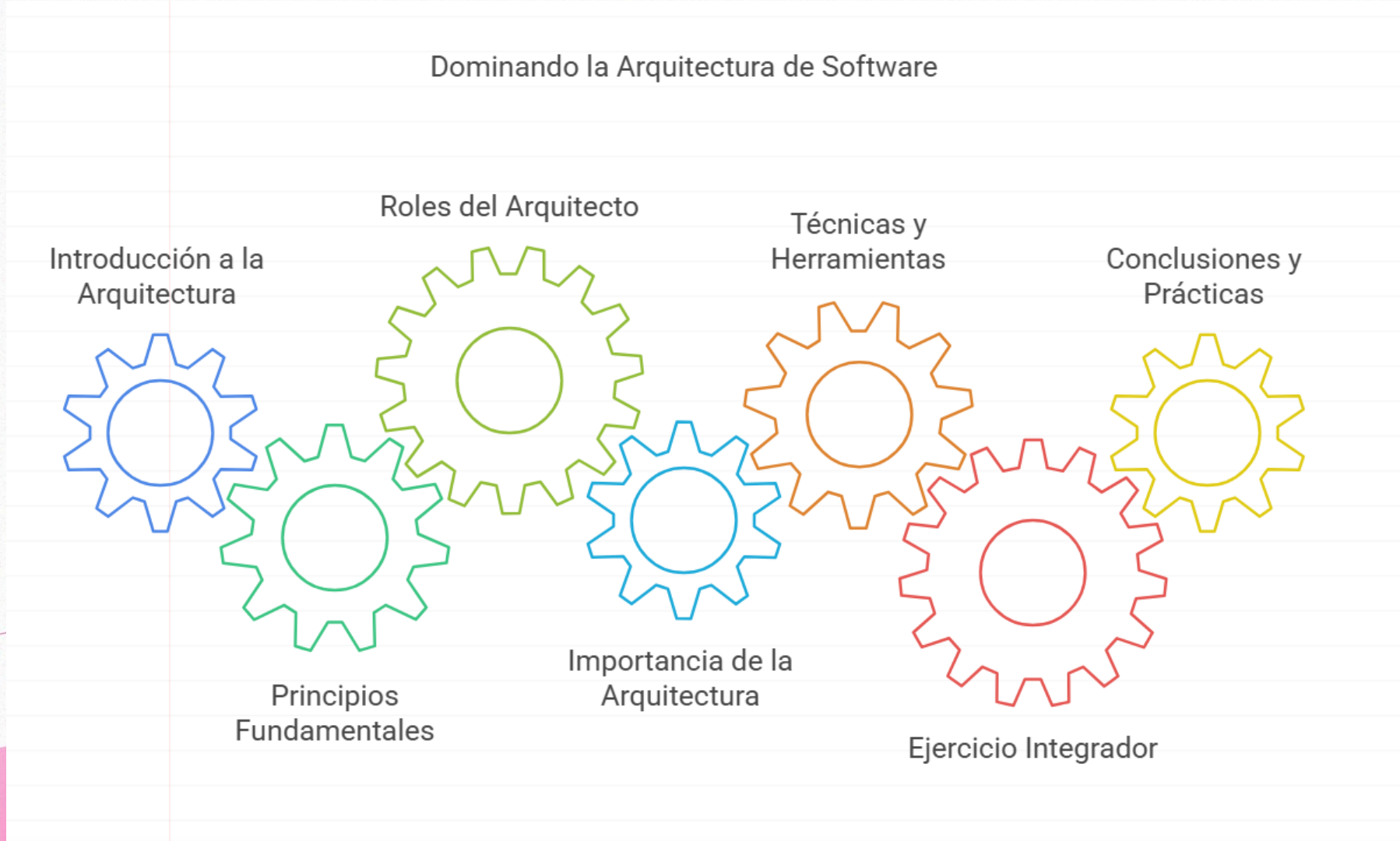
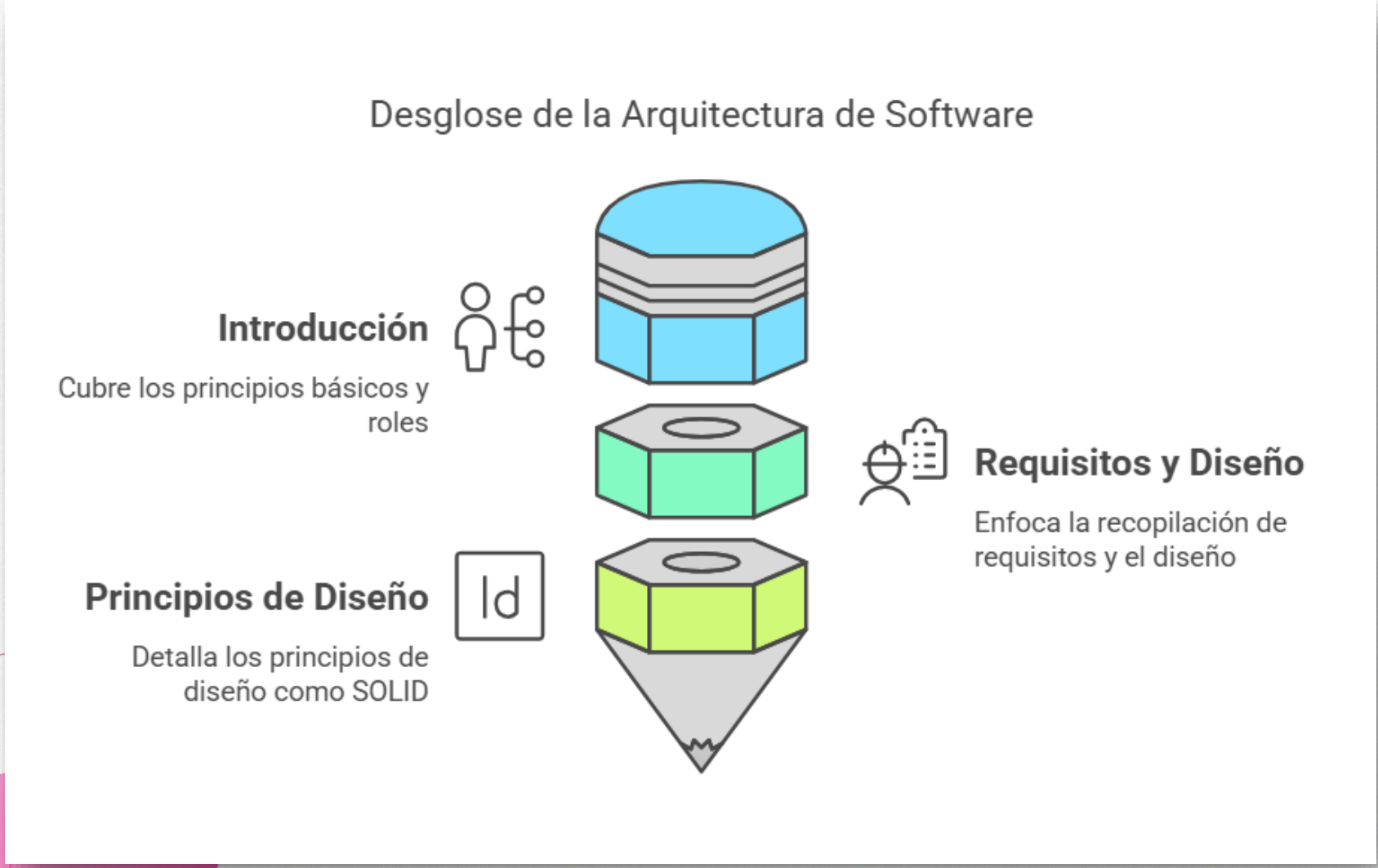
Arquitectura de Software

Arquitectura de Software



Módulo 1: Fundamentos de la Arquitectura de Software

Módulo 1: Fundamentos de la Arquitectura de Software



Módulo 1: Fundamentos de la Arquitectura de Software

1. **Introducción a la Arquitectura de Software**
   Principios básicos de la arquitectura de software.
   Roles y responsabilidades del arquitecto de software.
   Importancia de la arquitectura en el desarrollo de software.
2. **Requisitos y Diseño Arquitectónico**
   Técnicas de recopilación y análisis de requisitos.
   Especificación de casos de uso y diagramas de requisitos.
   Introducción al diseño arquitectónico: vistas y estilos arquitectónicos.
3. **Principios del Diseño de Software**
   Principios del diseño de software.
   Principios SOLID.
   Taller: Implementación de principios SOLID en un proyecto pequeño.

