

Vimos varias, pero en
el curso usamos est.

ESTRUCTURAS

- ¿Qué es la arquitectura del software?

Conjunto de estructuras necesarias para razonar sobre el sistema, que consta de elementos de software, relaciones entre ellos y propiedades de ambos.

Conjunto de estructuras.



Las estructuras tienen elementos.

Tipos de
estructuras

Modules

Components & Connectors

Allocation.

Diseñamos estructuras.

La arquitectura es abstracta. → La visualizamos con vistas.

Estructuras → entender
para el sistema

↓
Tener elementos de software → que se relacionan.

(Estructuras abstractas de alto nivel.)
x fundamentales

(Decisiones de diseño)

• Atributos de calidad (QA - Quality Attributes)

["Propiedad mensurable de un sistema."]

↓ para saber csmo cumplimos con las necesidades de los stakeholders

Se favorecen o desfavorecen con tácticas

• Tácticas

↳ "Decisiones de diseño y restricciones, que afectan a los atributos"

Se eligen qué atributos priorizar

} En el design space.

↳ requerimientos del sistema

Estos fueron elegidos según → objetivos de negocio.

• Tácticas

→ "Decisiones que promueven o desfavorecen algún atributo de calidad"

1B

Más

- Todos los atributos

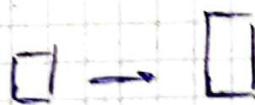
• Scalability (escalabilidad)

"Capacidad del sistema de incorporar recursos adicionales de manera efectiva"

Habilidades de soportar un gran número de componentes.

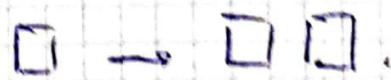
Escalabilidad vertical

Mejorar la capacidad de las unidades preexistentes



Escalabilidad horizontal

Crear más instancias del mismo tamaño (más unidades lógicas).



Elasticidad — Capacidad del sistema de crecer y decrecer según la demanda.

(estilos con)
MicroSERVICIOS, stateless, desequilibramiento, sharding

o Security (seguridad)

i Capacidad de hacer seguro
el sistema?

Resistir ataques ^{acceso} externos.

o Usability (usabilidad)

"Facilidad que tiene el usuario de realizar la tarea deseada,

y lo ayuda que ofrece el software al usuario".

Tácticos

↳ Respaldar la iniciativa del usuario.

Cancel, Undo, Pause, Resume

↳ Respaldar la iniciativa del sistema.

Autocorrector

Redundancy + Spare
Removal from service
Process monitor

• Availability (disponibilidad)

"Habilidad para reparar fallas para minimizar el tiempo fuera de servicio"

$$\begin{matrix} \text{Fault} + \text{Fault} + \\ \text{Fault} \end{matrix} = \text{Failure} \\ (\text{Faults}) \qquad \qquad \qquad (\text{Failure})$$

"Probabilidad que esté operativo cuando se lo precisa"

↓ Omissiones
Crashses
Responses incorrectly
(Mal) timing

• Performance (rendimiento)

"Capacidad para reaccionar a eventos en cierto tiempo"

(Durante el desarrollo vs Ya con la app. en uso.)

- Network performance

• Throughput (caudal)

• Capacity
(máximo throughput posible EN EL SISTEMA)

• Bandwidth

(máximo throughput EN UN CANAL)

- User perceived performance
 - Completion time
 - Latency_{host} (tiempo hasta la primera indicación de respuesta).
 - Responsiveness (tiempo que toma el sistema en aceptar una request).
- Efficiency → "Performance por recurso consumido"
 - Es posible que sea menos eficiente si tiene más carga (Load sensitivity).

MONITOREAR (VER)

- Visibility:

"Monitorear y medir entre otros y los componentes"

Técnicas: Firewall, cache

- Testability:

"Qué tanto puede ser probado un componente"

Mock.

- Portability:

"Capacidad del software para ser usado en distintos ambientes".

- Interoperability:

"Capacidad para trabajar con otros sistemas".

- Usability:

"Qué tan fácil de usar es para los usuarios"

- Manageability:

"Qué tan fácil de gestionar es para los administradores"

Monitors, debuggers.

