

# Introducción a la Arquitectura de Software

MSc. César Augusto López Gallego

Profesor Facultad Ingeniería en TIC – UPB

Coordinador Área de Programación, Computación y Desarrollo de Software

[cesar.lopezg@upb.edu.co](mailto:cesar.lopezg@upb.edu.co)

 por Cesar Lopez



# Es el Coliseo Romano una Arquitectura?

El coliseo Romano es el resultado de una arquitectura. El resultado de una arquitectura es una instancia de ella, una implementación. Si los arquitectos no hubiesen creado una representación descriptiva, ellos seguramente no lo habían podido construir.

La arquitectura es un conjunto de representaciones que son requeridas para crear un objeto.

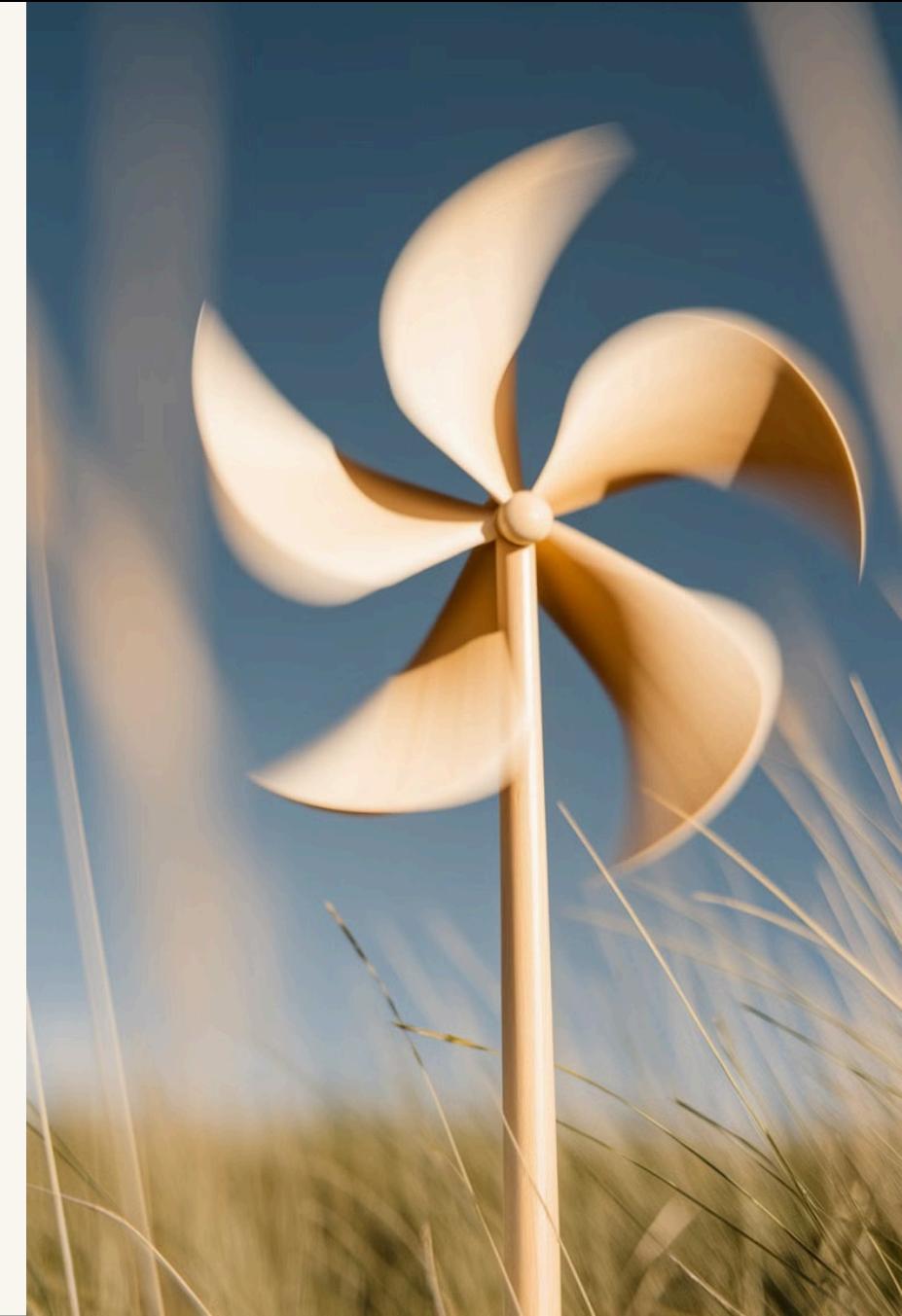


# ¿Se puede construir algo sin arquitectura?

Simple

Empírico

Prestaciones Básicas



# Definición de Arquitectura

## Descripción Formal

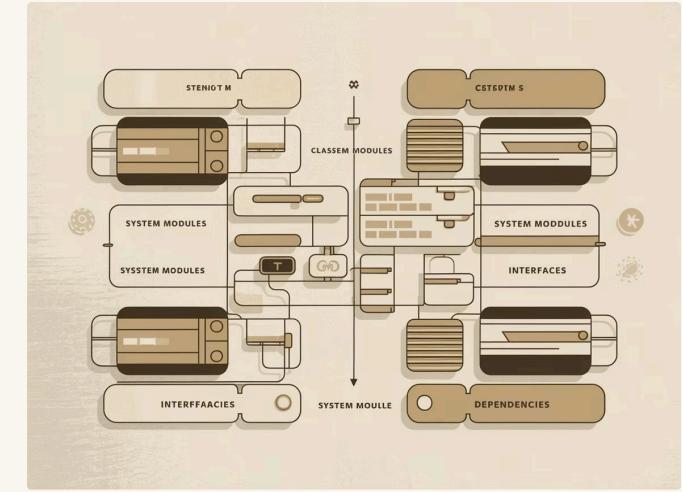
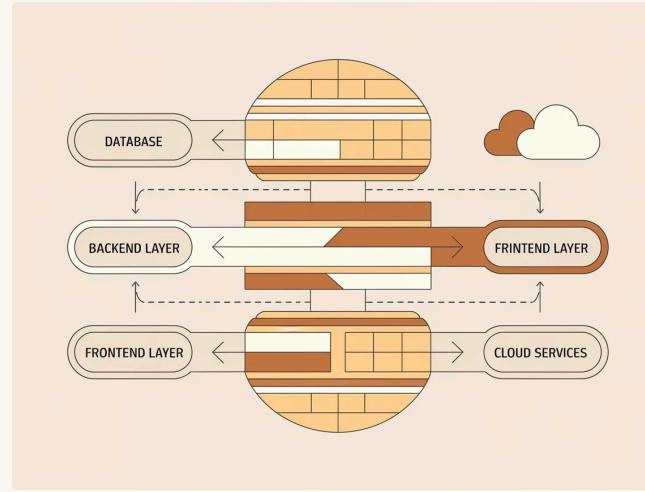
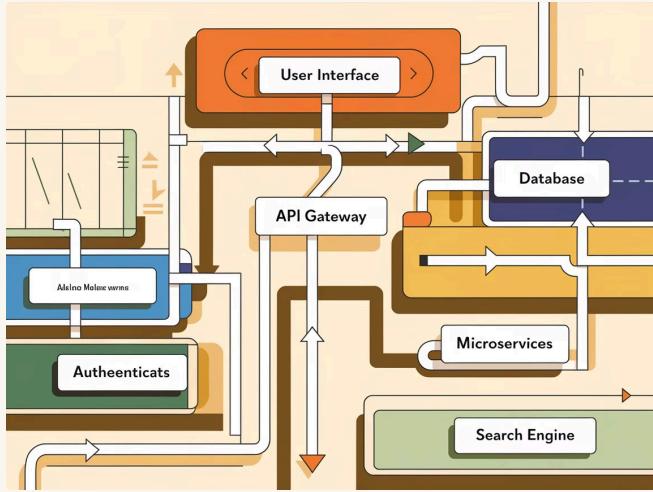
Un plano detallado del sistema, a nivel de sus componentes, para guiar su implementación.

## Estructura y Principios

Define la estructura de los componentes, sus interrelaciones y los principios que rigen su diseño y evolución.



# ¿Qué es la Arquitectura de Software?



## Definición 1

Se define arquitectura software como la estructura del sistema, que comprende elementos software, las propiedades de esos elementos visibles externamente y las relaciones entre ellos. La arquitectura software conforma el esqueleto de cualquier sistema software, y es la principal responsable de los atributos de calidad del sistema.

## Definición 2

Arquitectura de Software es la especificación técnica que explica cómo debiera organizarse y funcionar una infraestructura tecnológica a través de patrones y métodos para procesar y producir información coherentes al medio organizacional al cual se aplica.

## Definición 3

Es la organización fundamental de un sistema, que se detalla en: componentes, la relación que existe entre ellos y el ambiente, también los principios que guían el diseño y evolución del mismo.

IEEE 1471-2000

# Con Arquitectura de Software

## Eficiencia en el Desarrollo

Mejora la eficiencia al proporcionar una estructura clara para el desarrollo. Facilita la reutilización de componentes.

## Calidad del Producto

Aumenta la calidad y confiabilidad del software al seguir estándares y patrones establecidos.

## Mantenimiento y Escalabilidad

Facilita el mantenimiento y la escalabilidad del sistema, permitiendo adaptaciones a futuras necesidades.

## Integración de Sistemas

Permite una integración más fácil entre diferentes sistemas y componentes, mejorando la interoperabilidad.

## Gestión de Riesgos

Ayuda a identificar y mitigar riesgos potenciales desde las etapas iniciales del desarrollo.

Con una arquitectura de software bien definida, se asegura una mejor alineación estratégica y colaboración efectiva entre equipos, aunque requiere una inversión inicial que se justifica por los beneficios a largo plazo.

# Sin Arquitectura de Software

## Caos en el Desarrollo

Falta de estructura y dirección clara, resultando en código desorganizado y difícil de mantener

## Problemas de Calidad

Ausencia de estándares y patrones que lleva a inconsistencias y errores frecuentes

## Dificultad de Mantenimiento

Código difícil de modificar y actualizar, con alto riesgo de efectos secundarios no deseados

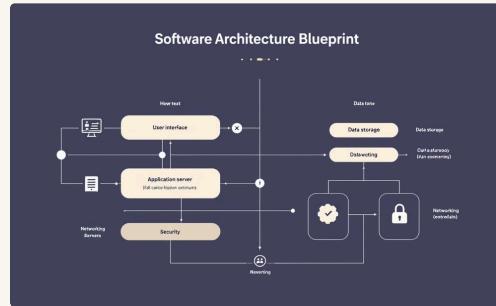
## Complejidad en la Integración

Sistemas aislados y desconectados que dificultan la comunicación entre componentes

## Riesgos Elevados

Mayor probabilidad de fallos críticos y problemas de rendimiento sin detección temprana

# Implicaciones de la Arquitectura de Software

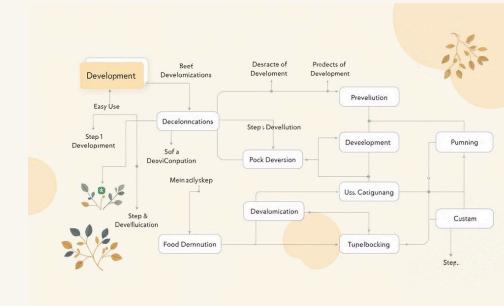


## Diseño de Alto Nivel

Define la estructura y organización general del sistema, similar a un plano de una casa.

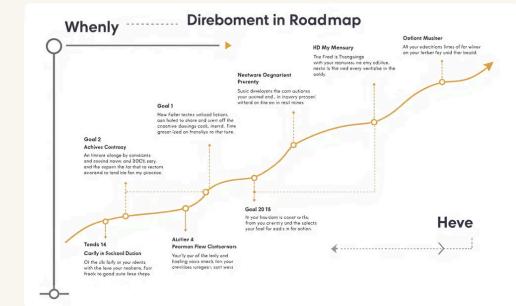
## Abstracciones y Patrones

Utiliza conceptos y estructuras predefinidas para facilitar la organización y el desarrollo.