Tema 1 Introducción a la Arquitectura de Software

Introducción a la Arquitectura de Software (1 hora)

- **Objetivo:** Comprender qué es la arquitectura de software, sus elementos clave y su relevancia.
- **Contenido:**
  - Definición y propósito de la arquitectura de software.
  - Diferencias entre arquitectura y diseño.
  - Elementos principales:
    - Componentes
    - Conexiones
    - Estilos arquitectónicos
  - Ejemplos prácticos de arquitecturas conocidas (monolítica, microservicios, etc.).
- **Actividad:** Discusión en grupo: "¿Qué sistemas arquitectónicos conocen y qué desafíos enfrentan?"

1. ¿Qué es la Arquitectura de Software?

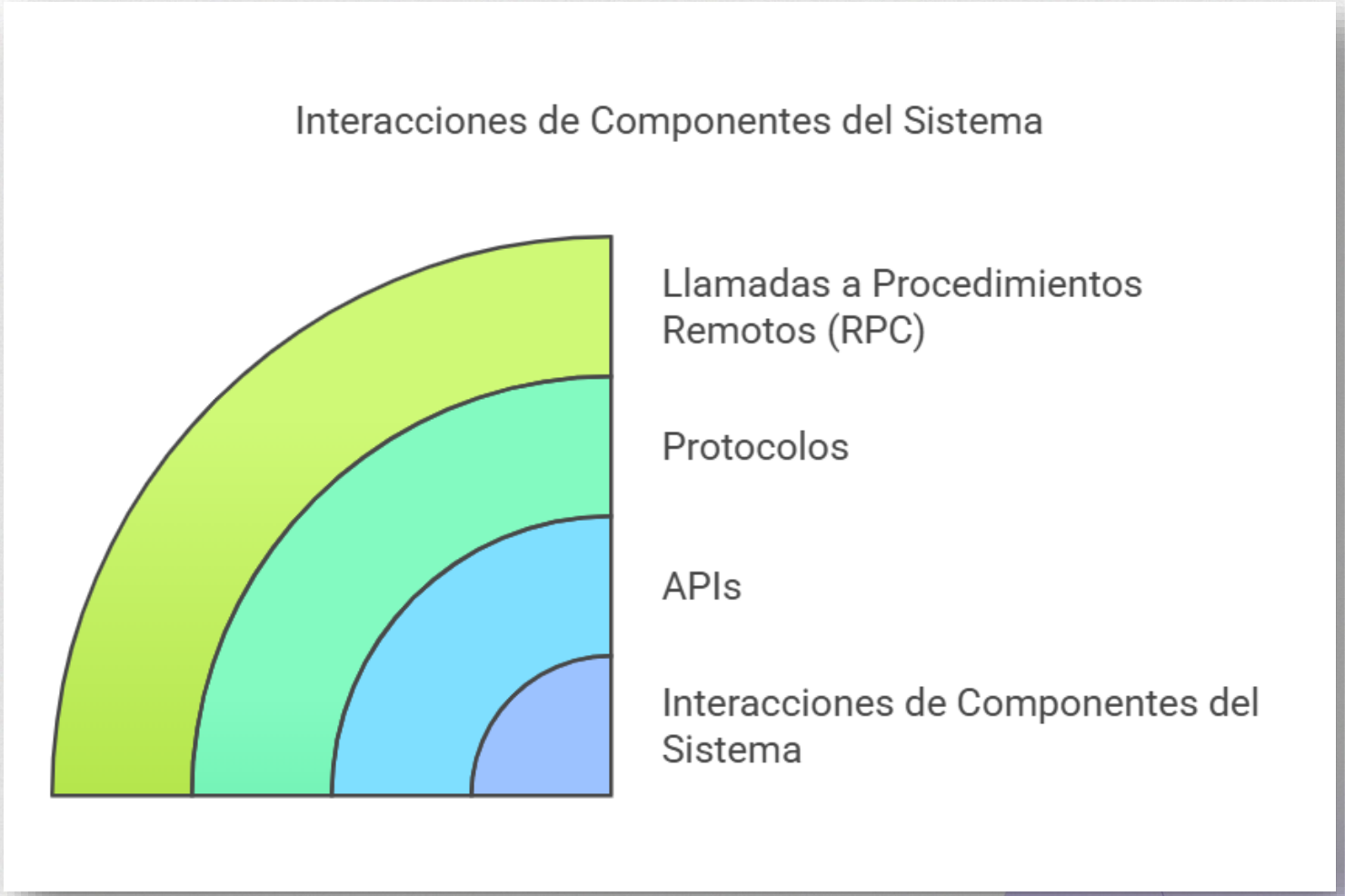
- La arquitectura de software es una disciplina dentro de la ingeniería de software que se centra en el diseño y organización de los sistemas. Define la estructura de los componentes del software, cómo interactúan y las propiedades clave que el sistema debe cumplir.
- **Definición:** Según el *IEEE Standard 1471*, la arquitectura de software es "la organización fundamental de un sistema representada por sus componentes, sus relaciones mutuas y con el entorno, y los principios que guían su diseño y evolución".
- **Propósito principal:**
  - Asegurar que el software cumpla con los requisitos funcionales (qué hace) y no funcionales (cómo lo hace).
  - Guiar a los desarrolladores en la implementación y evolución del sistema.

2. Elementos clave de la Arquitectura de Software

- **Componentes:**
  - Son las partes funcionales del sistema.
  - Pueden ser módulos, servicios, subsistemas.
  - Ejemplo: Servicio de autenticación, motor de pagos.

Conexiones:

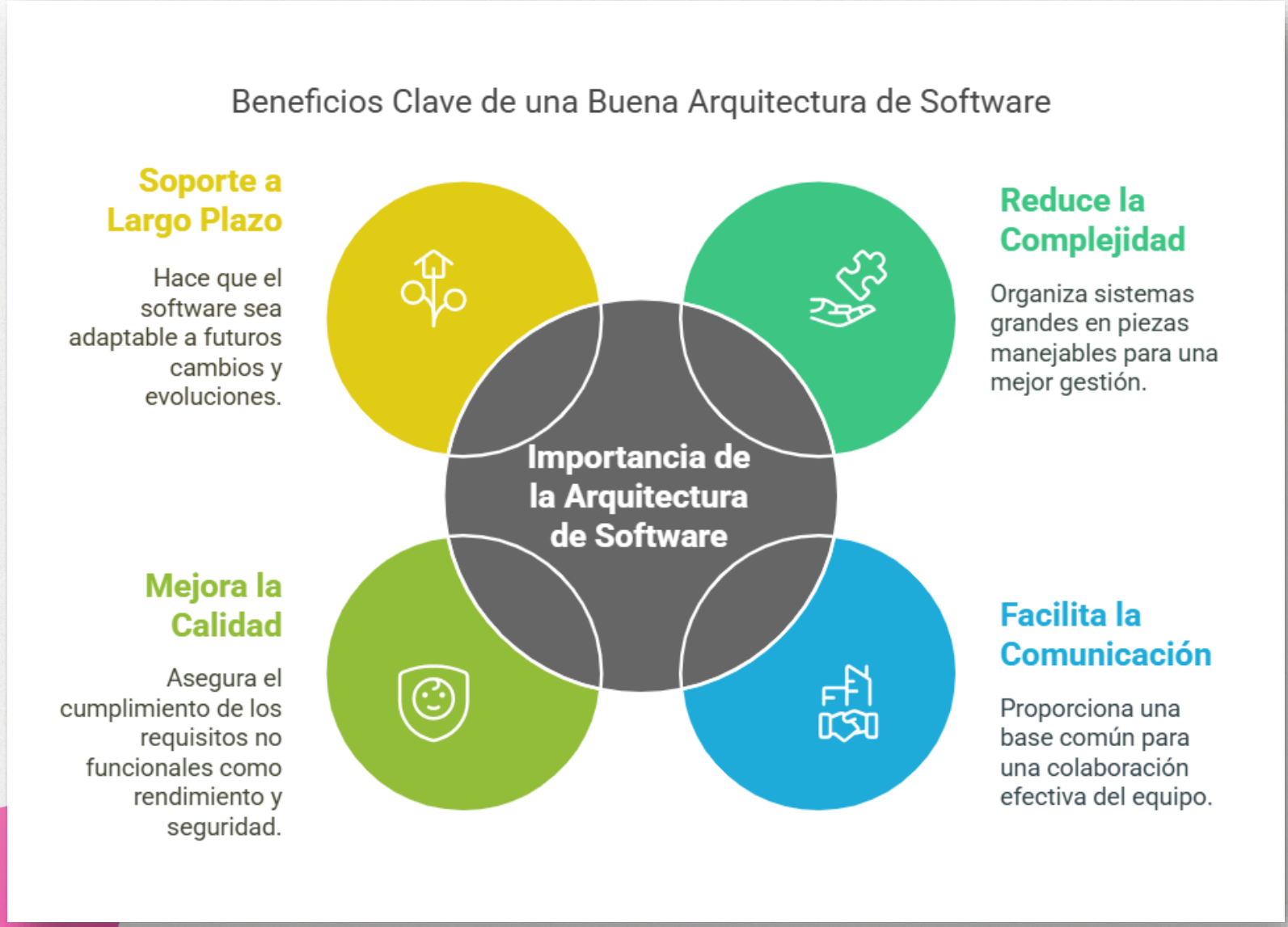
- Definen cómo los componentes interactúan entre sí.
- Tipos: APIs, colas de mensajes, conexiones directas.
- Ejemplo: Un servicio web que expone una API REST.