- Reliability:

"Qué tan sensible es a fallos"

- Security:

"Capacidad para resistir ataques maliciosos"

Encryptado, resistencia → Dos (attack-limiting)

- Simplicity:

"Facilidad de comprender la arquitectura"

Componentes más complejos

- Modifiability:

"Qué tan costoso es hacerle cambios al sistema, a lo largo del tiempo"

Evolve, Extend, Customiza | Configure, Reuse.

(por usuario) | (mod. post...) | (reutilizar sus l...

2

{ No existen]
Silver Bullets ! }

UNo mismo
soluciona no
sirve para
todo.

- ¿Qué son los estilos arquitectónicos?

Conjunto de arquitecturas que tienen características comunes.

[Conjunto de restricciones que limita el rol de los elementos arquitectónicos y las relaciones entre estos elementos.]

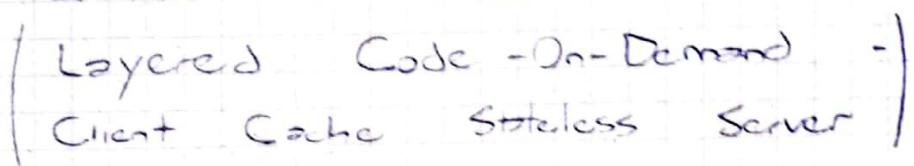
Al definir qué hacen sus elementos, restringe sus características, roles y relaciones. Les da un rol específico.

CONJUNTO DE DECISIONES

ARQUETIPO DE ELEMENTOS Y SUS RELACIONES.

• Arquitectura Web.

La web utiliza $\frac{1}{m^2}$ → estilo arquitectónico:



Internet ≠ Web { Web es una aplicación que corre sobre Internet }

Conjunto de computadoras interconectadas.

Documentos hiperTexto interrelacionados accedidos a través de internet.

REST → es el estilo arquitectónico de la web.

CSS + U + Layers + COD

Client Cache + Uniform Interface + Layers + Code On Demand } REST

Stateless Server

Estado

Application state: Donde está el cliente (en la web) lo contiene el cliente (mi browser sabe en qué URL estoy).

Resource state: El estado de un recurso está guardado en el servidor.
Se modifica con mensajes HTTP?

• Estilos arquitectónicos

2.

- Remote session . "El estado de la app. se mantiene por el server"

+ Simplicidad / - Escalabilidad - Visibilidad.

- Remote job access . "Los grandes datos se procesan en el servidor".

+ Eficiencia + Visibilidad / - Escalabilidad - Confidencialidad.

- Mobile code : "Movemos dónde se ejecuta el código, en vez de mover los datos".

- Virtual machine: "Se crea un ambiente controlado".

+ Portabilidad + Extensibilidad / - Visibilidad.

- Remote evaluation (VM+CS).

"Se envía qué hay que hacer a otro lado (el server)".

+ Extensibilidad + Eficiencia / - Visibilidad - Escalabilidad
- Confidencialidad.

- Code-on-demand.

"Se envía el código al cliente".

+ Configurabilidad + Extensibilidad + Escalabilidad
+ Performance / - Visibilidad.

- Layered Code-on-demand Client Cache Stateless Server.

"Se mueve un agente entre máquinas (no solo su código)."

- Mobile agent

• Estilos arquitectónicos

3

○ Event based integration (EBI)

"En vez de llamar a un componente en particular, se publica un **EVENTO** y los componentes adecuados reaccionan".

+ Extensibilidad + Reusabilidad + Evolucionabilidad + Eficiencia
- Escalabilidad - Entendimiento (imprevisible).

○ Distributed objects

○ Brokered distributed objects.

○ Data centered architectures.

← Database (simplemente donde se guardan datos) (Depende del input stream).

→ Blackboard "El estado se publica en el blackboard, y todo se hace en función de quién esto publicó ahí".

○ Pipe & Filter

"Una cadena de elementos recibe info, la procesa y la pasa al siguiente".

Uniform → Todos los Filters deben tener la misma interfaz

○ Batch sequential

"Se envía una porción de la info al próximo componente antes de terminar en el propio".

• Web

"Un espacio de información compartido donde las personas y las máquinas pueden comunicarse"

Almacenar info / Referenciar info que tiene otro.

Ente mundial / Sistemas heterogéneos.

Atributos de calidad

Baja = bajas barreras de entrada (Usability)

Extensible

Availability (resistente a condiciones adversas)

Hypermedio → Modo de hypertext

Instrucciones incluidas
sobre como manejar y
presentar la información

Documentos que
referencian a
otros (hyperlinks).

