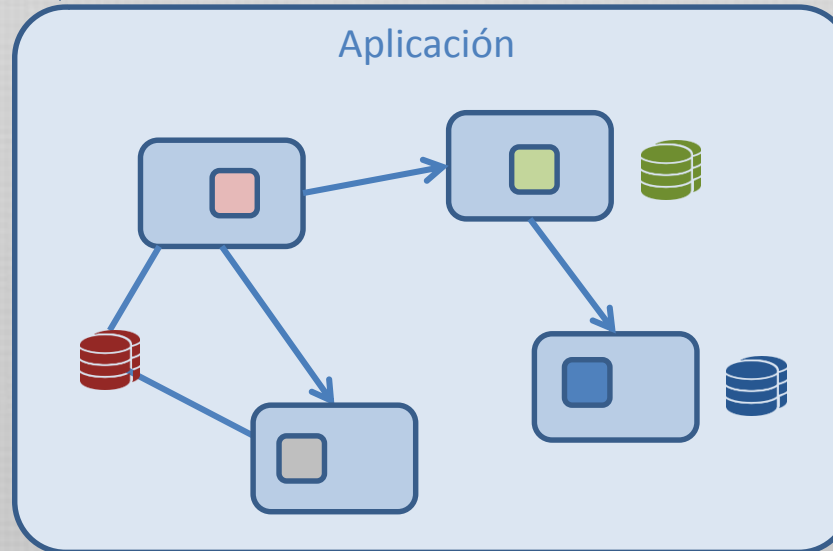


Microservicios - características

Microservicios

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

el modelo de
datos puede
diferir
eg, "cliente"



La BD puede variar
"polyglot
persistence"

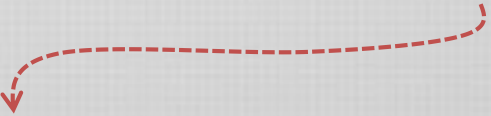
Microservicios - características

Microservicios



- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Se hace uso de técnicas automatizadas para Continuous Delivery



Disciplina de producción que permite la entrega del producto en cualquier momento

requiere un ciclo de vida acorde

requiere un equipo de trabajo cohesionado

requiere automatización en todas las partes posibles del proceso de entrega

Microservicios - características

Microservicios



- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

La evolución de Cloud facilita la infraestructura automatizada
(AWS por ejemplo)

Automated Tests
Automated Deployment
Automated Tools



Netflix OSS (<http://netflix.github.io/>)
Dropwizard (<http://www.dropwizard.io/>)

Microservicios - características

Microservicios



- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Alta tolerancia a fallas
(eg, *SimianArmy de Netflix*)

Monitoreo en tiempo real de los microservicios
...complementando *Event Collaboration Pattern*

Microservicios - características

Microservicios

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Al particionar el sistema en componentes independientes, se facilita el **cambio** y la **evolución** del sistema en general

Microservicios es una forma de arquitectura evolutiva