Estilos arquitectónicos:

- Patrones comunes utilizados para organizar sistemas.
- Ejemplos:
  - Monolítica: Todo el código y la lógica están centralizados en un solo lugar.
  - Microservicios: El sistema se divide en pequeños servicios independientes.
  - Cliente-servidor: Un cliente solicita servicios o datos a un servidor.

3. Importancia de la Arquitectura de Software



4. Diferencias entre Arquitectura y Diseño

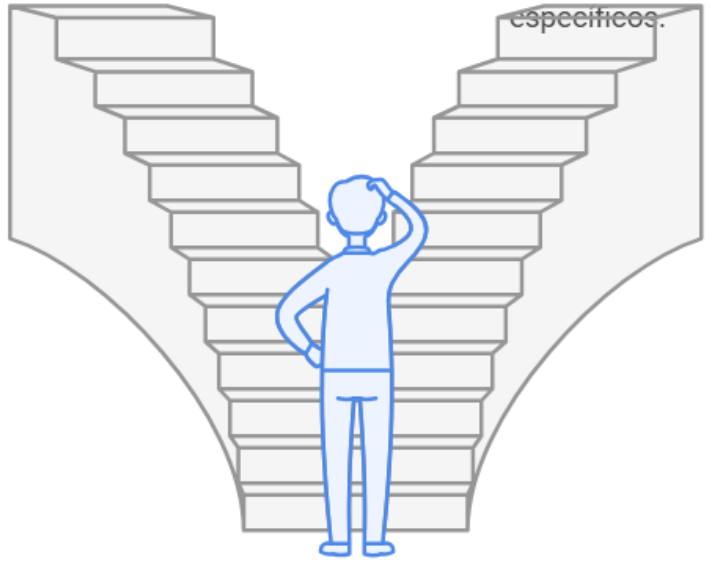
¿Qué enfoque se debe priorizar en el desarrollo del sistema?

Enfoque de Arquitectura

Se centra en la estructura general y los principios del sistema, abordando la interacción de módulos y estilos arquitectónicos.

Enfoque de Diseño

Se centra en la implementación de componentes individuales, detallando algoritmos y módulos específicos.



Diferencias entre Arquitectura y Diseño

Aspecto	Arquitectura	Diseño
Nivel de abstracción	Alto nivel, vista global	Detalle técnico de componentes específicos
Enfoque	Estructura general y relaciones	Implementación interna de cada componente
Ejemplos	Elección de microservicios, monolito, etc.	Estructura de clases, patrones de diseño
Quién lo define	Arquitecto de software, equipo técnico	Desarrolladores, equipo de implementación

5. Ejemplos de Arquitectura

- **Arquitectura Monolítica:**
  - Ideal para aplicaciones pequeñas.
  - Fácil de desarrollar inicialmente, pero difícil de escalar y mantener.
- **Arquitectura de Microservicios:**
  - Permite una alta escalabilidad.
  - Cada servicio es independiente y puede desarrollarse en diferentes tecnologías.
- **Arquitectura en Capas:**
  - Divide el sistema en capas lógicas (e.g., presentación, lógica de negocio, datos).
  - Fácil de entender y mantener.

Actividad

- **Discusión en Grupo:**
- Pregunta: "Piensen en una aplicación que usen a diario, como WhatsApp o Amazon. ¿Qué estilo arquitectónico creen que utiliza? ¿Qué ventajas y desventajas ven en esa arquitectura?"
- Tiempo: 15 minutos
- Objetivo: Fomentar el análisis crítico de los participantes y vincular conceptos teóricos con casos reales.

Referencias Bibliográficas

- Bass, L., Clements, P., & Kazman, R. (2021). **Software Architecture in Practice** (4th Edition). Addison-Wesley Professional.
  - Un libro clave que aborda conceptos y prácticas de arquitectura de software con casos de estudio.
- IEEE. (2000). **IEEE Recommended Practice for Architectural Description of Software-Intensive Systems** (IEEE 1471). IEEE Computer Society.
  - Estándar ampliamente reconocido sobre la definición y documentación de arquitectura.
- Fowler, M. (2002). **Patterns of Enterprise Application Architecture**. Addison-Wesley.
  - Referencia para estilos arquitectónicos comunes como capas y microservicios.
- Rozanski, N., & Woods, E. (2011). **Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives**. Addison-Wesley.
  - Profundiza en cómo documentar y comunicar arquitecturas a diferentes audiencias.
- Kruchten, P. (1995). **Architectural Blueprints—The “4+1” View Model of Software Architecture**. IEEE Software.
  - Modelo de vistas arquitectónicas que ayuda a organizar la descripción de un sistema.

Tema 2 Principios Básicos de la Arquitectura de Software

Principios Básicos de la Arquitectura de Software (1.5 horas)

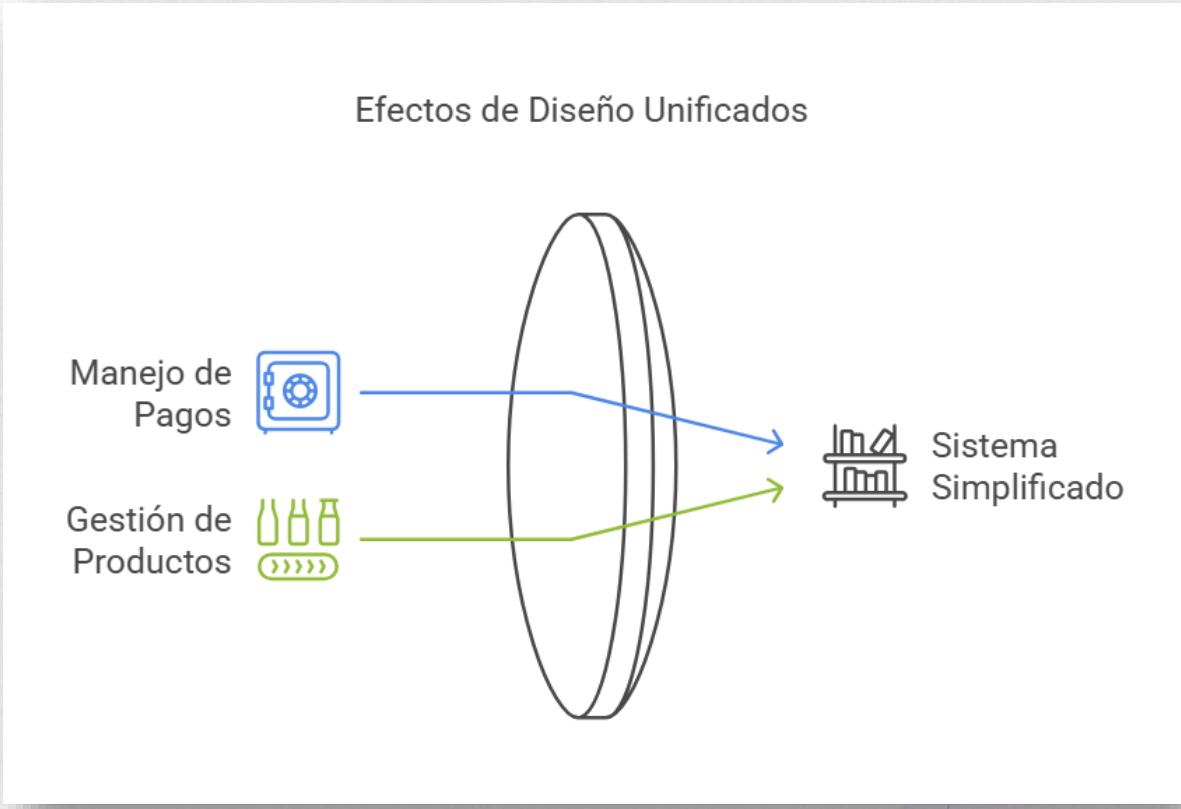
- **Objetivo:** Aprender y aplicar principios fundamentales en la arquitectura de software.
- **Contenido:**
  - **Principio 1:** Separación de preocupaciones.
  - **Principio 2:** Alta cohesión y bajo acoplamiento.
  - **Principio 3:** Modularidad y reutilización.
  - **Principio 4:** Escalabilidad y rendimiento.
  - **Principio 5:** Interoperabilidad y estándares.
- **Actividad:** Taller práctico: Identificar principios aplicados en un sistema sencillo (e.g., sistema de gestión de biblioteca).

