Arquitectura del software

Ya se empieza a detallar el *cómo* se va a resolver el problema
Es la primera de las tres etapas de **diseño** y la de más alto nivel:

1. **Arquitectura**
2. Diseño Alto Nivel
3. Diseño Bajo Nivel

Da una vision de las partes del sistema y de las relaciones entre ellos.
La arquitectura permite que los cambios sean simples de hacer en sistemas grandes y complejos. Permite *information hiding* y *encapsulación*.

El objetivo de la arquitectura es identificar subsistemas y la forma que interactúan entre ellos.

Definición:
La *arquitectura de software* de un sistema es la estructura del sistema que comprende los elementos del software, las propiedades externamente visibles de tales elementos, y la relación entre ellas.

No son importantes los detalles de como se aseguran las propiedades externas.
En general se tienen varias estructuras (aspectos ortogonales).

A este nivel se hacen las elecciones de tecnología, productos a utilizar, servidores, ...
Es la etapa más temprana donde algunas propiedades como confiabilidad y desempeño pueden evaluarse. Aunque a nivel arquitectura se omiten muchos detalles.

¿Por qué es importante?

- **Comprensión y comunicación**: Muestra la *estructura de alto nivel* del sistema ocultando la complejidad de sus partes, lo cual define un marco de *comprensión común* entre los distintos interesados (usuarios, cliente, arquitecto, diseñador, ...).

- **Reuso**: Una de las principales técnicas para incrementar la *productividad*.
  Se elige una arquitectura tal que las *componentes existentes encajen* adecuadamente con otras componentes a desarrollar.

- **Construcción y evolución**: la división provista servirá para *guiar* el desarrollo y *asignar equipos de trabajo*. Durante la evolución del software, al arquitectura ayuda a dimensionar cuán caro puede costar un cambio.

- **Análisis**: La arquitectura permite predecir parcialmente propiedades de *confiabilidad y desempeño*. Se requerirá descripción precisa de la arquitectura así como de las propiedades de las componentes.

No hay una única arquitectura de un sistema.

Vistas de la arquitectura

Una vista consiste de **elementos** y **relaciones** entre ellos y **describe una estructura**. Destacan un aspecto. Por lo tanto, no hay una única vista de un sistema.
Una vista que se enfoca en algún aspecto reduce la complejidad con la que debe enfrentarse el lector.

La mayoría de las vistas propuestas pertenece a alguno de estos tipos:

1. **Vista de módulos**: estructuras de código, planeamiento.
2. **Vista de Componentes y conectores**: estructuras de ejecución, análisis de desempeño.
3. **Vista de Asignación de recursos**: co-estructuras de software y entorno.

Existen relaciones entre los elementos de una vista y los de otra; y tal relación puede ser muy compleja.

Vista de módulos

Un sistema es una colección de **unidades de código** (no representan entidades en ejecución). Por ejemplo: clases, paquetes, funciones, procedimientos, métodos, etc.

La **relación** entre ellos está **basada en el código**. Por ejemplo: "parte de", "usa a", "depende de", llamadas, generalización o especialización, etc.

Vista de Asignación de Recursos

Se enfoca en *cómo* las unidades de software se asignan a recursos como hardware, sistemas de archivos, personas, etc.

Exponen propiedades estructurales como qué proceso ejecuta en qué procesador, qué archivo reside dónde, etc.