## Marco de Producción

Establece las directrices para la construcción y el desarrollo del sistema, asegurando la coherencia y la eficiencia.



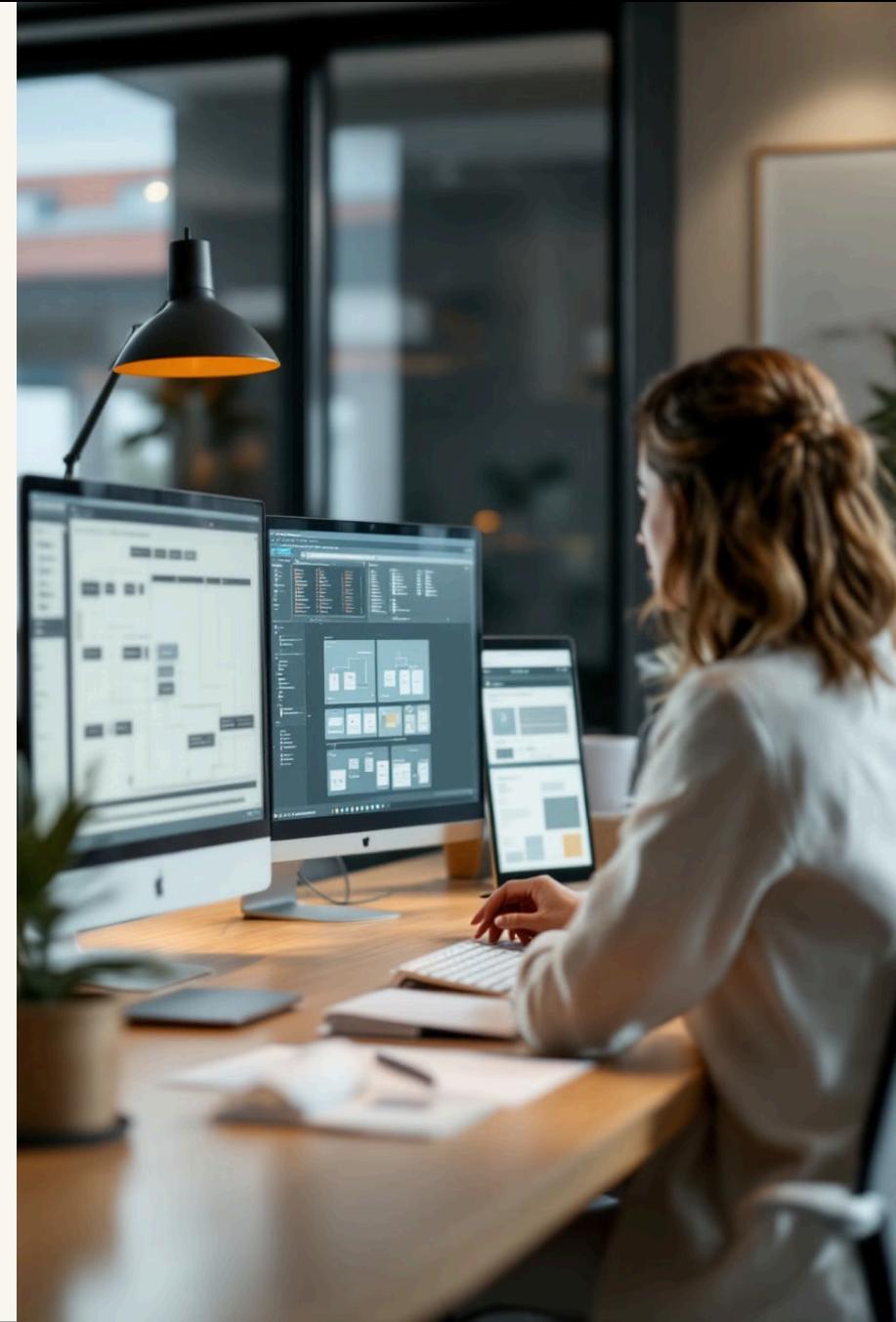
## Objetivos de Desarrollo

Guia el proceso de desarrollo hacia el cumplimiento de las metas y requisitos específicos.

# ¿Quién es un Arquitecto de Software?

Un arquitecto de software es un profesional responsable de diseñar la estructura general de un sistema de software.

Su rol es crucial para definir la estructura, los componentes y las interacciones de un sistema, asegurando que se cumplan los requisitos técnicos y funcionales del proyecto.





# Procesos de Diseño Arquitectónico

## Big Design Up Front

Se diseña una arquitectura completa antes de comenzar el proceso de desarrollo.

## Adaptativa

Se basa en los principios de agilidad, y propone el diseño de una arquitectura por sprints y de forma incremental.

# Ejemplo de Proceso Arquitectónico

## Stakeholders

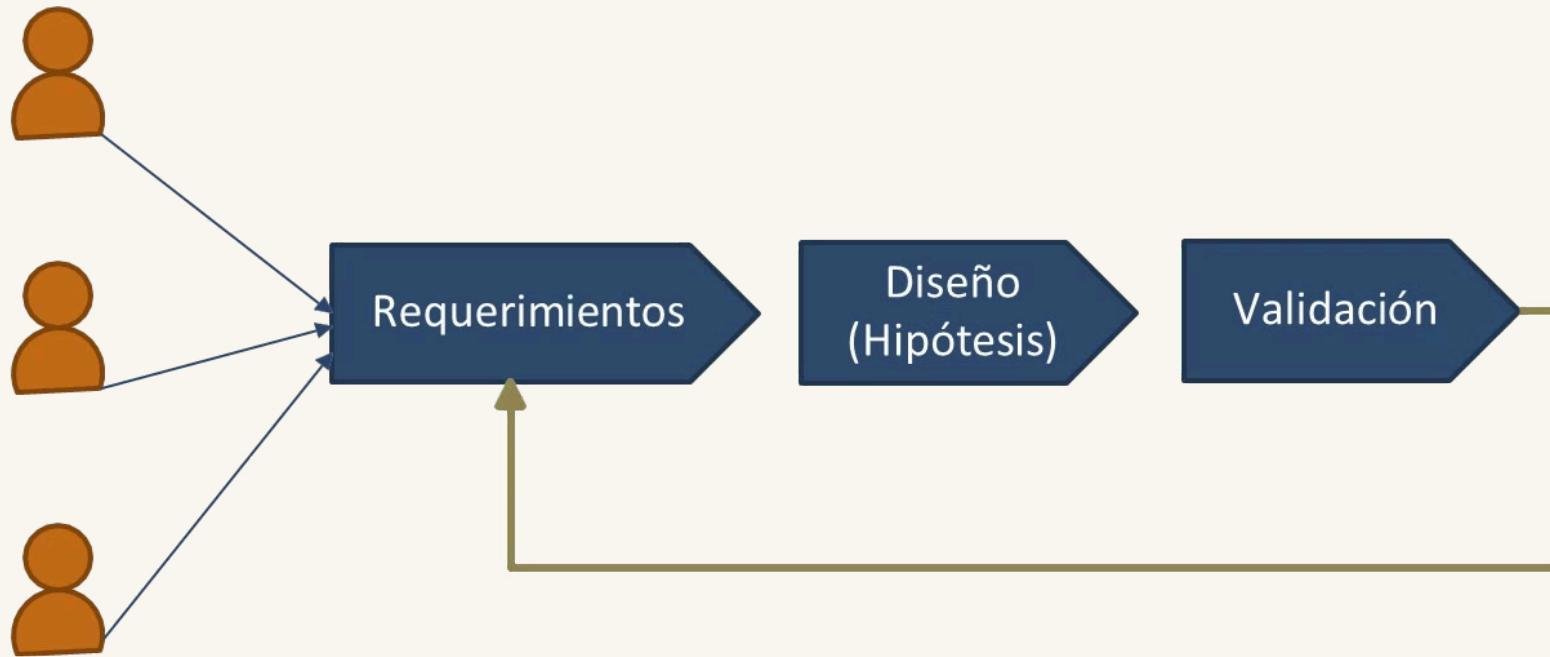
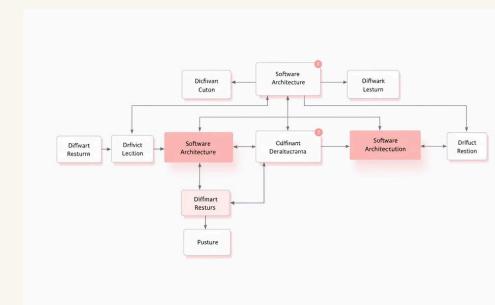
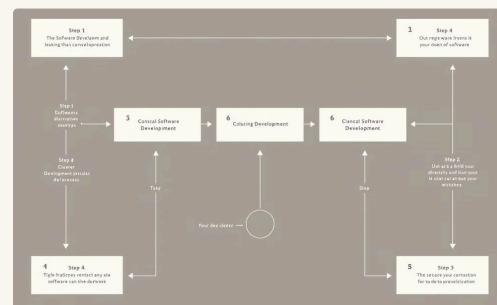


Imagen: Curso Arquitectura de Software Universidad de los Andes



## Definición de Requerimientos

Comienza con una comprensión clara de las necesidades del usuario y las funciones que el sistema debe cumplir.

## Diseño Arquitectónico

Se definen los componentes principales del sistema, sus interacciones y las tecnologías a utilizar.

## Desarrollo e Implementación

El equipo de desarrollo construye el sistema basándose en el diseño arquitectónico.

## Pruebas y Validación

Se asegura que el sistema cumple con los requerimientos y funciona correctamente.

# Proceso Arquitectónico: Requerimientos

El proceso arquitectónico inicia identificando los requerimientos funcionales (qué debe hacer el sistema) y los requerimientos de calidad (cómo debe hacerlo). La arquitectura debe satisfacer tanto las necesidades operativas como los atributos de calidad: escalabilidad, seguridad y rendimiento.

El arquitecto colabora con los stakeholders para priorizar requerimientos y resolver conflictos entre atributos de calidad, buscando un balance efectivo entre las necesidades del negocio y las restricciones técnicas.



# Proceso Arquitectónico: Diseño

El proceso de diseño arquitectónico implica la toma de decisiones fundamentales sobre la estructura del sistema de software. Comienza con la descomposición del sistema en componentes manejables y la definición de sus responsabilidades.

Los principios clave incluyen el encapsulamiento para ocultar la complejidad interna, la identificación de oportunidades de reuso de componentes, y el establecimiento de esquemas de comunicación entre procesos. Se debe prestar especial atención a los atributos de calidad como los tiempos de respuesta.

Durante esta fase, se definen las interfaces entre componentes, los patrones de interacción, y los mecanismos de integración que asegurarán el funcionamiento cohesivo del sistema.