1. Introducción a los Principios Básicos

- Los principios básicos de la arquitectura de software son las reglas fundamentales que guían la organización, diseño y evolución de un sistema. Estos principios ayudan a los arquitectos a tomar decisiones informadas, considerando tanto los requisitos actuales como los futuros.
- **Por qué son importantes los principios básicos**
  - Aseguran que el sistema sea escalable, mantenible y adaptable.
  - Reducen la probabilidad de fallos técnicos o problemas de rendimiento.
  - Facilitan la comunicación y colaboración entre los equipos.

2. Principios Fundamentales

- **2.1 Separación de Preocupaciones** (Separation of Concerns)
  - **Definición:** Cada componente o módulo del sistema debe abordar una única responsabilidad o preocupación.
  - **Ejemplo:** En un sistema de comercio electrónico, separar el manejo de pagos del módulo de gestión de productos.
  - **Beneficio:** Facilita el mantenimiento y reduce la complejidad.



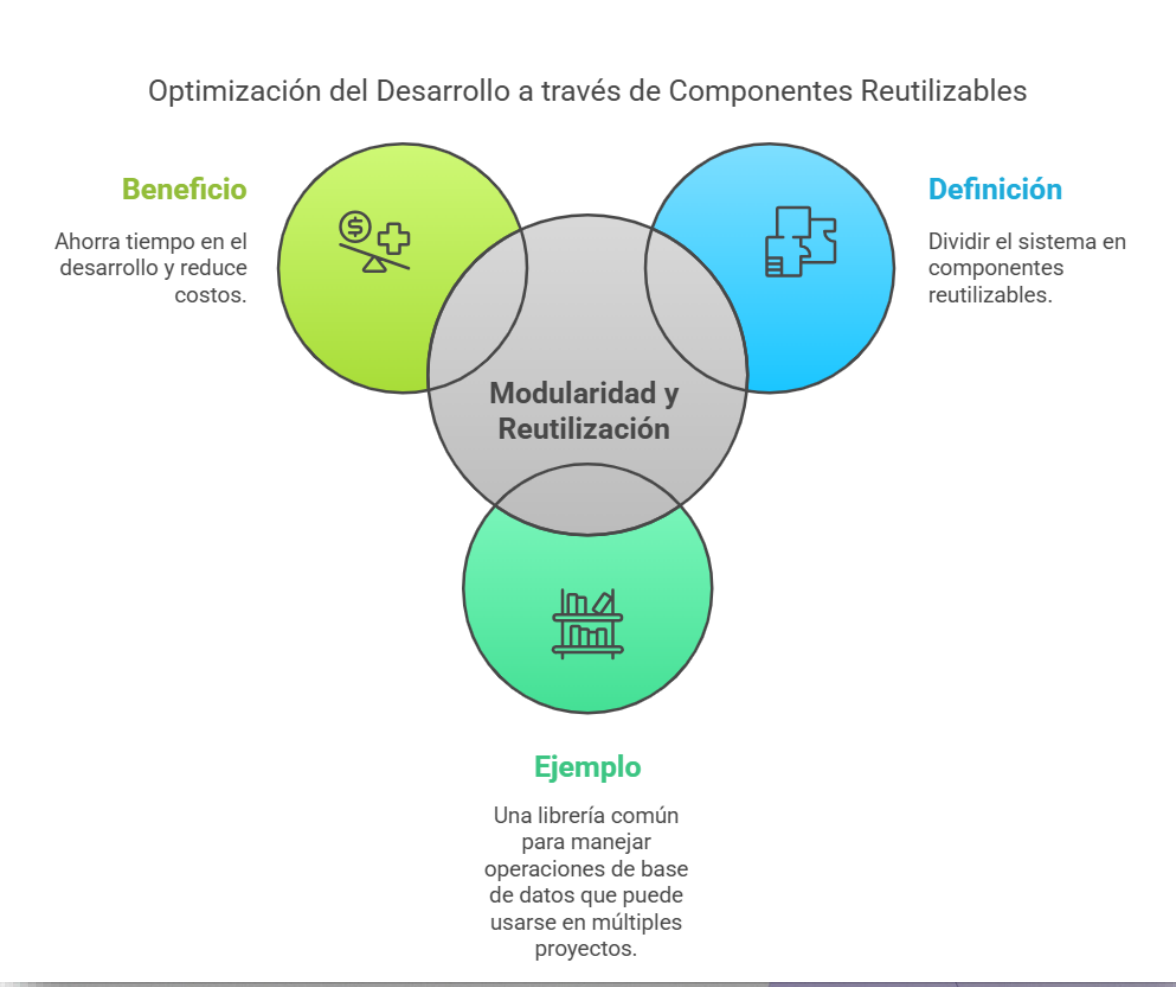
…Principios Fundamentales

- **2.2 Alta Cohesión y Bajo Acoplamiento**
  - **Alta cohesión:** Los elementos dentro de un módulo deben estar estrechamente relacionados.
  - **Bajo acoplamiento:** Los módulos deben ser lo más independientes posible.

  - **Ejemplo:** Un módulo de autenticación que puede operar de forma independiente del módulo de perfil de usuario.
  - **Beneficio:** Mejora la reutilización y facilita los cambios en el sistema.

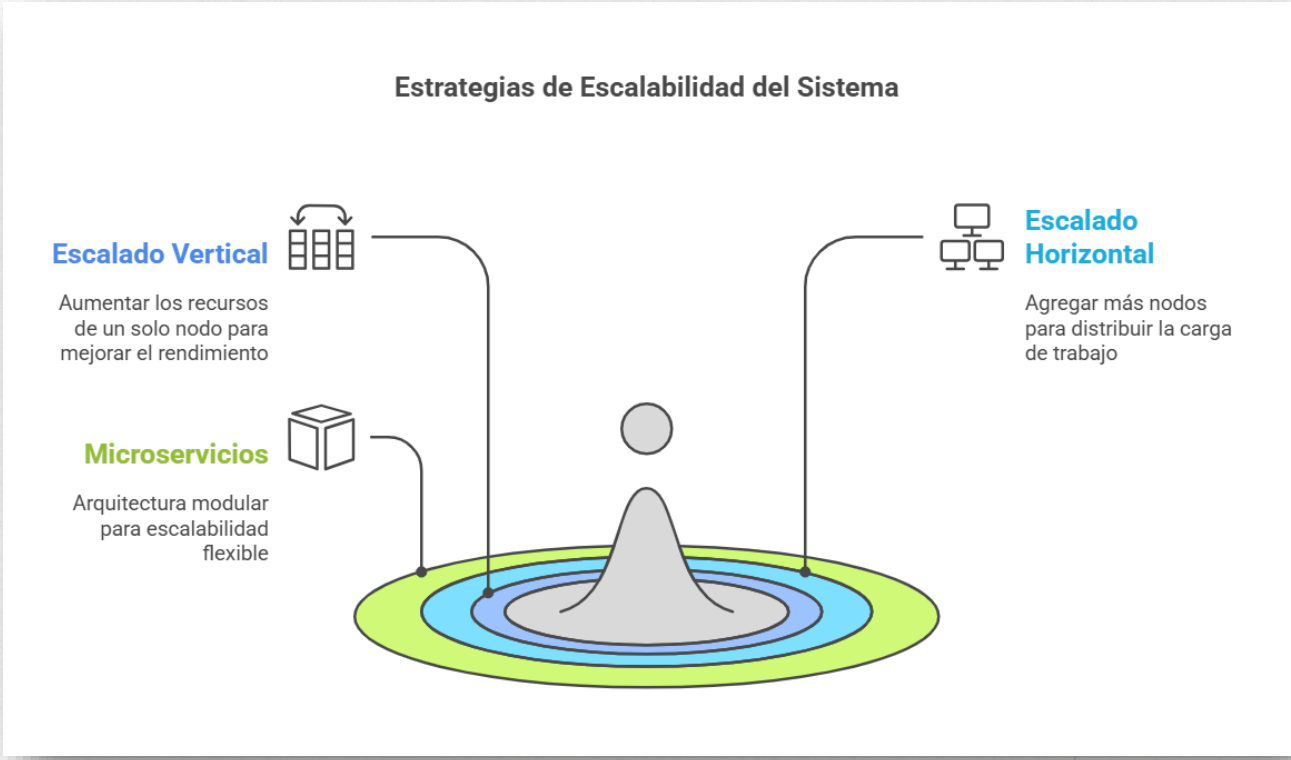
…Principios Fundamentales

- **2.3 Modularidad y Reutilización**
  - **Definición:** Dividir el sistema en componentes reutilizables.
  - **Ejemplo:** Una librería común para manejar operaciones de base de datos que puede usarse en múltiples proyectos.
  - **Beneficio:** Ahorra tiempo en el desarrollo y reduce costos.