HATEOAS → Hypermedia as the engine
of application state

El cliente solo sabe como modificar
su application state o agregar o
eliminar opciones que recibió por
Hypermedio.

Rest API → "No prior knowledge
beyond the initial URI".

• Sistemas Distribuidos de Gran Tamaño.

Sistemas legacy
" larga vida"
Información redundante
Importante

Sistemas heterogéneos
" complejos"
Muchos owners
Cuellos de botella

IT es elemental para una empresa. Las necesidades (y funcionalidad) crecen, punto → los devueltos técnicos.
~~Arquitecturas de lados~~
Los SDGT son necesarios.

soluciⁿ:

• Service Oriented Architecture

Donde cada capa
del negocio

→ Es un componente de
software llamado
"Servicio".

PARADIGMA
" Service-oriented architecture (SOA) is a paradigm for the realization and maintenance of business processes that span large distributed systems."

SOA → Paradigma para → realización
y mantenimiento

↓
de procesos → en sistemas
de negocio distribuidos
de gran tamaño.

SOA, al ser
un paradigma
es abstracto.

↓
NO es un
software
en particular

Elementos:

- Servicios
- ESB (Enterprise Service BUS)
- Políticas y procesos
- Gestión.

SOA se diferencia de microservicios ya que SOA ~~estándar~~ se usa para SDGT y negocios. Microservicios es para software en general.

NEGOCIO

$$1 \text{ Servicio} = 1 \text{ funcionalidad de negocio.}$$

Se consume a través de una INTERFAZ → Accesible a la gente del negocio (gente no-IT).

INTERFACE ! ORIENTED

Interfaces BUSINESS ! DRIVEN

Enterprise Service Bus (ESB).

Es la infraestructura de SOA. → Aporta INTEROPERABILIDAD (atributo de calidad).

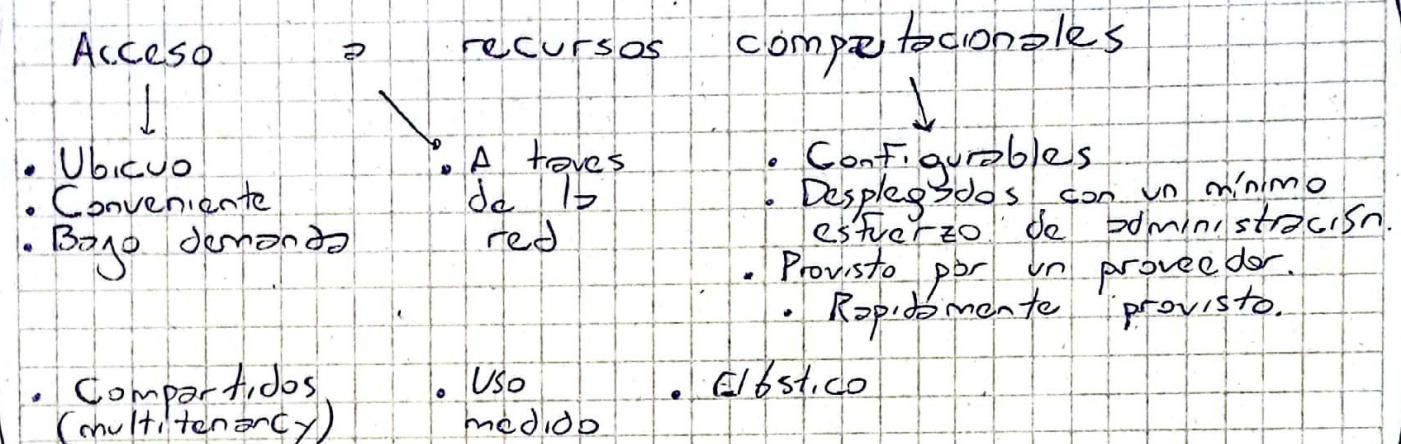
El consumidor solicita un servicio a ESB, él lo consigue.	→ Permite que varios proveedores ofrezcan el mismo servicio.	{ Ejemplo consumo geolocalización GPS vs Galileo ?
(ESB puede ser load balancer.)	FSB autoriza o no el acceso a diferentes servicios (SEGURIDAD).	
(ESB maneja errores (RELIABILITY) FIABILIDAD).	ESB ayuda al monitoreo, logea qué se hace.	