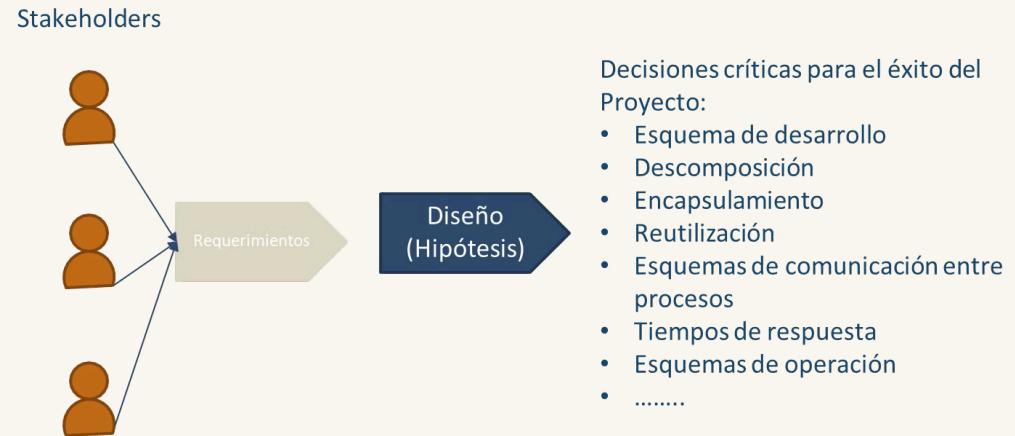


Imagen: Curso Arquitectura de Software Universidad de los Andes

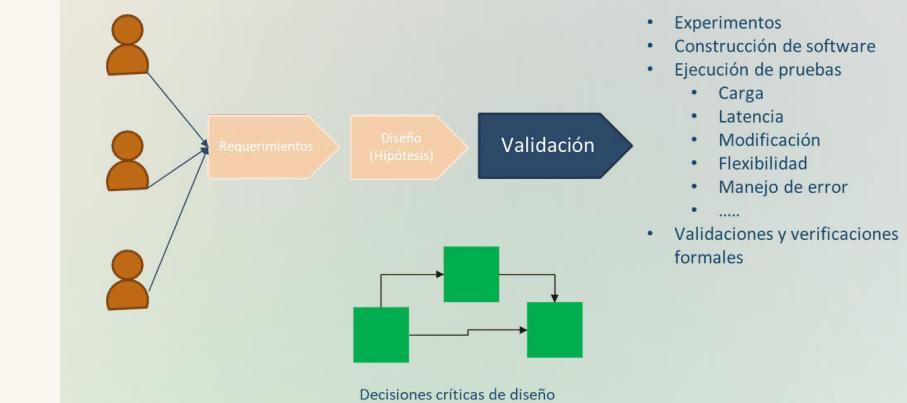
# Proceso Arquitectónico: Validación

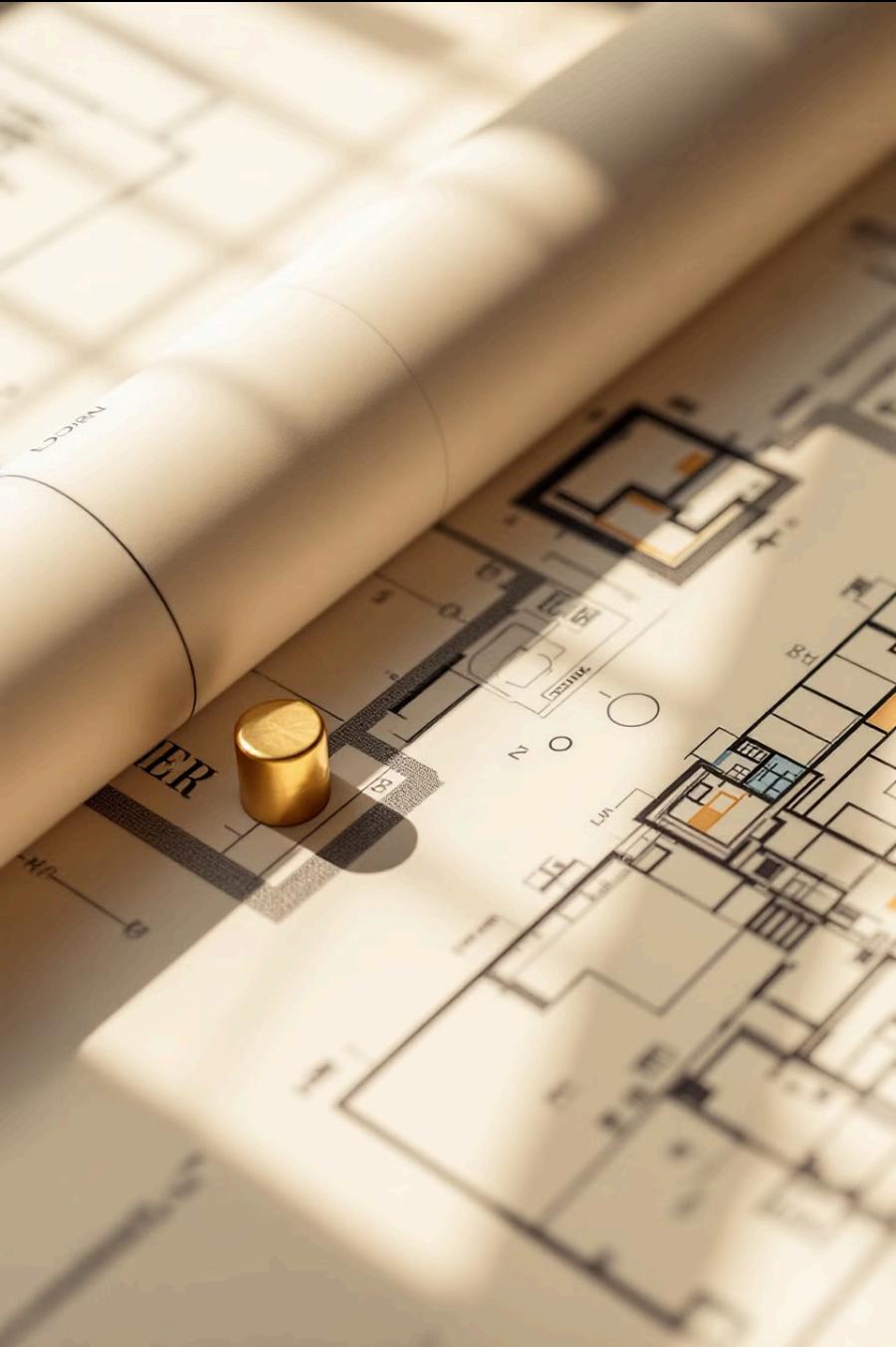
La validación arquitectónica implica pruebas exhaustivas que incluyen tests unitarios, pruebas de integración y validaciones end-to-end para asegurar que el sistema cumple con los requerimientos establecidos.

Se realizan pruebas de carga y estrés para verificar la escalabilidad, medir la latencia bajo diferentes condiciones y evaluar el rendimiento general del sistema en situaciones críticas.

Es fundamental validar los mecanismos de manejo de errores, la tolerancia a fallos y la capacidad de recuperación del sistema, asegurando su robustez en producción.

Imagen: Curso Arquitectura de Software Universidad de los Andes





# Vistas Arquitectónicas

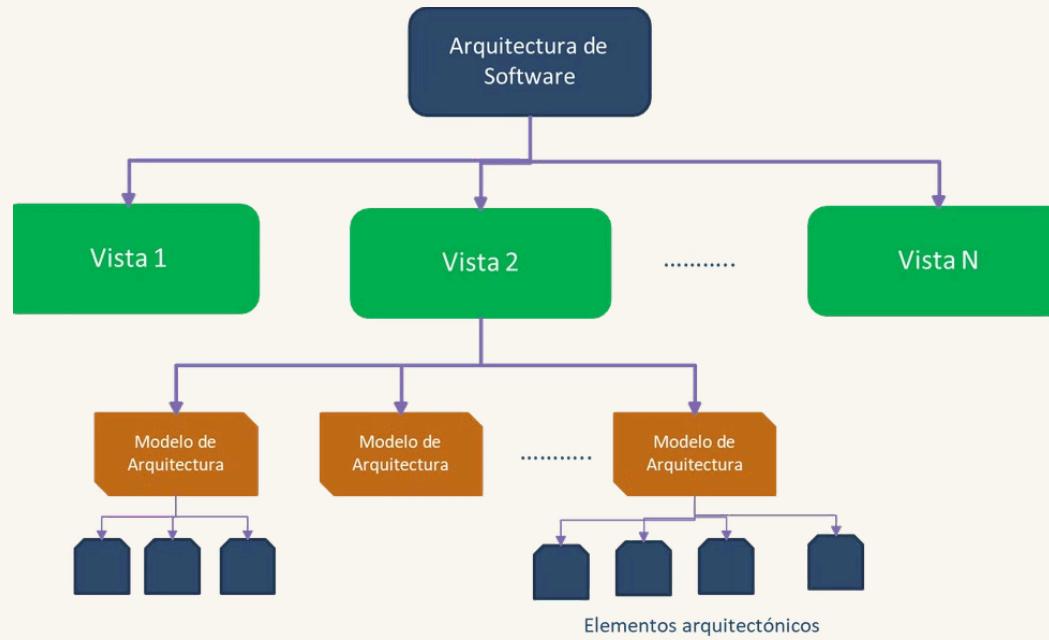
## Perspectiva

Una vista es la aplicación de un punto de vista en una arquitectura de software específica.

## Ejemplo

Por ejemplo, una vista lógica representa la interacción entre los componentes del software, mientras que una vista física describe cómo se implementan los componentes en el hardware.

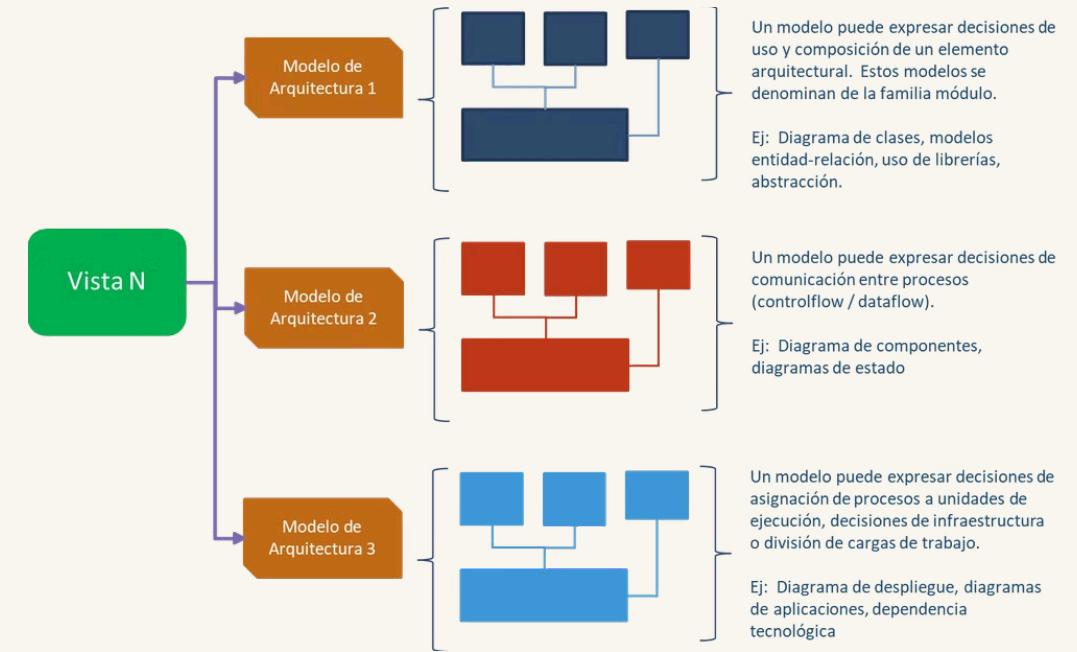
# Ejemplo de Vistas Arquitectónicas



## Vista Lógica

Muestra la interacción entre los componentes del sistema, sin importar su implementación física.

Imagen: Curso Arquitectura de Software Universidad de los Andes



## Vista de Desarrollo

Describe la organización del código fuente, las dependencias y la estructura del sistema.

# Puntos de Vista: Modelo 4+1 (*Philippe Kruchten*)

El modelo 4+1 es un marco de referencia para describir la arquitectura de sistemas software intensivos, basados en el uso de múltiples vistas concurrentes.

## Vista Lógica

Describe la estructura y funcionalidad del sistema, identificando los componentes y sus interacciones.

## Vista de Desarrollo

Se centra en la organización del código y los artefactos de software, mostrando cómo se gestionan y desarrollan los componentes.

## Vista de Proceso

Aborda el comportamiento dinámico del sistema, incluyendo aspectos como la concurrencia y la comunicación entre procesos durante la ejecución.

## Vista Física

Relacionada con la infraestructura tecnológica, describe cómo se despliegan los componentes en el hardware y las conexiones físicas entre ellos.