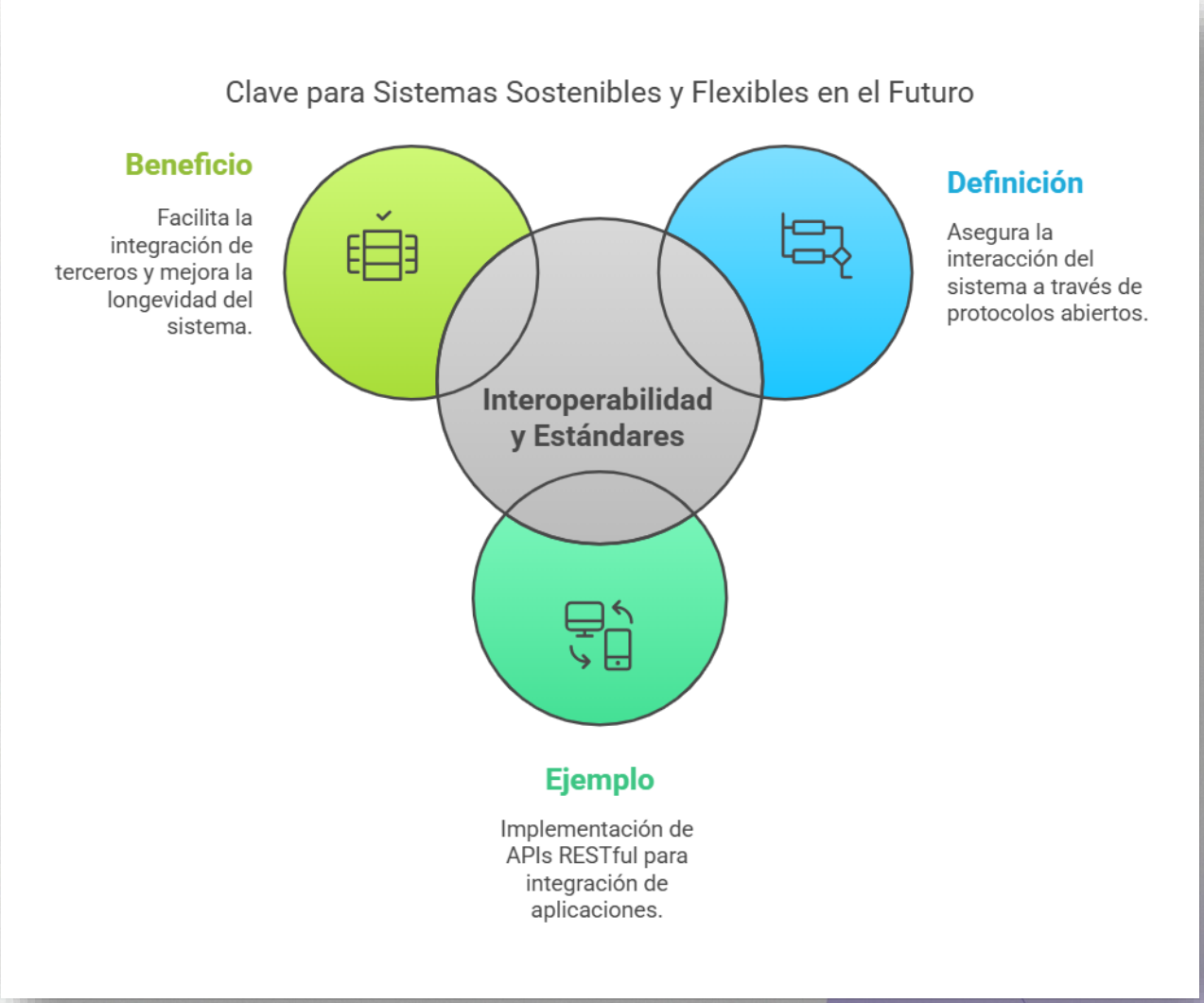
…Principios Fundamentales

- **2.4 Escalabilidad y Rendimiento**
  - **Definición:** Diseñar el sistema para que pueda manejar un aumento en la carga de trabajo sin degradar el rendimiento.
  - **Ejemplo:** Uso de arquitecturas de microservicios para permitir escalado horizontal.
  - **Beneficio:** Permite que el sistema crezca de acuerdo con las necesidades del negocio.



…Principios Fundamentales

- **2.5 Interoperabilidad y Estándares**
  - **Definición:** Asegurar que el sistema pueda interactuar con otros sistemas utilizando protocolos y estándares abiertos.
  - **Ejemplo:** Uso de APIs RESTful para permitir que diferentes aplicaciones se integren.
  - **Beneficio:** Facilita la integración con terceros y mejora la longevidad del sistema.



3. Cómo Aplicar los Principios en la Práctica

- **Evaluar los requisitos del sistema:**
  - Identificar los requisitos funcionales (qué hace el sistema) y no funcionales (cómo debe hacerlo).
- **Definir los límites del sistema:**
  - Determinar qué debe estar dentro de cada componente y cómo interactuarán.
- **Elegir estilos arquitectónicos adecuados:**
  - Seleccionar entre monolítica, microservicios, arquitectura orientada a eventos, etc.
- **Documentar las decisiones arquitectónicas:**
  - Utilizar registros de decisiones arquitectónicas (ADRs) para justificar las elecciones.

4. Ejemplos de Implementación

- **Ejemplo 1: Sistema Bancario**
  - Separación de Preocupaciones: Módulos separados para gestión de cuentas, transacciones y reportes.
  - Escalabilidad: Uso de bases de datos distribuidas para manejar grandes volúmenes de transacciones.
  - Interoperabilidad: APIs para permitir la integración con aplicaciones de terceros.
- **Ejemplo 2: Plataforma de Streaming**
  - Modularidad: Dividir el sistema en módulos de recomendación, gestión de contenido y análisis de usuarios.
  - Alto rendimiento: Uso de caché y balanceo de carga para asegurar tiempos de respuesta rápidos.

5. Actividad

- **Taller práctico:**
- **Objetivo:** Aplicar los principios básicos al diseño de un sistema sencillo.
- **Instrucciones:**
  - Proponer un sistema (e.g., aplicación para reserva de vuelos).
  - Dividirlo en módulos siguiendo los principios de separación de preocupaciones y modularidad.
  - Identificar cómo aplicar escalabilidad e interoperabilidad.
- **Duración:** 45 minutos
- **Resultados esperados:** Un esquema de alto nivel del sistema con principios aplicados.

6. Retos y Buenas Prácticas

- **Retos:**
  - Encontrar el equilibrio entre cohesión y acoplamiento.
  - Priorizar principios cuando hay conflictos (e.g., modularidad vs. rendimiento).
- **Buenas prácticas:**
  - Revisar constantemente la arquitectura frente a los requisitos.
  - Incluir a los stakeholders en las decisiones críticas.

Referencias Bibliográficas

- Bass, L., Clements, P., & Kazman, R. (2021). **Software Architecture in Practice** (4th Edition). Addison-Wesley Professional.
  - Incluye una sección detallada sobre principios arquitectónicos y su aplicación práctica.
- Fowler, M. (2002). **Patterns of Enterprise Application Architecture**. Addison-Wesley.
  - Profundiza en la cohesión, acoplamiento y otros principios relacionados.
- Rozanski, N., & Woods, E. (2011). **Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives**. Addison-Wesley.
  - Enfocado en cómo aplicar principios a sistemas complejos.
- Kruchten, P. (1995). **Architectural Blueprints—The “4+1” View Model of Software Architecture**. IEEE Software.
  - Proporciona un modelo práctico para documentar arquitecturas basadas en principios sólidos.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley.
  - Ayuda a entender cómo los principios arquitectónicos pueden implementarse mediante patrones de diseño.