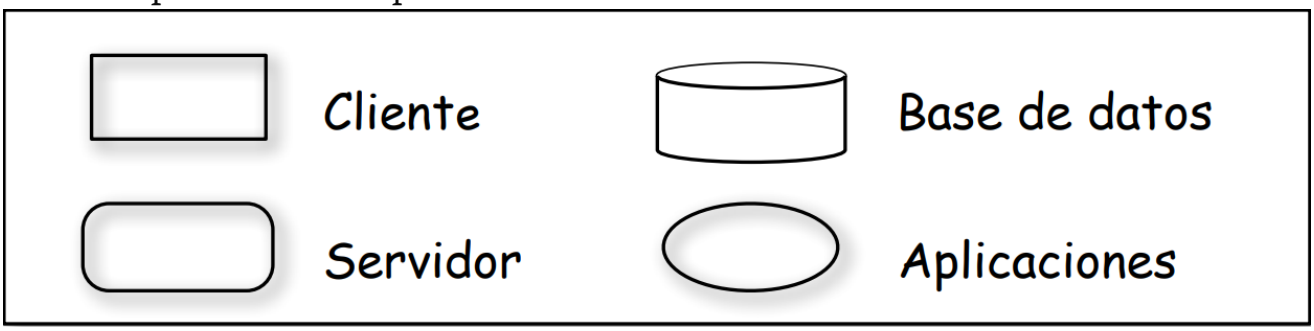
Vista de Componentes y conectores

Los elementos son **entidades de ejecución** denominados componentes. La vista C&C define las componentes y cómo se conectan entre ellas a través de conectores. Describe un estructura en ejecución del sistema: qué componentes existen y cómo interactúan entre ellos en **tiempo de ejecución**.

Componentes

Son **elementos computacionales** o **de almacenamiento** de datos. Por ejemplo: objetos, procesos, `.exe`, `.dll`, etc.
Las componentes utilizan interfaces o puertos para comunicarse con otras componentes.

Cada componente tiene tipos.