## Escenarios

Proporcionan casos de uso concretos que validan el diseño arquitectónico e ilustran cómo las diferentes vistas trabajan juntas para cumplir los requisitos del sistema.

# Vista Lógica y Vista de Desarrollo

## Vista Lógica

Describe la funcionalidad del sistema desde la perspectiva de los objetos y sus interacciones.

Utiliza diagramas de clases y comunicación (UML) para representar la estructura interna.

**Interesados:** Esta vista está dirigida a diseñadores de software, arquitectos y desarrolladores que necesitan entender cómo se organizan los componentes del sistema.

## Vista de Desarrollo

Muestra la organización del software desde la perspectiva del desarrollador.

Utiliza diagramas de componentes y paquetes (UML) para representar la organización del código.

**Interesados:** Está dirigida a desarrolladores, ingenieros de software y gerentes de proyecto que requieren información sobre la estructura del código y su organización.

# Vista de Proceso y Vista Física

## Vista de Proceso

Se centra en la interacción de procesos, cómo se comunican y su comportamiento durante la ejecución.

**Modelos o Diagramas:** Se utilizan el Diagrama de Actividad y el Diagrama de Secuencia (UML).

**Interesados:** Esta vista es relevante para desarrolladores, arquitectos de software y analistas que están interesados en la concurrencia, rendimiento y escalabilidad del sistema.

## Vista Física

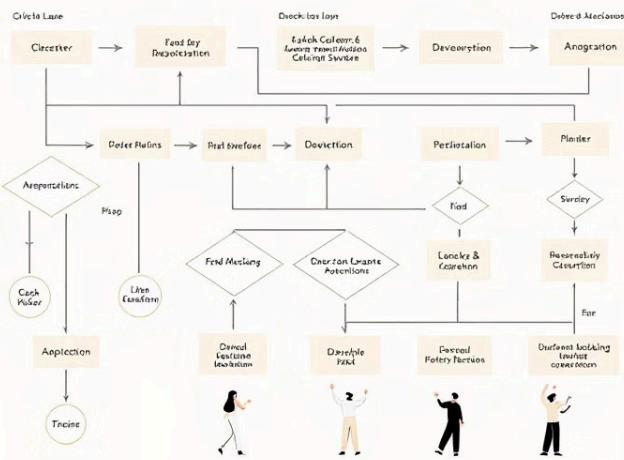
Describe la distribución física de los componentes del sistema, incluyendo su ubicación en hardware.

**Modelos o Diagramas:** Se utiliza el Diagrama de Despliegue (UML).

**Interesados:** Ingenieros de sistemas y administradores de infraestructura que necesitan entender cómo se implementa físicamente el sistema.



Fame  
Soil one o' of funn you leave's a new software application



# Escenarios



## Objetivo

Ilustrar la arquitectura mediante un conjunto de casos de uso que describen interacciones entre objetos y procesos.



## Modelos

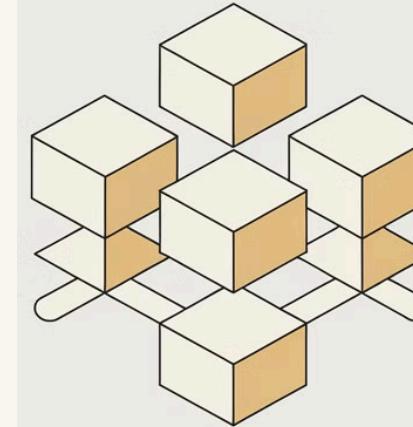
Se utilizan descripciones narrativas o diagramas UML para representar casos de uso.



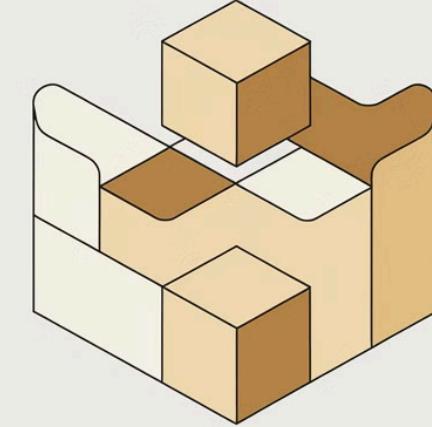
## Interesados

Esta vista está dirigida a todos los interesados, incluidos usuarios finales, desarrolladores y gerentes.

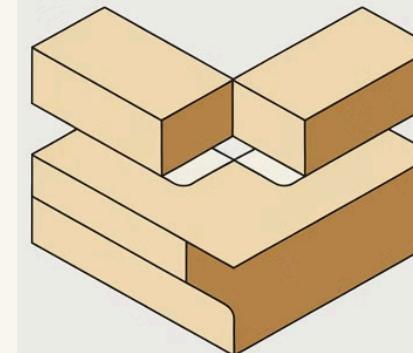
# Arquitecturas de Software más Relevantes



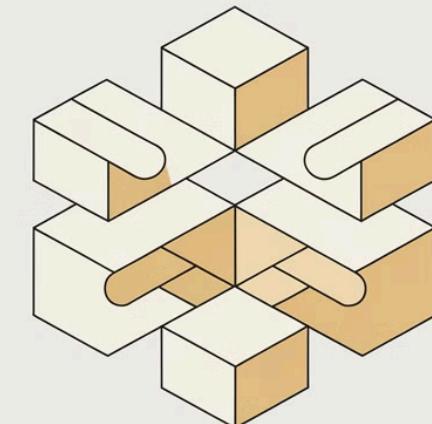
MICROSERVICES



MONOLITHIC



LAYERED



CLIENT-SERVER

# Arquitectura Monolítica

## Desarrollo Inicial

Facilita el desarrollo inicial y el despliegue de una aplicación.

## Mantenimiento

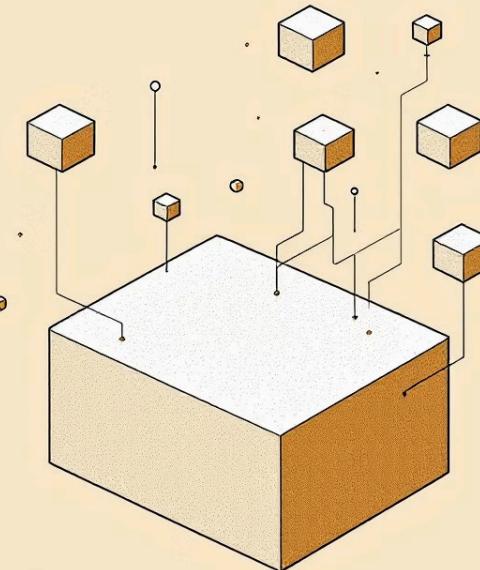
Puede complicar el mantenimiento y la escalabilidad a medida que la aplicación crece.

## Acoplamiento

Las funcionalidades están acopladas en un solo bloque de código.

## Recursos

Todos los componentes comparten el mismo espacio de memoria y recursos.



# Arquitectura Cliente-Servidor

## 1 Elementos

Se compone de un cliente, que solicita servicios, y un servidor, que los proporciona.

## 2 Descripción

Este modelo divide las tareas entre proveedores de recursos o servicios y solicitantes. Es común en aplicaciones web donde el cliente puede ser un navegador y el servidor aloja la lógica de negocio y la base de datos.



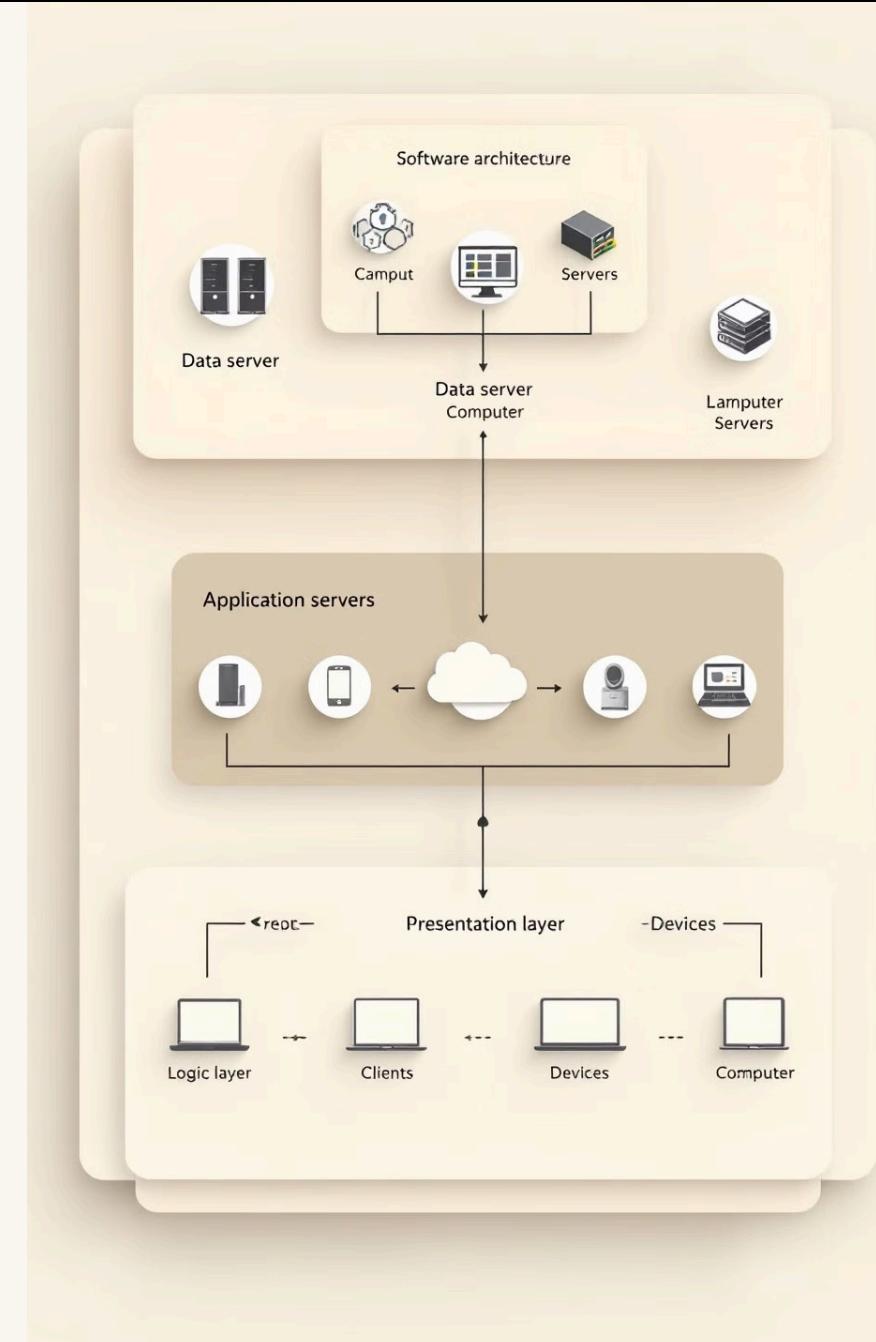
# Arquitectura en Capas

## Elementos

La arquitectura en capas incluye capas como presentación, lógica de negocio y acceso a datos.

## Descripción

Organiza el software en capas, donde cada una tiene una responsabilidad específica y solo interactúa con la capa adyacente. Promueve la modularidad y facilita el mantenimiento.



# Modelo-Vista-Controlador (MVC)

Este patrón permite separar las preocupaciones, facilitando el desarrollo colaborativo y la reutilización del código, siendo ampliamente utilizado en aplicaciones web.