Tema 3 Roles y Responsabilidades del Arquitecto de Software

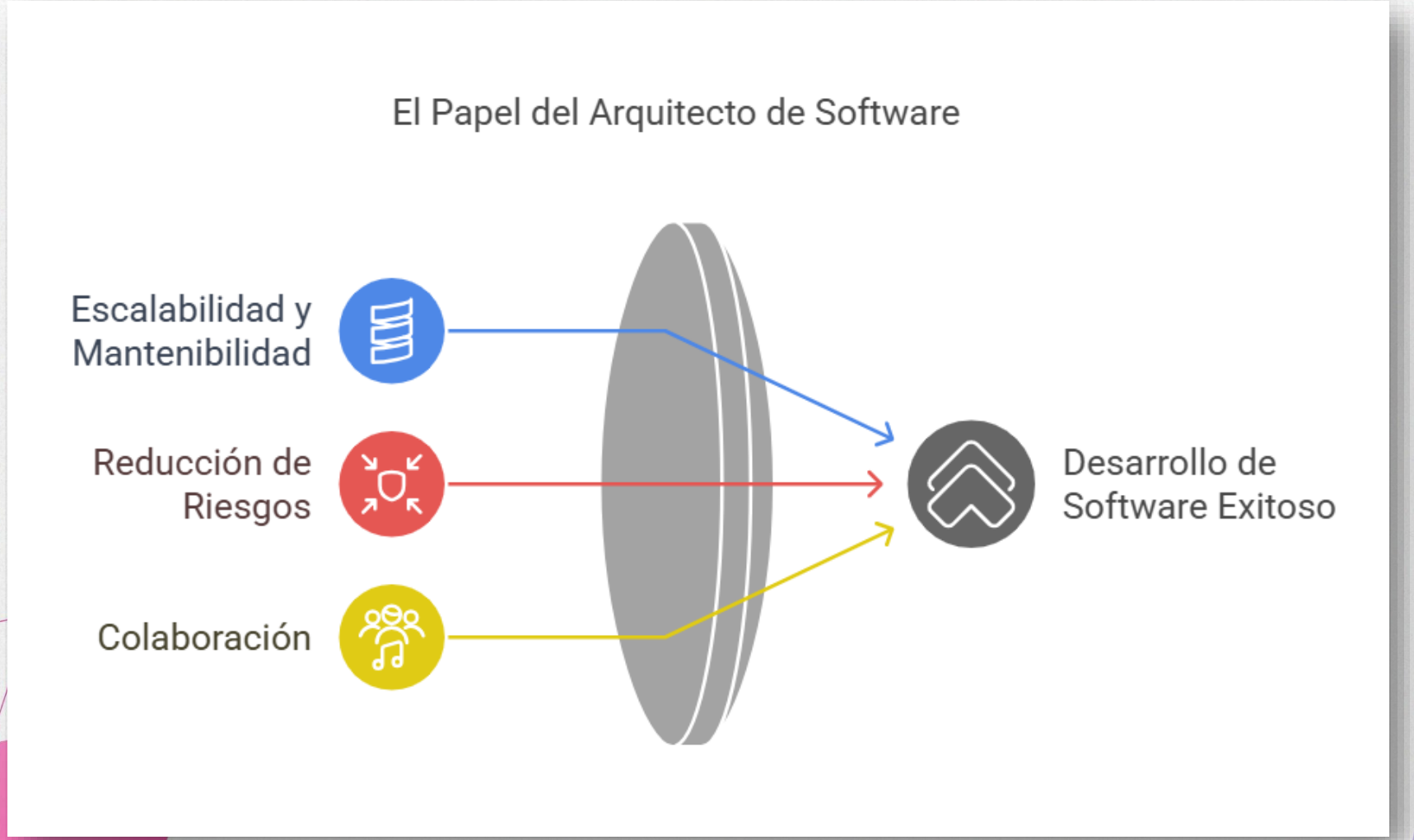
Roles y Responsabilidades del Arquitecto de Software (1.5 horas)

- **Objetivo:** Comprender el papel del arquitecto en equipos de desarrollo y proyectos de software.
- **Contenido:**
  - **Roles clave:**
    - Diseñador técnico.
    - Tomador de decisiones.
    - Mediador entre partes interesadas.
  - **Responsabilidades:**
    - Documentación de la arquitectura.
    - Evaluación de riesgos.
    - Liderazgo técnico.
  - Habilidades necesarias:
    - Comunicación efectiva.
    - Análisis de problemas.
    - Visión a largo plazo.
- **Actividad:** Estudio de caso: Analizar las responsabilidades de un arquitecto en un proyecto de software real.

1. Introducción al Rol del Arquitecto de Software

- Un arquitecto de software es el responsable de definir la estructura técnica de un sistema y garantizar que cumpla con los requisitos funcionales y no funcionales. Este rol actúa como un puente entre los objetivos del negocio y las decisiones técnicas.

Por qué es importante el rol



2. Roles del Arquitecto de Software

- **2.1 Diseñador Técnico**
  - Diseñar la estructura general del sistema, seleccionando estilos arquitectónicos adecuados.
  - Tomar decisiones sobre tecnologías, patrones de diseño y herramientas.
  - Crear diagramas arquitectónicos claros para comunicar la visión técnica.
- **2.2 Tomador de Decisiones**
  - Evaluar opciones técnicas, sopesar ventajas y desventajas y tomar decisiones críticas.
  - Ejemplo: Decidir entre una arquitectura monolítica o basada en microservicios.

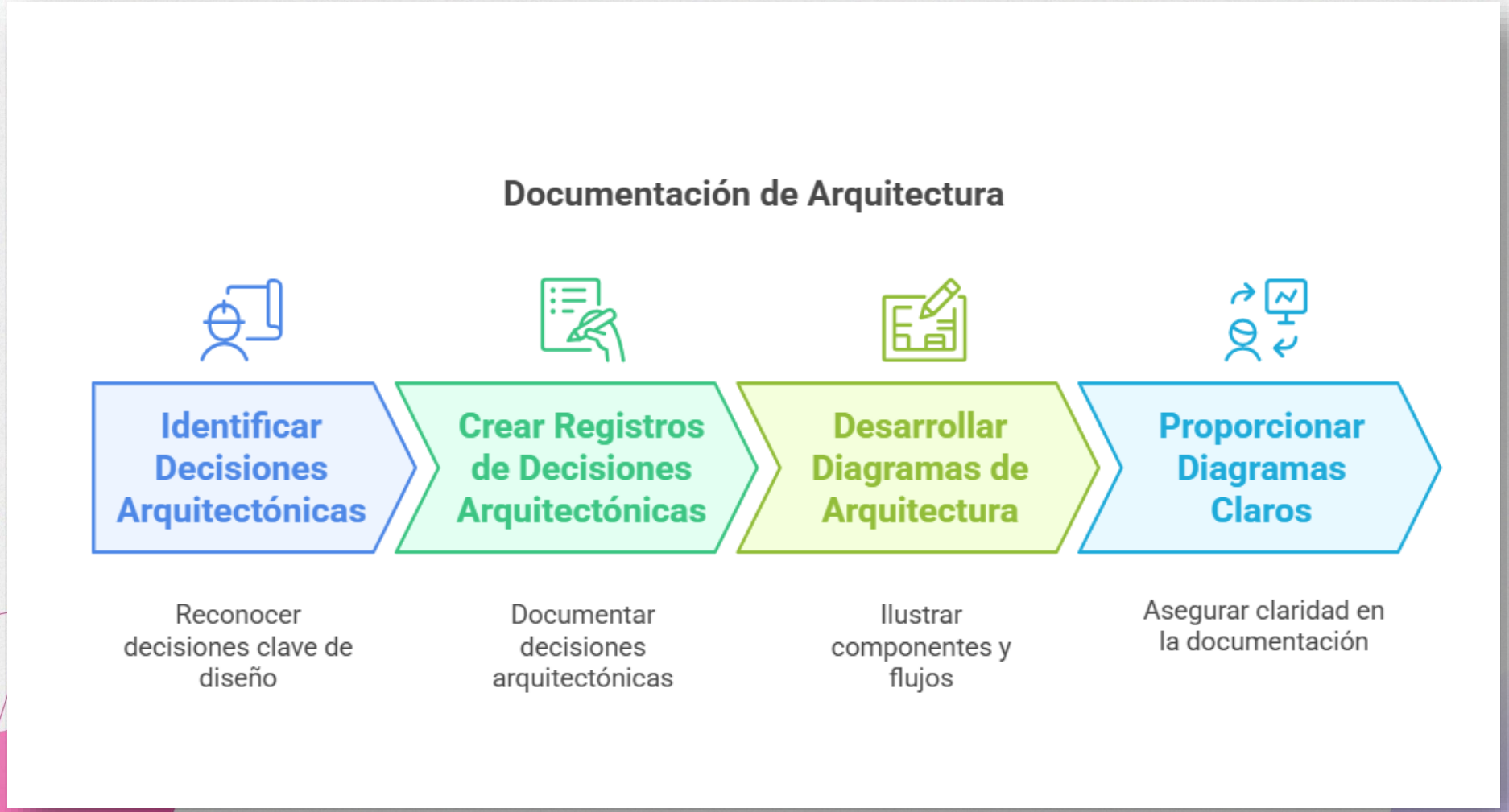
…Roles del Arquitecto de Software

- **2.3 Mediador entre Partes Interesadas**
  - Colaborar con stakeholders técnicos y no técnicos para garantizar que el sistema satisfaga las necesidades de negocio.
  - Traducir los requisitos del negocio en soluciones técnicas viables.
- **2.4 Mentor Técnico**
  - Actuar como guía para los desarrolladores, resolviendo dudas técnicas y promoviendo buenas prácticas.
  - Ejemplo: Implementar estándares de codificación o herramientas de revisión de código.