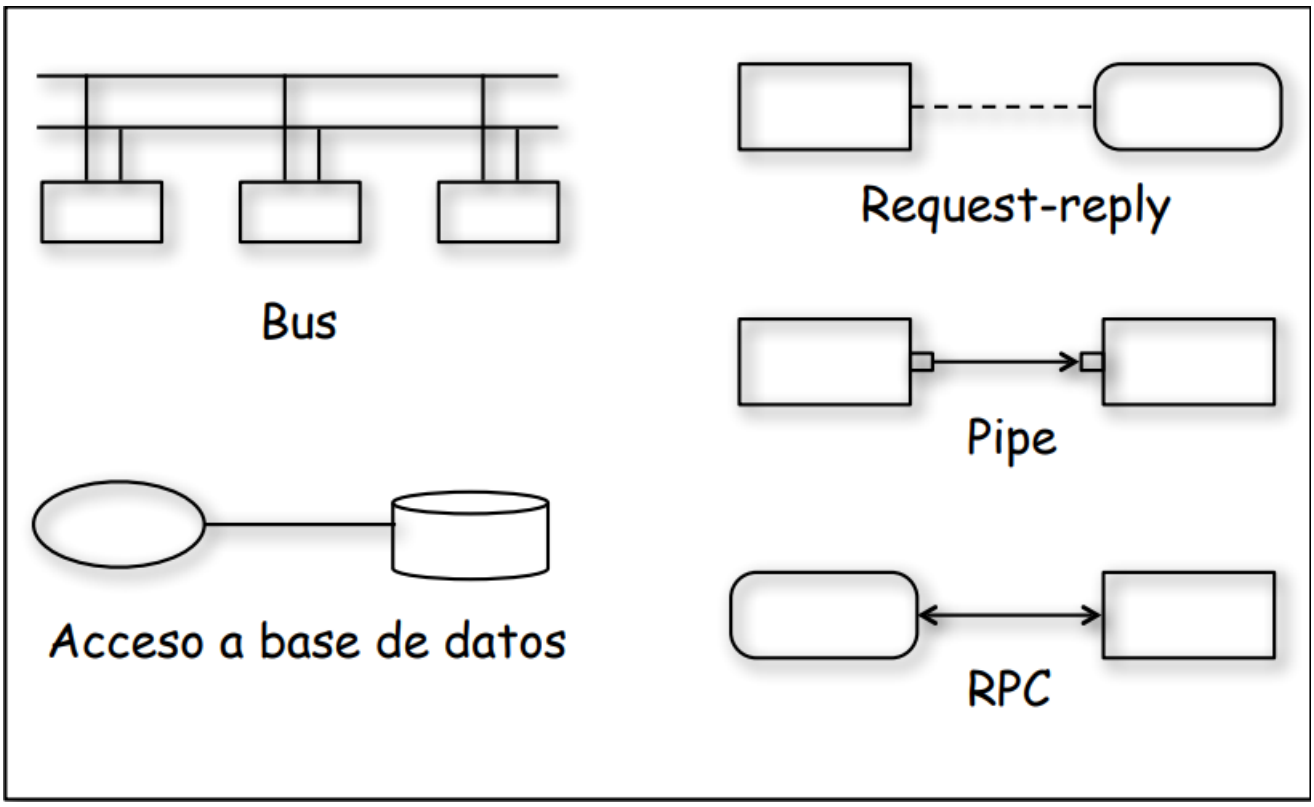
Conectores

Son mecanismo de **interacción** entre las componentes. Los conectores describen el **medio** en el cual la interacción entre componentes toma lugar.

Por ejemplo: pipes, sockets, memoria compartida, protocolos,llamada a procedimiento/función, puertos TCP/IP, RPC, etc.

Los conectores no necesariamente son binarios. Tienen nombre y tipo.

Notar que estos mecanismos requieren una **infraestructura de ejecución** significativa y **programación especial** dentro de la componente para poder utilizarla.



Estilos arquitectónicos para la vista de C&C

Define una familia de arquitecturas que satisfacen las restricciones de ese estilo.
Distintos estilos pueden combinarse.

En general se usan diferentes combinaciones de estilos o se usan estilos nuevos.

Pipe and Filter

Adecuado para sistemas que fundamentalmente realizan *transformaciones de datos*.
Un filtro realiza transformaciones y le pasa los datos a otro filtro a través de un tubo.

- Tipo de componente: filtro: es **independiente** y **asíncrona**, se limita a consumir y producir datos.
- Tipo de conector: pipe (tubo). Unidireccional y no tiene bifurcaciones. Cada tubo solo conecta 2 componentes.

Restricciones

- Cada filtro debe trabajar sin (necesariamente) conocer la identidad de los filtros productores o consumidores.
- Un tubo debe conectar un puerto de salida de un filtro a un puerto de entrada de otro filtro.
- Un sistema *puro* de *Pipe&Filter* usualmente requiere que cada filtro tenga su propio hilo de control.

Estilo de Datos Compartidos

- Componentes: **Repositorio** de datos y **usuario** de datos.
  - Repositorio de datos: provee **almacenamiento** permanente y confiable.
  - Usuarios de datos: acceden a los datos en el repositorio, **realizan cálculos** y ponen los resultados otra vez en el repositorio.
- Conector: un solo tipo, de **lectura/escritura**.

La comunicación entre los usuarios sólo se hace a través del repositorio, los componentes no actúan directamente entre ellas.

Estilo pizarra
Cuando se modifica el repositorio, se **informa** a todos los usuarios.

Estilo repositorio
Es pasivo.

Estilo cliente-servidor

- Componentes: **clientes** y **servidores**
  - Los clientes solo se comunican con el servidor, no con otros clientes.
  - La comunicación es siempre iniciada por el cliente y usualmente sincrónica.
- Conector: solicitud/respuesta (*request/reply*), es asimétrico.

Usualmente el cliente y el servidor residen en distintas máquinas.

Estructura multinivel

Nivel	Descripción
del cliente	Contiene a los clientes.
intermedio	Contiene las reglas del servicio.
de base de datos	reside la información.

Otros estilos

Publicar-suscribir

- Componentes: los que publican eventos y los que se suscriben a eventos.
- Se publica un evento → se invoca a las componentes suscritas a dicho evento.

Estilo peer-to-peer
Componente: Peer (único tipo), pueden pedir servicios a otros peers.
Parecido al modelo de computación orientada a objetos.

Estilo de procesos que se comunican
Procesos que se comunican a través de mensajes.

Documentación del diseño arquitectónico

El documento de diseño arquitectónico debe especificar precisamente las vistas y las relaciones entre éstas.

Organización del documento

1. Contexto del sistema y la arquitectura
2. Descripción o detalle de las vistas
    1. Presentación principal de la vista
    2. Catálogo de elementos
    3. Fundamento de la arquitectura
    4. Comportamiento
    5. Otra información
3. Documentación transversal a las vistas

Contexto del sistema y la arquitectura

Un diagrama de contexto provee el contexto **general**. Establece:

- el alcance del sistema
- los actores principales
- las fuentes y consumidores de datos.