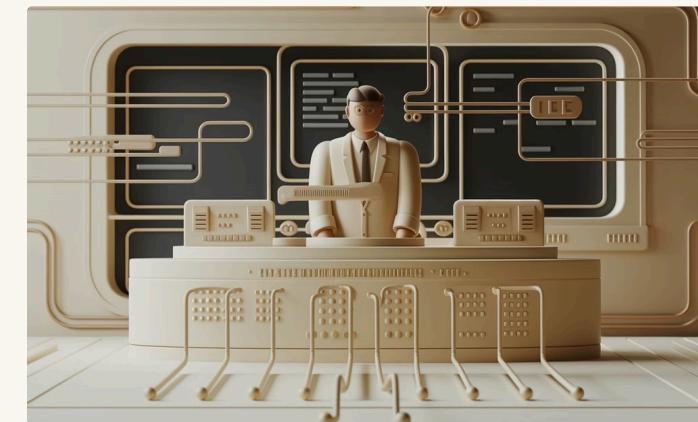
**Modelo**

Gestiona los datos y la lógica de negocio.



**Vista**

Presenta la interfaz de usuario, mostrando la información del modelo.



**Controlador**

Coordina la interacción entre el modelo y la vista, respondiendo a eventos del usuario.



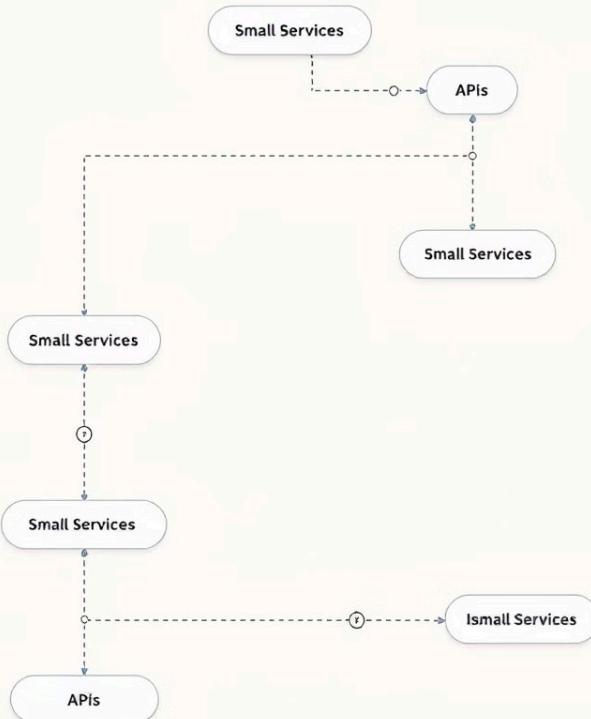
# Arquitectura Orientada a Servicios (SOA)

## Elementos

Servicios independientes que interactúan a través de interfaces bien definidas.

## Descripción

Permite la creación de aplicaciones a partir de servicios reutilizables, promoviendo la interoperabilidad y flexibilidad en el desarrollo.



# Arquitectura de Microservicios

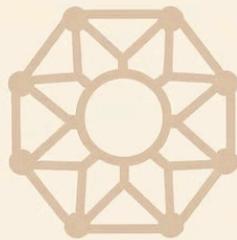
## Elementos

Conjunto de microservicios pequeños, independientes que se comunican a través de APIs.

## Descripción

Facilita el desarrollo ágil y escalable, al permitir que cada microservicio se implemente, escale y mantenga de manera independiente.

# Arquitectura Hexagonal (Puertos y Adaptadores)



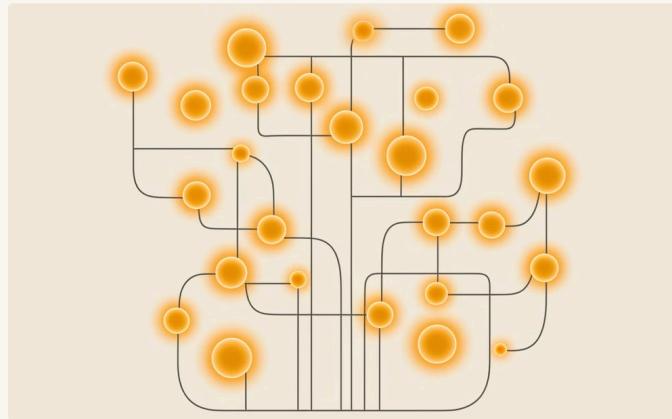
## Elementos

Núcleo de negocio central rodeado por adaptadores que permiten la comunicación con diferentes interfaces externas.

## Descripción

Separa la lógica del negocio del código específico de infraestructura, permitiendo cambios en las tecnologías externas sin afectar al núcleo.

# Arquitectura Basada en Eventos



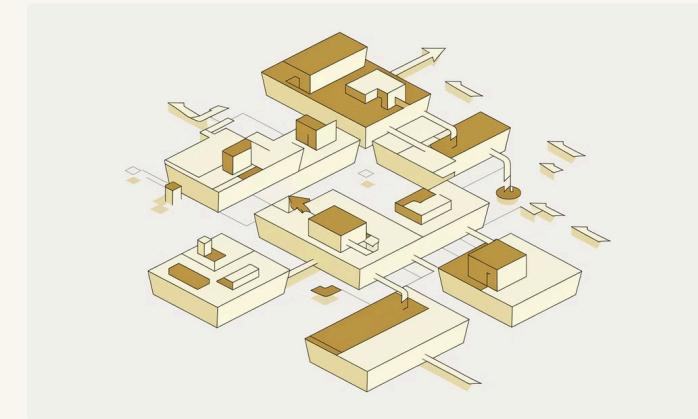
## Sistema de Eventos

Los componentes del sistema se comunican a través de eventos, actuando como publicadores y suscriptores de información.



## Reacción en Tiempo Real

El sistema responde dinámicamente a los eventos según ocurren, permitiendo actualizaciones inmediatas.



## Diseño Desacoplado

La arquitectura permite que los componentes evolucionen de forma independiente, manteniéndose conectados solo a través de eventos.

# Ventajas y Desventajas de Tipos de Arquitectura

Tipo de Arquitectura	Ventajas	Desventajas
Cliente-Servidor	<ul style="list-style-type: none"><li>- Centralización de datos</li><li>- Fácil mantenimiento</li><li>- Recuperación de datos eficiente</li></ul>	<ul style="list-style-type: none"><li>- Vulnerabilidad a ataques (virus, phishing)</li><li>- Dependencia del servidor; si falla, afecta a todos los clientes.</li></ul>
Arquitectura en Capas	<ul style="list-style-type: none"><li>- Modularidad y separación de preocupaciones</li><li>- Facilita la colaboración entre equipos</li><li>- Mantenimiento simplificado</li></ul>	<ul style="list-style-type: none"><li>- Puede tener un rendimiento menor debido a la comunicación entre capas</li><li>- Modificaciones pueden ser costosas</li></ul>
Modelo-Vista-Controlador (MVC)	<ul style="list-style-type: none"><li>- Separación clara entre lógica, presentación y control</li><li>- Facilita la reutilización de componentes</li></ul>	<ul style="list-style-type: none"><li>- Complejidad en la gestión de interacciones entre componentes</li><li>- Puede requerir más tiempo para implementar cambios.</li></ul>

# Ventajas y Desventajas de Tipos de Arquitectura

Tipo de Arquitectura	Ventajas	Desventajas
Arquitectura Orientada a Servicios (SOA)	- Reutilización de servicios y componentes - Interoperabilidad entre diferentes sistemas	- Puede ser costosa en términos de implementación y mantenimiento - Complejidad en la gestión de servicios
Arquitectura de Microservicios	- Escalabilidad y despliegue independiente de servicios - Mejora en el mantenimiento y agilidad en el desarrollo	- Alto consumo de recursos (memoria y CPU) - Complejidad en la gestión y comunicación entre microservicios
Arquitectura Hexagonal	- Aislamiento del núcleo del negocio de las tecnologías externas - Flexibilidad para cambiar adaptadores sin afectar al núcleo	- Puede ser compleja de implementar inicialmente - Requiere un diseño cuidadoso para evitar sobrecarga
Arquitectura Basada en Eventos	- Desacoplamiento entre componentes, lo que facilita la escalabilidad - Respuesta rápida a eventos en tiempo real	- Dificultad para depurar soluciones asíncronas - Puede ser complicado gestionar la consistencia de datos

*En una arquitectura de software, los niveles de subsistema, servicio y función son conceptos que ayudan a estructurar y organizar el sistema en diferentes capas de abstracción.*

## Nivel de Subsistema



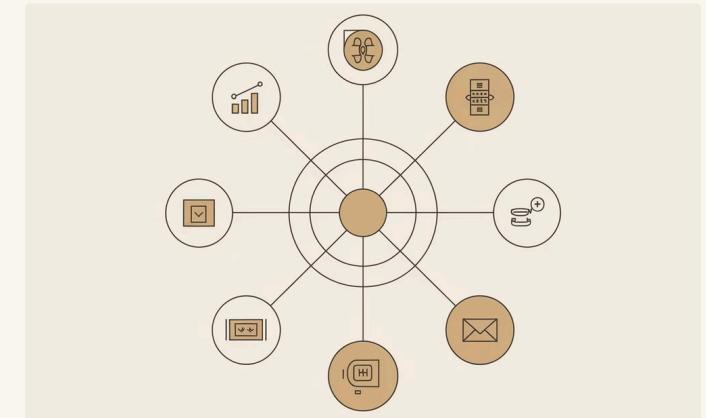
### Subsistema

Un subsistema es una parte de un sistema más grande que puede funcionar de manera independiente o como parte de un sistema más complejo.



### Componentes

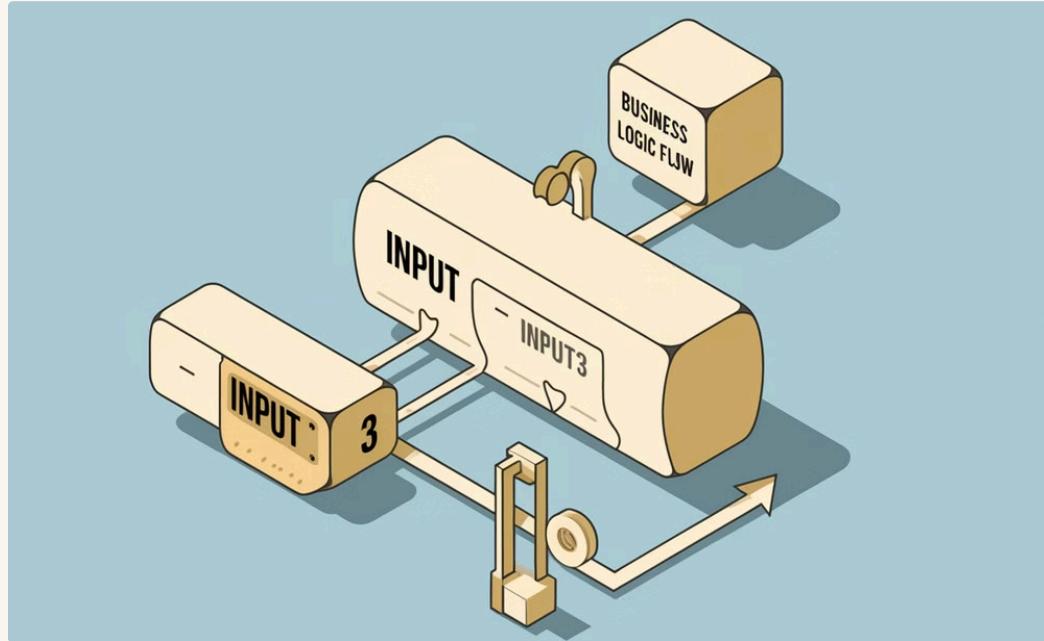
Los componentes son módulos o clases que implementan la lógica del subsistema. Cada uno tiene una función específica dentro del sistema general.



### Interfaces y Datos

Las interfaces son puntos de interacción con otros subsistemas, mientras que los datos son las estructuras que almacenan la información relevante para el funcionamiento del subsistema.