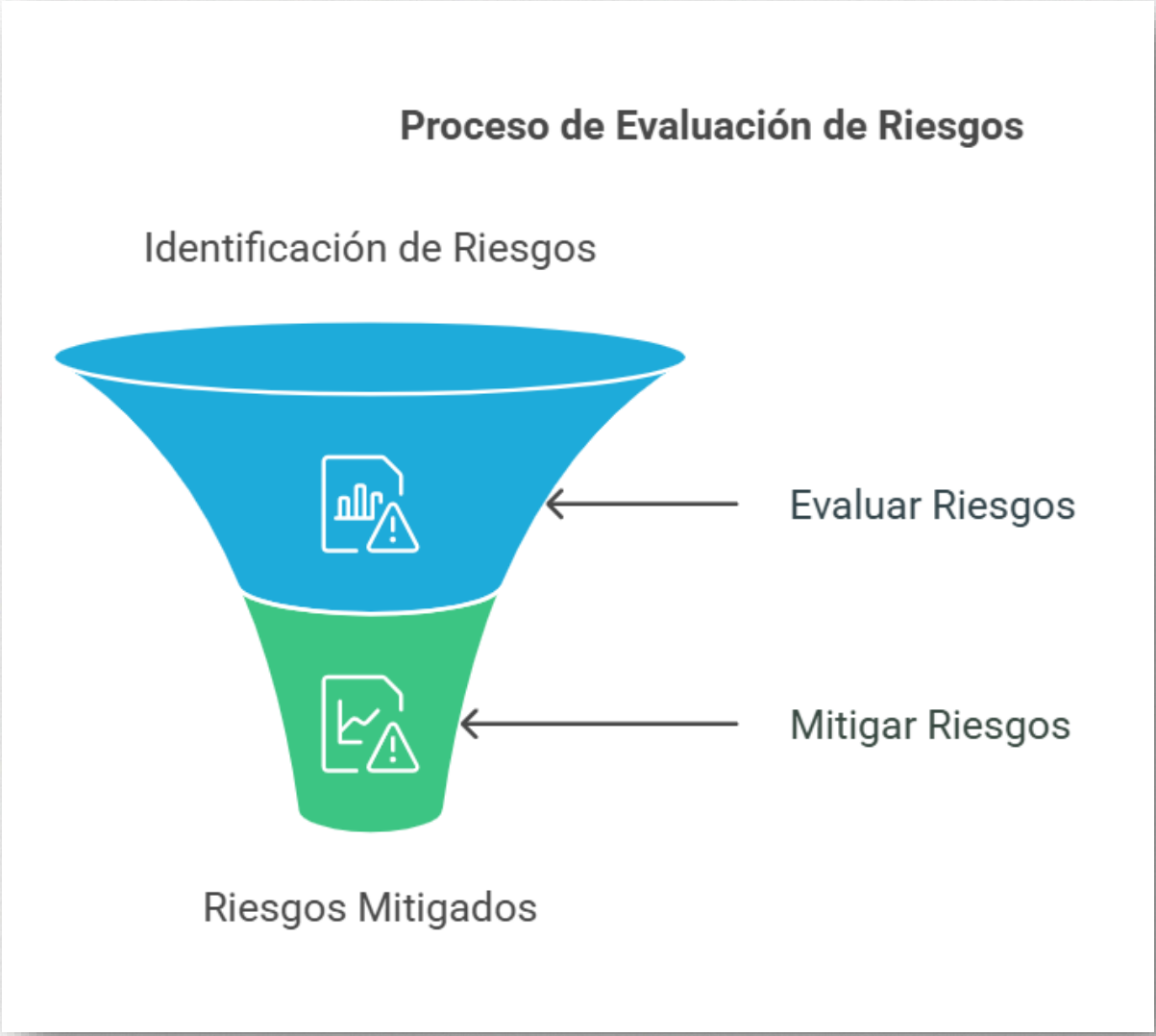
3. Responsabilidades del Arquitecto de Software

- **3.1 Documentar la Arquitectura**
- **3.2 Evaluar Riesgos**
- **3.3 Liderazgo Técnico**
- **3.4 Alinear Tecnología y Negocio**