SOA no es solo software. También es gestión, convenciones, protocolos, manejar a los diferentes owners.

Requiere un equipo centralizado para definir cuestiones de toda la empresa.

Apoyo CEO y CIO. Es un proceso importante que toma a TODA LA EMPRESA.

Cloud Computing



- 6 características
- On demand
 - Measured
 - Multitenant
 - Elástico
 - Ubiquitous
 - Resiliency

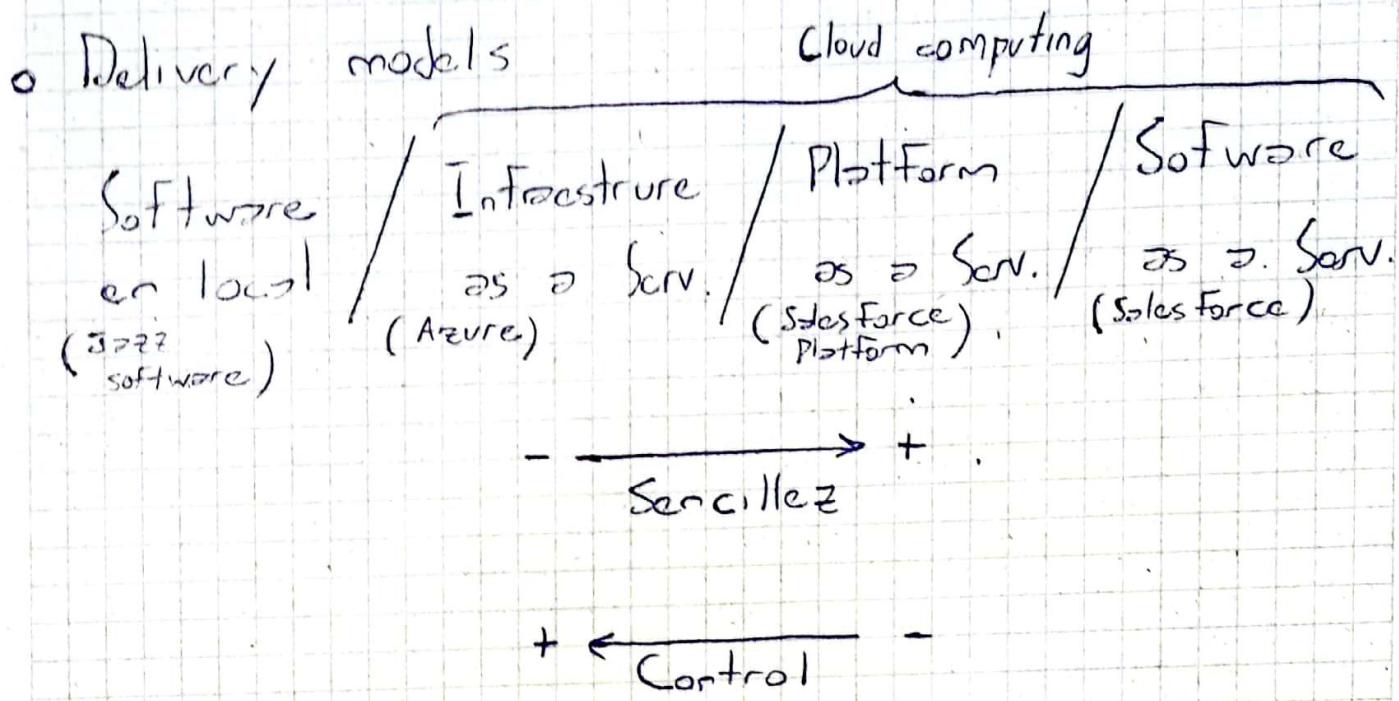
Beneficios

- Menor inversión inicial
- Costo proporcional al consumo
- "Infinitos" recursos disponibles
- Escalabilidad
- Elasticiadad
- Disponibilidad

VS

Riesgos

- Seguridad (responsabilidad compartida)
- Menor control
- Baja portabilidad.



o Arquitecturas Cloud Habituales

- N-tier → Separación de responsabilidades en capas. Cada capa está en una máquina (físico separado). Los capas altas consumen a las bajas.
 - Web - queue - worker
 - Microservices
 - Event - Driven
- Un event producer produce un evento, que le llega a todos los consumidores (subscriptores). Los consumers no se conocen y es unidireccional.
- El client consume a través de un Front-end (web), enviando a un worker a realizar la tarea (algo). Las tareas se encuelan (queue).
- Servicios autónomos pequeños, cada servicio representa un área del negocio. Se comunican a través de APIs.