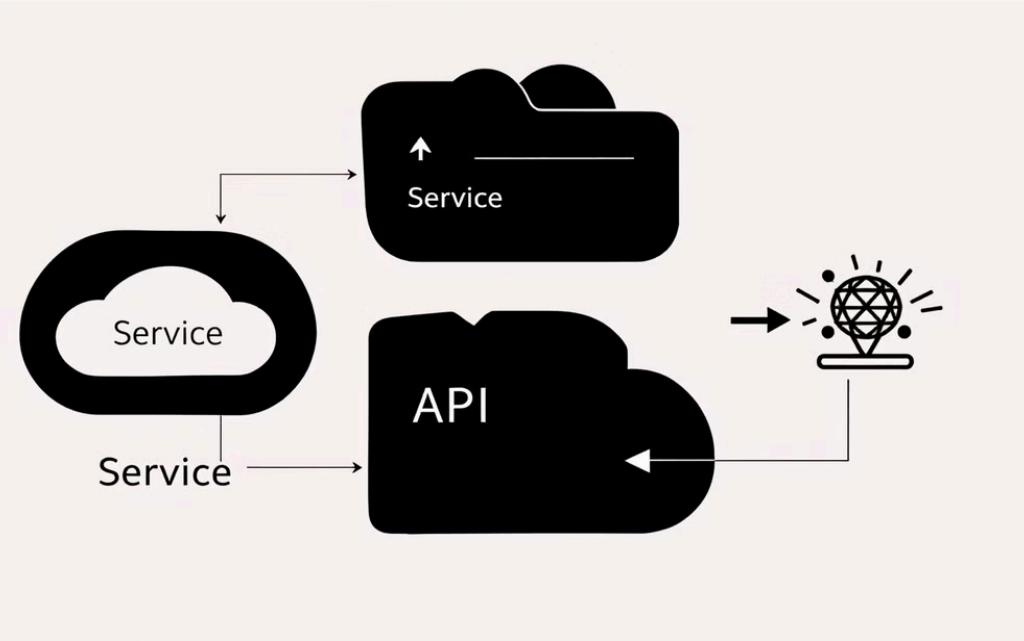
# Nivel de Función y Nivel de Servicio



## Nivel de Función

Se centra en acciones específicas del sistema.

- Métodos
- Parámetros
- Lógica Empresarial



## Nivel de Servicio

Interfaces que exponen funcionalidad.

- Servicios Web
- Protocolos de Comunicación
- Contratos de Servicio

# Tipos de Dependencias



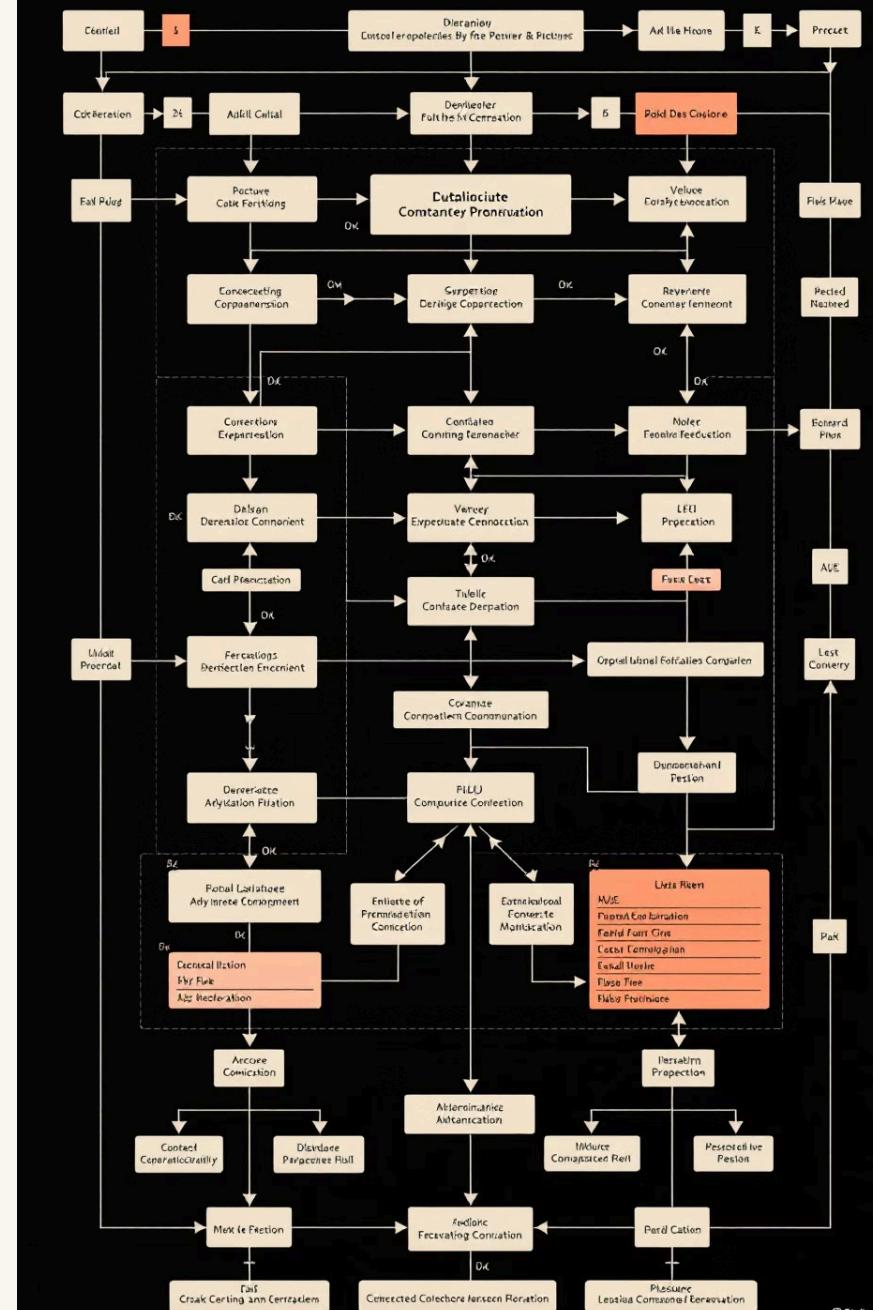
## Dependencias Verticales

Relaciones entre diferentes tipos de componentes, como servicios que dependen de aplicaciones específicas.



## Dependencias Horizontales

Relaciones entre componentes similares, como aplicaciones que interactúan entre sí.



# Manejo de Dependencias

## Correlación de Dependencias

Identificar y visualizar las relaciones entre los componentes del sistema.

## Automatización y Herramientas

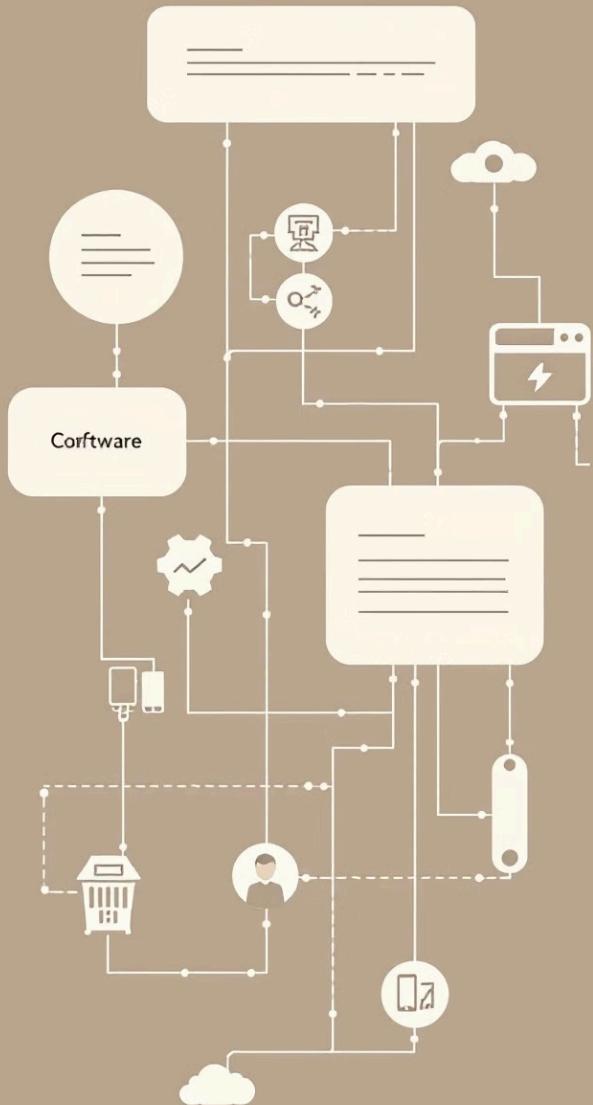
Herramientas que permiten visualización dinámica, detección de problemas y gestión de las dependencias.

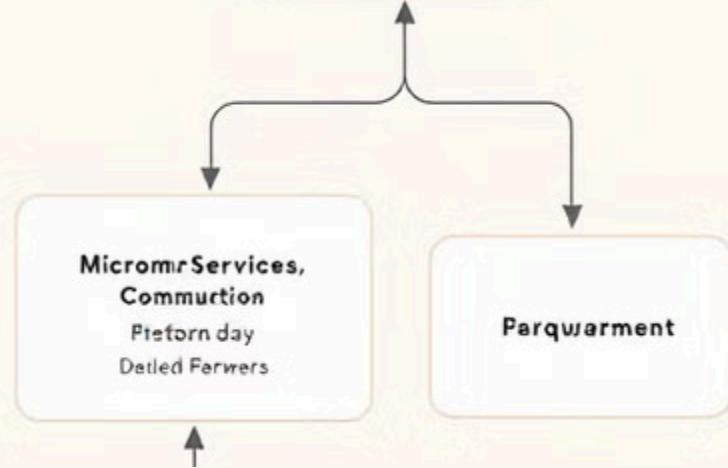
## Pruebas de Integración

Garantizan que los componentes que dependen unos de otros funcionen correctamente.

## Documentación de Dependencias

Mantener registros detallados de las relaciones entre los componentes para facilitar el mantenimiento y la resolución de problemas.





When can fidlin worn!

# Microservicios y Dependencias

## 1 Servicios Independientes

Cada microservicio opera de forma autónoma y se comunica a través de APIs, lo que reduce el acoplamiento entre componentes.

## 2 Gestión de Comunicaciones

Aunque los microservicios son independientes, la comunicación entre ellos puede generar nuevas dependencias que deben ser gestionadas cuidadosamente.

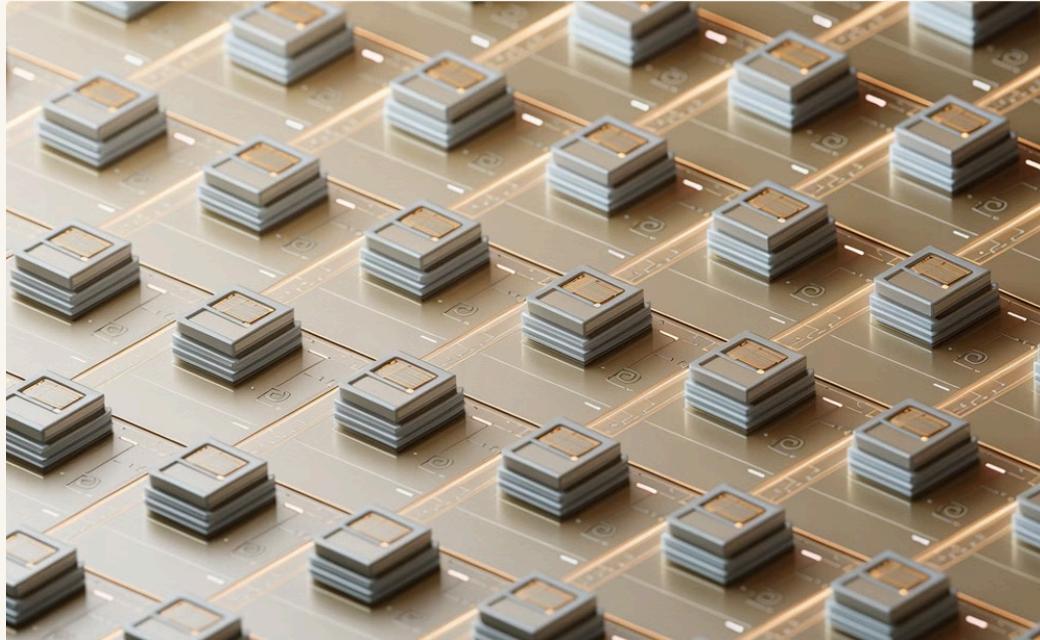
## 3 Descomposición Modular

Dividir sistemas grandes en módulos más pequeños y manejables facilita el desarrollo y la gestión de dependencias.