Detalle de las vistas

Uno por cada uno de los distintos tipos de vistas que se eligieron representar. Mientras más, mejor.

1. **Presentación principal de la vista**: Descripción gráfica, como se presentó anteriormente. Algunas vistas podrían combinarse en un solo diagrama, no graficar el diagrama si no es fácil de comprender.
2. **Catálogo de elementos**: Más información sobre los elementos que se muestran en la presentación principal. Por c/elemento describe su **propósito** y sus **interfaces** (tanto sintaxis como semántica).
3. **Fundamento de la arquitectura**: Justificaciones de las decisiones. Podría proveer una discusión sobre las alternativas consideradas y descartadas. Para futuros desarrolladores. Decir explícitamente el porqué y el porqué no.
4. **Comportamiento**: Comportamiento real del sistema/componente en algunos escenarios. Útil para analizar propiedades. Se especifica un poco el *cómo*.
5. **Otra información**: Por ejemplo, decisiones dejadas intencionalmente para el futuro.

Documentacion transversal a las vistas

Describe cómo los elementos de las distintas vistas se relacionan entre sí. Justificación de las vistas elegidas + otro tipo de info transversal.

Relación entre Arquitectura y Diseño

- La arquitectura **es** un diseño: se encuentra en el dominio de la solución y no en el del problema.
- Es un diseño de **muy alto nivel** que se enfoca en las componentes principales. La arquitectura no considera la estructura interna.
- La arquitectura impone restricciones sobre elecciones que pueden realizarse en otras fases del diseño y en la implementación.
- Para que la arquitectura tenga sentido, ésta debe *acompañar el diseño y el desarrollo del sistema*.

Evaluación de las arquitecturas

La arquitectura tiene **impacto** sobre los **atributos no funcionales**
(como modificabilidad, desempeño, confiabilidad, portabilidad, etc).
Por lo tanto se deben evaluar estas propiedades en la arquitectura propuesta.

Métodos para evaluar propiedades:
Técnicas formales: redes de colas, model checkers, lenguajes de especificación, ...
Otra posibilidad: metodologías rigurosas. Como el método de análisis ATAM.

Método de análisis ATAM

Architecture Trade off Analysis Method
Analiza las propiedades y las concesiones entre ellas.

Pasos principales

1. **Recolectar escenarios**:
   - Los escenarios *describen las interacciones del sistema*.
   - Elegir los escenarios de interés para el análisis (escenarios *críticos*).
   - Incluir escenarios excepcionales solo si son importantes
2. **Recolectar requerimientos y/o restricciones**:
   - Definir *lo que se espera del sistema* en tales escenarios.
   - Deben especificar los *niveles deseados* para los **atributos de interés** (preferiblemente cuantificados).
3. **Describir las vistas arquitectónicas**
   - Las vistas del sistema que serán evaluadas son recolectadas.
   - Distintas vistas pueden ser necesarias para distintos análisis.
4. **Análisis específicos a cada atributo**.
   - Se analizan las vistas bajo distintos escenarios separadamente para cada atributo de interés distinto; esto determina los niveles que la arquitectura puede proveer en cada atributo.
   - Se comparan esos niveles con los requeridos: Esto forma la base para la elección entre una arquitectura u otra o la modificación de la arquitectura

propuesta.
    - Puede utilizarse cualquier técnica o modelado.
5. **Identificar puntos sensitivos y de compromisos**
    - Análisis de sensibilidad: cuál es el impacto que tiene un elemento sobre un atributo de calidad. Los elementos de mayor impacto son los puntos de sensibilidad.
    - Análisis de compromiso: Los puntos de compromiso son los elementos que son puntos de sensibilidad para varios atributos.

ATAM vs. CBAM

CBAM: *Cost-Benefit Analysis Method*

ATAM	CBAM*
Desempeño	Costo del desempeño
Escalabilidad	Costo de la escalabilidad
Disponibilidad	Costo de la disponibilidad

* Esta medida se está utilizando más ahora.

Practico3