"Buenas soluciones → problemas recurrentes"

Cloud usage patterns

Auto-scaling → Asignación dinámica mediante reglas para cumplir con la estrategia y Escalado horizontal.

Availability
Scalability
Performance

Queue-Based Load Leveling → Encolar los mensajes del cliente al servicio. (unidireccional). (para un service provider).

Availability
Scalability
Performance

Comparting customers. → Colas de mensajes (para varios service providers). Sirve cuando no importa qué worker lo procesa.

Scalability
Performance

Priority queue → Colas de mensajes, pero se prioriza los mensajes más importantes. (para cumplir con un SLA).

Availability
Performance

Health Endpoint Monitoring → Un endpoint que periódicamente hace llamadas para probar el servicio. Logra sus resultados.

Availability

Retry → Frente a fallos temporales, reiniciaremos el pedido (después de X tiempo, por ejemplo).

Availability

Circuit Breaker → Frente a fallos largos, un proxy debería gestionar si llamar o no a un servicio.

Availability
Performance

Throttling → Limitar el uso de recursos, por ejemplo si un mismo usuario esté usando todo. Es una alternativa a auto-scaling.

Availability,
Scalability
Performance

Sharding → Cada servidor contiene una partición de la información. Según que se necesita, se consulta una u otras.

- no SQL (y BD relacionales).
 - BD Relacionales →
 - Gran almacenamiento
 - Modelo estandar (SQL).
 - Buena Apto concurrencia
 - Soporta varios aplicaciones
 - ACID.
 - No puede contener estructuras de datos
 - Incovenientes al ser usado por varios apps.
 - Difícil escalar verticalmente. Tiene que hacer clustering (las BD relacionales no fueron diseñadas para esto).
- no SQL → Bases de datos no relacionales
 - i DISEÑADO PARA HACER CLUSTERS ! (escalar horizontalmente).
 - Guarda información con agregados (no relacional)
 - Sin esquema. Son más flexibles que los relacionales.
- Modelos de distribución de datos
 - Al distribuir se puede.
 - { Replicación → Copiar unos mismos datos en distintos nodos
 - Sharding → Distribuir los datos en los distintos nodos.
 - Sharding
 - Aprovechar geolocalización
 - Al "unir" uniendo datos que se acceden siempre a la vez.
 - Replication
 - { Master-slave
 - Peer-to-peer
 - Sharding + Master-slave / Sharding + Peer-to-peer.

o Consistencia.

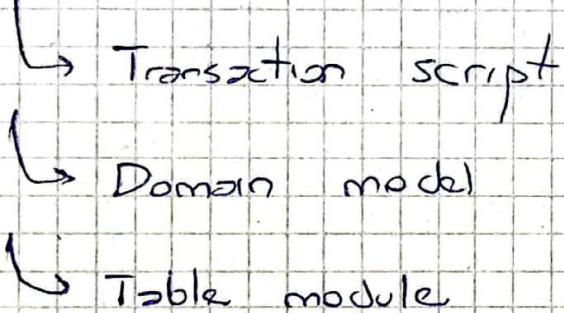
Los tipos no relacionales no tienen consistencia tan fuerte como los relacionales (ACID).

• Patterns of Enterprise App. Architecture

Patrones para → Aplicaciones Enterprise.

- Patrones para la organización de las lógicas del negocio.

Lógica de negocio.



Modelado en software de las reglas reales del negocio.

- Transaction Script. → 1 procedimiento (script) para cada transacción (acción)

No hay objetos con comportamiento propio.

- Domain model → Modelado, con objetos. Cada elemento en la realidad tiene una instancia en el software, con comportamiento propio. Mala integración con BD.

- Table module → Similar a objetos, pero con una clase por cada tabla que tenemos en la base de datos. Tiene una sola instancia para toda la tabla.

+

- Service layer. → La frontera del sistema. Este se usa mediante una API, y no se expone su funcionamiento interno.