## 4 Interfaz Bien Definida

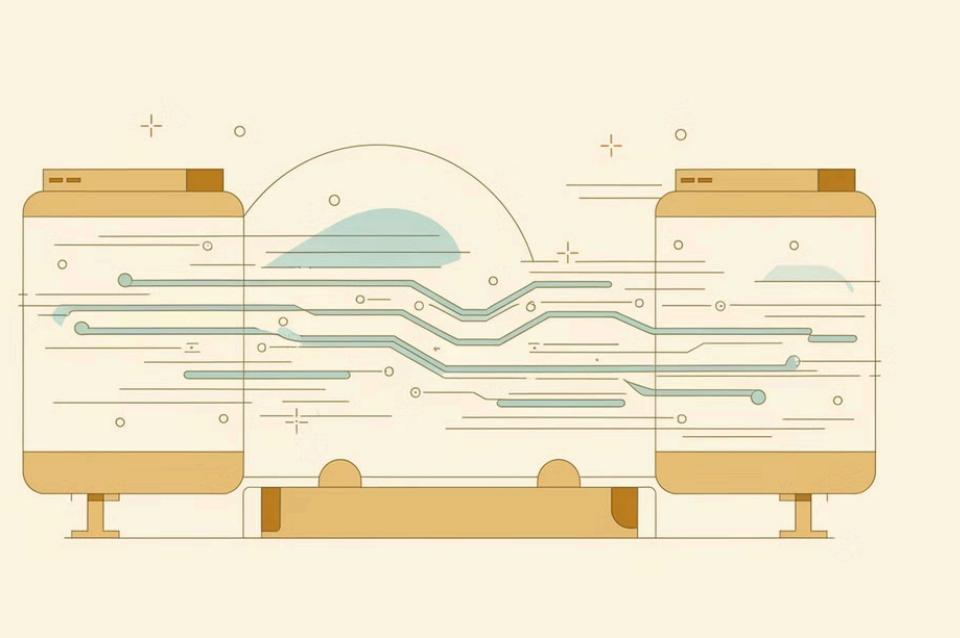
Establecer contratos claros para las interacciones entre componentes permite cambios sin afectar otras partes del sistema.

# Integración de Componentes



## Conexión de Módulos

La integración de componentes se refiere a la forma en que diferentes partes de un sistema de software se conectan e interactúan entre sí, creando una red cohesiva de funcionalidad.



## Flujo de Datos

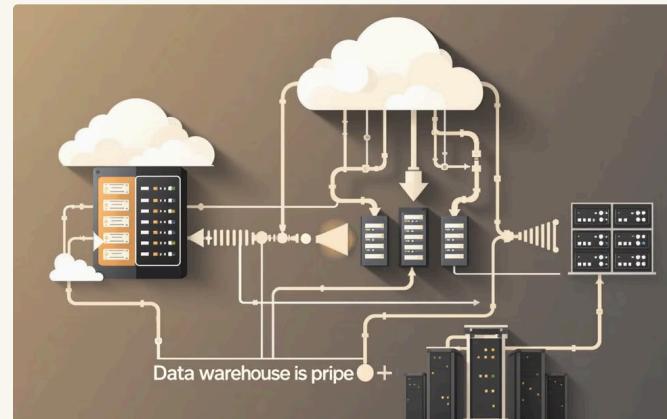
Es esencial para que los componentes compartan información y funcionen de forma coordinada, permitiendo una comunicación efectiva entre las diferentes partes del sistema.

# Integración Batch



## Procesamiento de datos por lotes

Este enfoque recopila y procesa datos en grupos a intervalos programados, en lugar de hacerlo en tiempo real.



## Eficiencia en el procesamiento

Ideal para grandes volúmenes de datos, ya que reduce la carga en los sistemas y permite un procesamiento eficiente.



## Retraso en la información

No proporciona información en tiempo real, lo que puede ser un inconveniente para aplicaciones donde la actualización de datos es crucial.



# Integración Spaghetti



## Implementación Rápida

Puede ser rápida de implementar inicialmente si se necesita una solución inmediata.



## Complejidad y Dificultad

Alta complejidad y dificultad para mantener el sistema.



## Riesgo de Errores

Aumenta el riesgo de errores y fallos debido a la falta de documentación y estructura.



# Integración en Tiempo Real

## Descripción

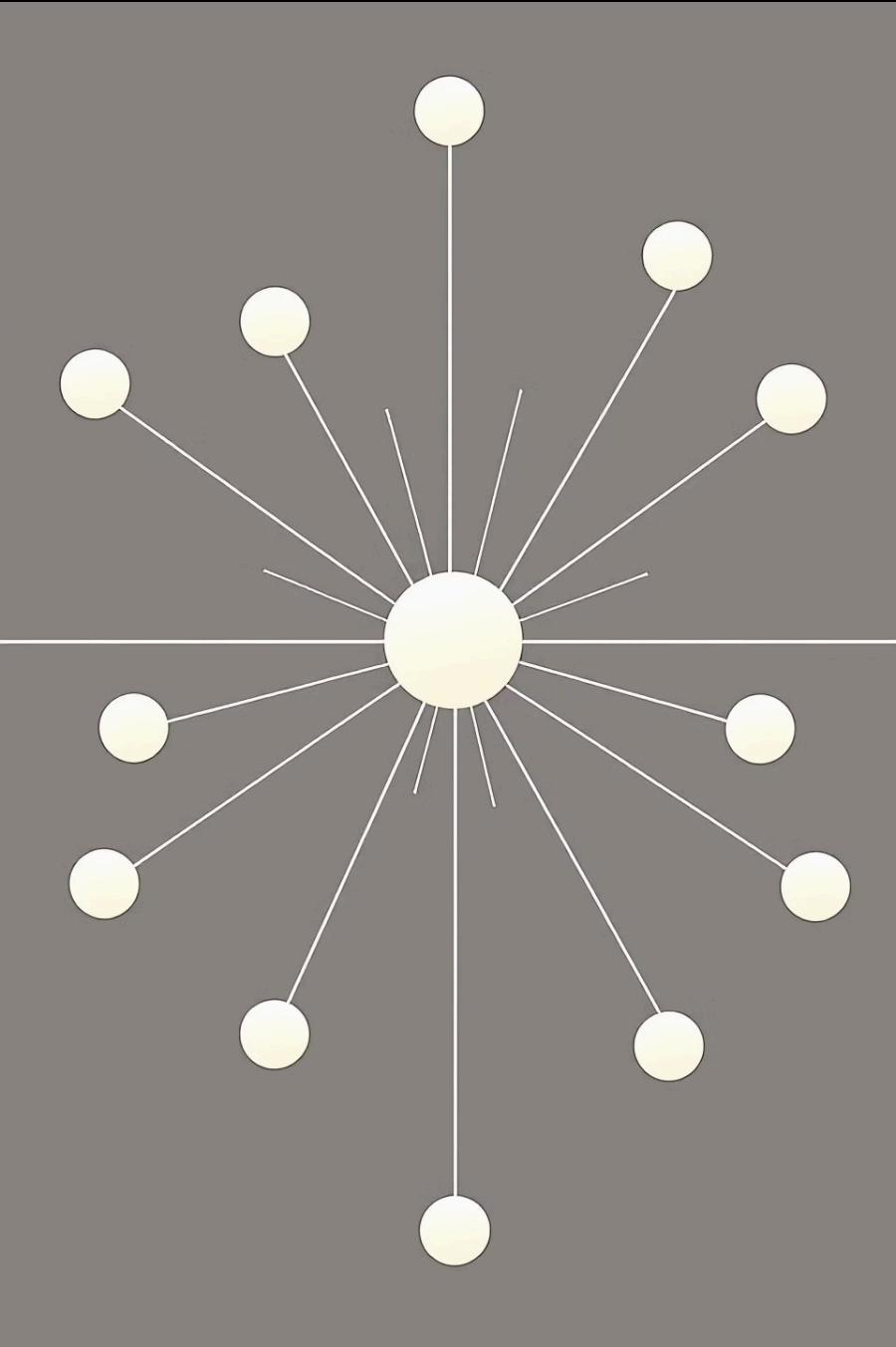
Este enfoque permite la interacción instantánea de los componentes del sistema, procesando datos a medida que llegan. Es crucial para aplicaciones que requieren respuestas inmediatas.

## Ventajas

Proporciona datos actualizados y respuestas inmediatas, esencial para sistemas críticos como el control industrial o financiero.

## Desventajas

Requiere una infraestructura compleja y robusta para manejar la carga constante de datos. Puede ser más costoso debido a los requisitos tecnológicos avanzados.



# Integración Hub-and-Spoke

1

## Descripción

Este modelo utiliza un "hub" central (nodo) para gestionar las conexiones entre varios "spokes" (nodos periféricos).

2

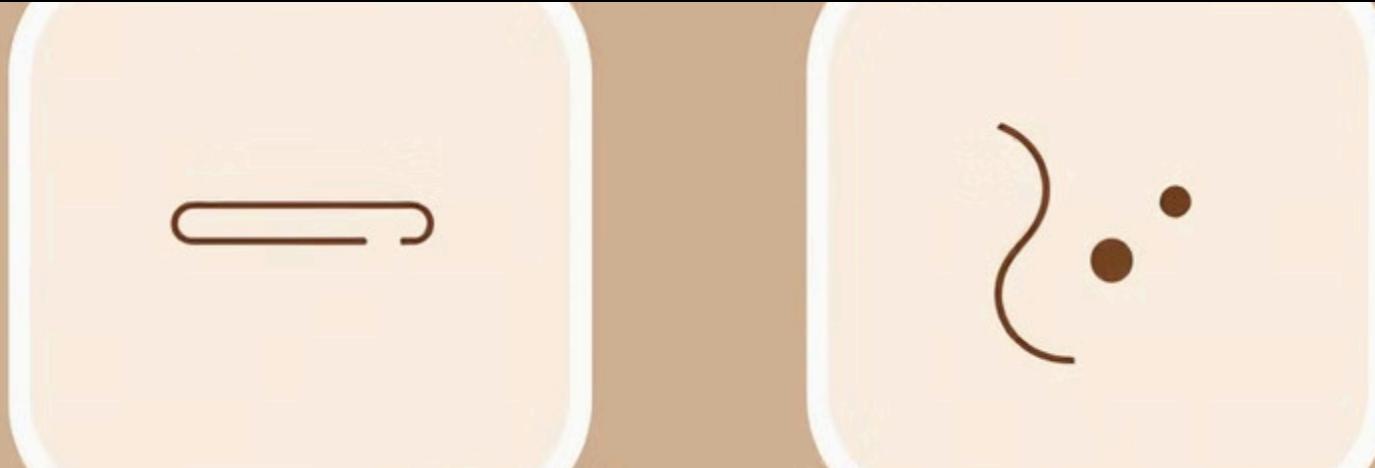
## Ventajas

Simplifica la gestión de conexiones al centralizar el control en un único punto.

3

## Desventajas

Si el hub falla, puede afectar a todos los spokes conectados, creando un punto único de fallo.



# Integración Punto a Punto

## 1 Descripción

Conecta dos aplicaciones para compartir información directamente.

## 2 Ventajas

Sencillez en la implementación.

## 3 Desventajas

Dificultad para escalar y mantener al añadir nuevas aplicaciones.

# Plataforma de Integración como Servicio (iPaaS)



Proporciona una solución basada en la nube que permite conectar aplicaciones y datos a través de conectores y flujos de trabajo prediseñados.

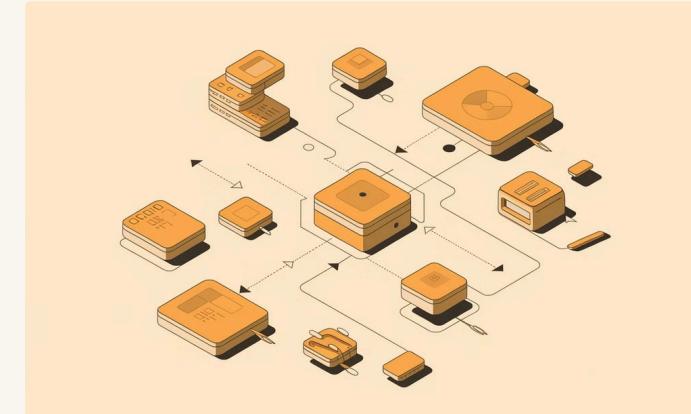
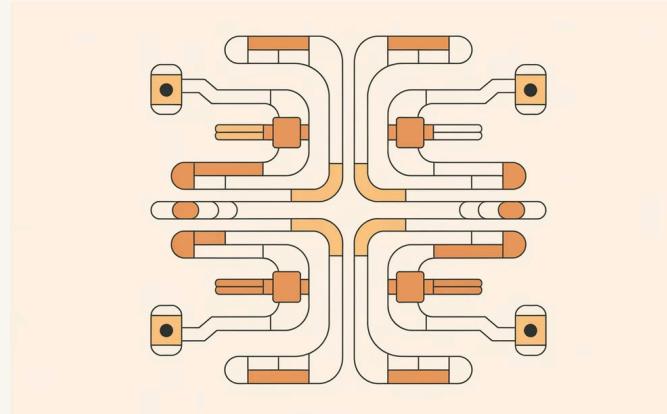


Costos más bajos y tiempos de despliegue rápidos.



Dependencia del proveedor para la seguridad y fiabilidad.

# Bus de Servicios Empresariales (ESB)



## Descripción

Un ESB actúa como intermediario, facilitando la comunicación entre diferentes aplicaciones. Transforma formatos de datos y enruta mensajes, mejorando la interoperabilidad entre sistemas.

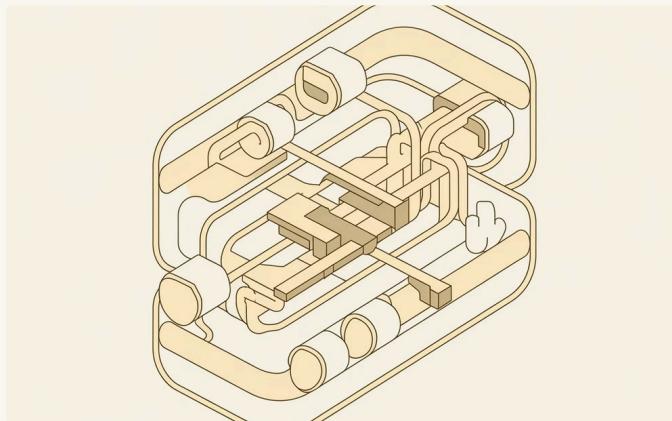
## Ventajas

Un ESB simplifica la integración entre sistemas heterogéneos, mejorando la flexibilidad y la escalabilidad.

## Desventajas

La complejidad de la implementación y el mantenimiento del ESB pueden ser un desafío, especialmente en entornos grandes y complejos.

# Agentes de Mensajes

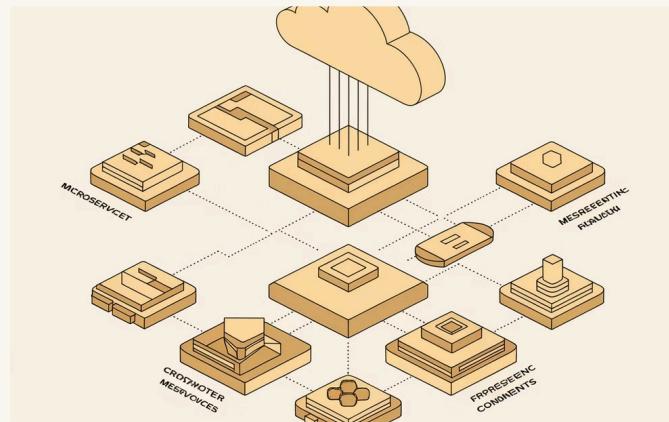


## ¿Qué son?

Facilitan la comunicación asincrónica entre aplicaciones mediante colas de mensajes.

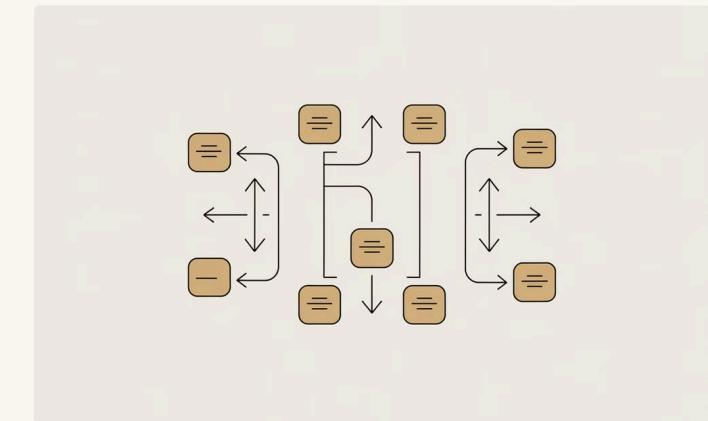
## Beneficios

Permiten arquitecturas desacopladas y escalables.



## Retos

Puede complicar la gestión del flujo de mensajes.





## Desarrollo a Medida



Crear soluciones personalizadas para integrar sistemas específicos.



Adaptación exacta a las necesidades de la organización.



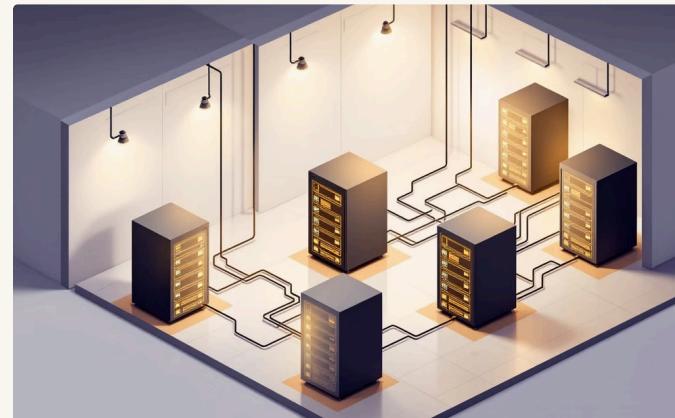
Puede ser costoso y requerir mucho tiempo.

# Patrón de Mensajería y Publicación/Suscripción



## Descripción

Facilita la comunicación entre componentes mediante el intercambio de mensajes o eventos.



## Ventajas

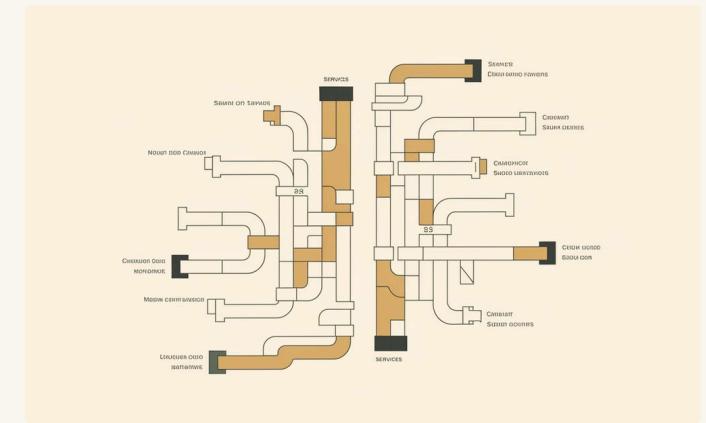
Desacoplamiento y escalabilidad.



## Desventajas

Puede ser complejo gestionar el flujo de eventos.

# Actividad de clase 1: (+4 puntos primer 25%)



## 1. Selección de Aplicación

Escoja una app que le guste (redes sociales, Streaming de vídeo o de Música, etc..)

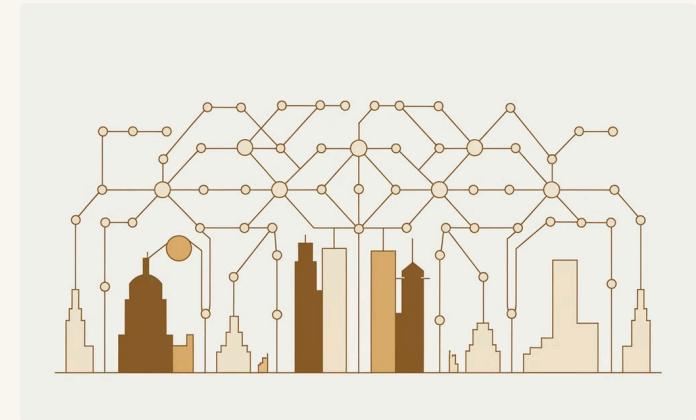
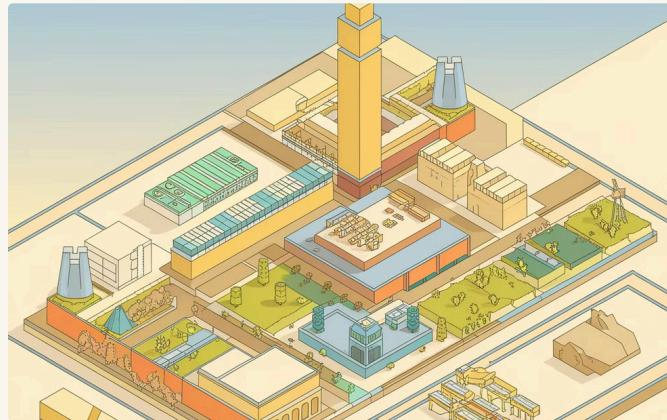
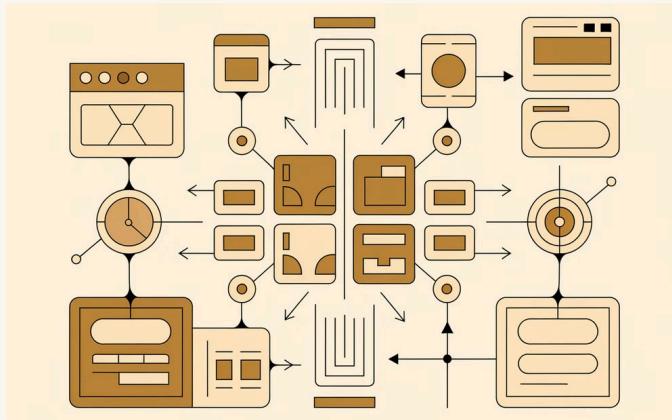
## 2. Investigación

Investigue qué tipo de arquitectura de software utiliza

## 3. Análisis de Componentes

Describa los componentes

# Actividad de clase 2: (+6 puntos primer 25%)



## Selección de Diagrama UML

- Clases
- Despliegue
- Dependencias
- Comunicación
- Componentes
- Paquetes
- Actividad
- Secuencia

## La Ciudad como Sistema

Representar una ciudad como un sistema de software, donde diferentes zonas representan subsistemas (transporte, energía, residencial). Utilizar Legos u otros elementos para construir la ciudad y desarrollar el diagrama UML correspondiente.

## Análisis y Discusión

- Interacción entre subsistemas
- Gestión de fallos y recursos
- Puntos de falla y cuellos de botella
- Resiliencia y tolerancia a fallos