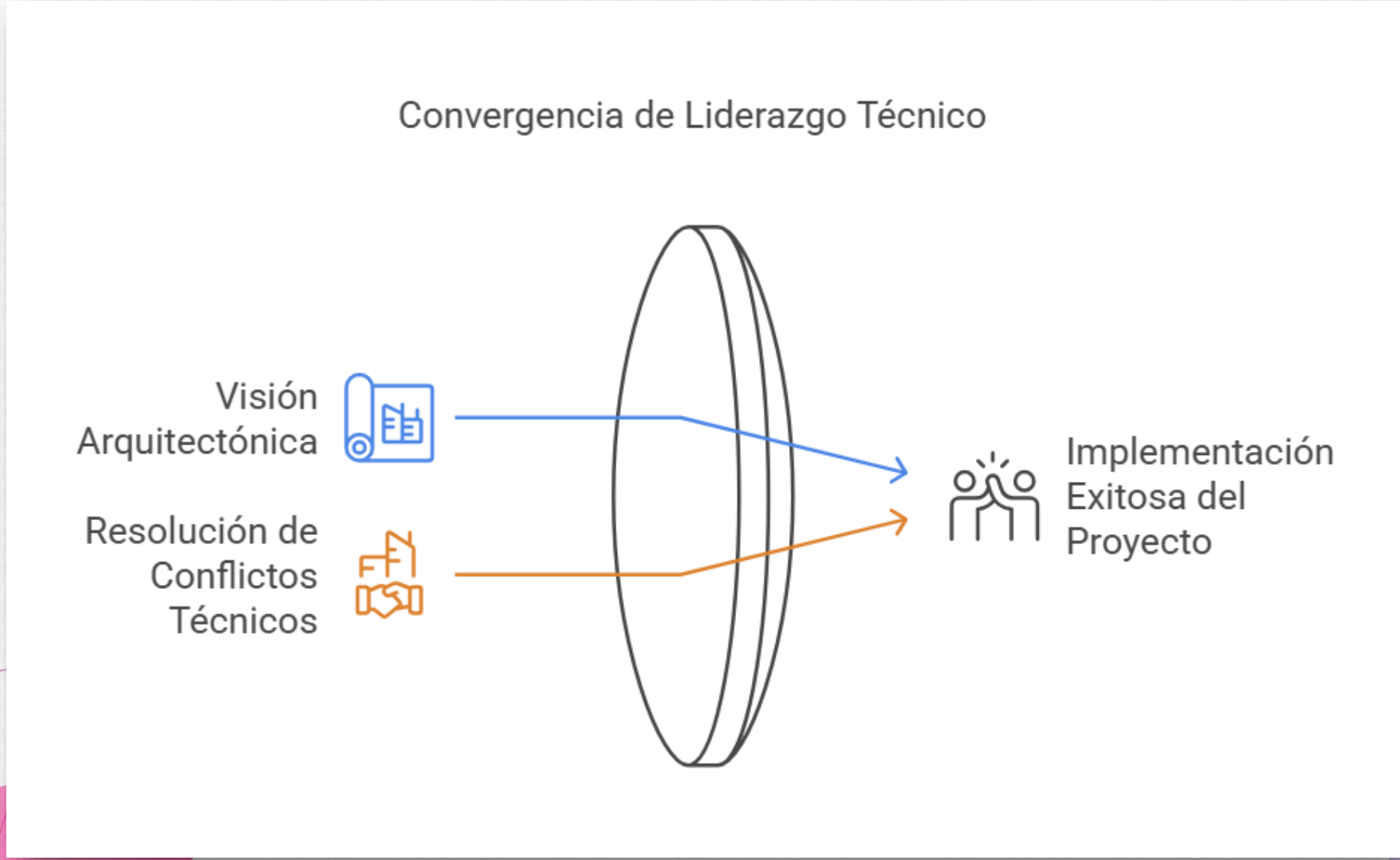
3.1 Documentar la Arquitectura



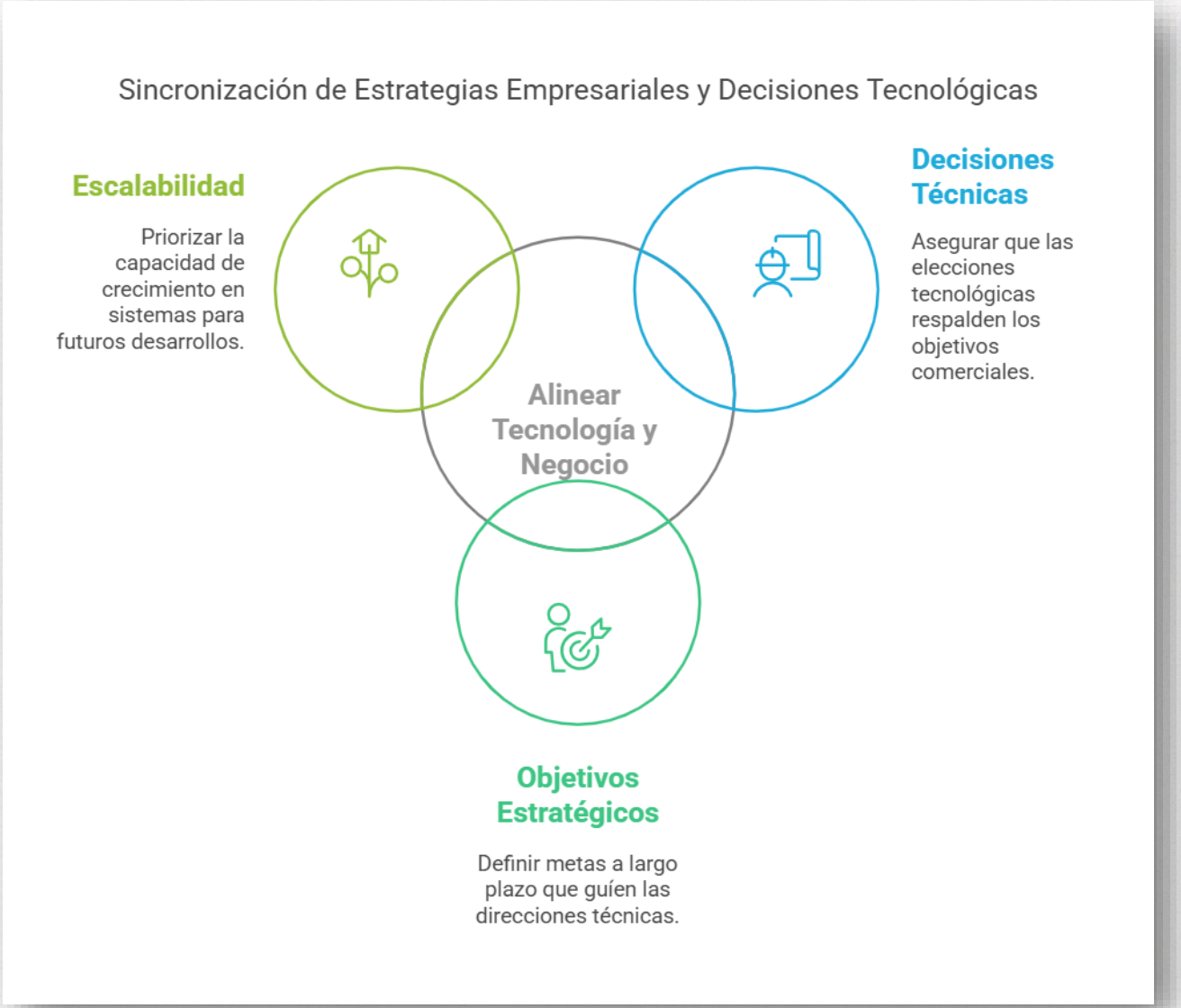
3.2 Evaluar Riesgos



3.3 Liderazgo Técnico



3.4 Alinear Tecnología y Negocio

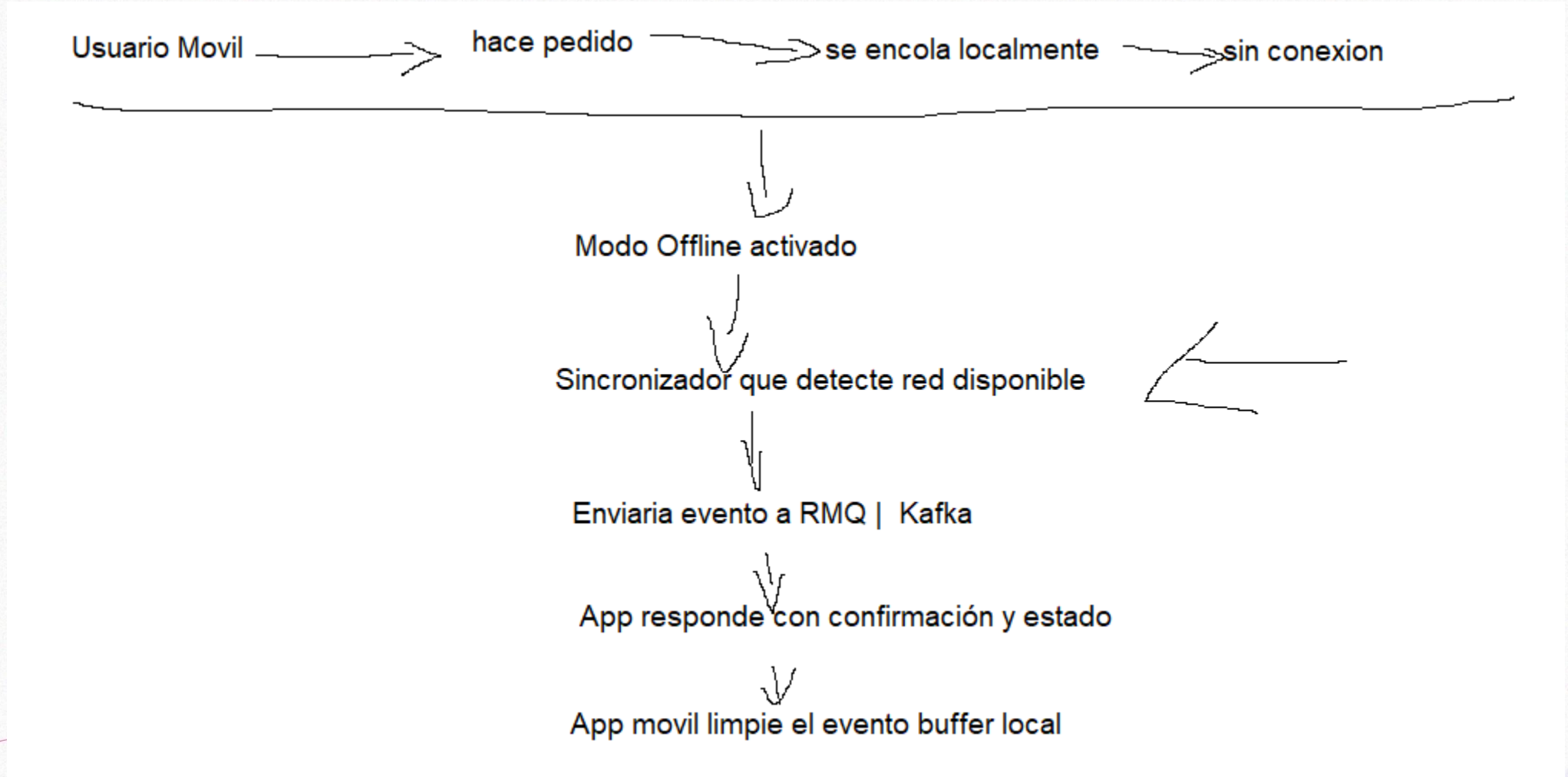


4. Habilidades Clave del Arquitecto de Software

- **4.1 Habilidades Técnicas**
  - Conocimientos avanzados en patrones de diseño, estilos arquitectónicos y herramientas.
  - Experiencia con tecnologías específicas del dominio del sistema.
- **4.2 Habilidades Blandas**
  - **Comunicación efectiva:** Saber explicar conceptos técnicos a audiencias no técnicas.
  - **Liderazgo:** Inspirar y guiar a los equipos técnicos hacia una meta común.
  - **Resolución de problemas:** Identificar y abordar desafíos rápidamente.

…Habilidades Clave del Arquitecto de Software

- **4.3 Habilidades Estratégicas**
  - Comprender los objetivos de negocio y su traducción en requerimientos técnicos.
  - Evaluar el impacto de las decisiones arquitectónicas a largo plazo.


Usuario Movil
hace pedido
se encola localmente
sin conexion
Modo Offline activado
Sincronizador que detecte red disponible
Enviaria evento a RMQ | Kafka
App responde con confirmación y estado
App movil limpie el evento buffer local

5. Ejemplos de Roles y Responsabilidades en Acción

- **Caso 1: Migración de Monolítico a Microservicios**
  - **Rol:** Diseñador técnico.
  - **Responsabilidad:** Dividir el sistema en servicios independientes, minimizando el impacto en los usuarios actuales.
  - **Habilidad requerida:** Conocimiento avanzado de arquitectura de microservicios.
- **Caso 2: Selección de una Tecnología de Base de Datos**
  - **Rol:** Tomador de decisiones.
  - **Responsabilidad:** Evaluar si usar una base de datos relacional o no relacional, considerando los requisitos de escalabilidad.
  - **Habilidad requerida:** Análisis comparativo de tecnologías

6. Actividad

- **Estudio de caso grupal:**
- **Objetivo:** Simular las decisiones que un arquitecto de software debe tomar en un proyecto real.
- **Instrucciones:**
  - Presentar un caso: *Diseño de una aplicación para reservación de hoteles.*
  - Definir los requisitos funcionales y no funcionales.
  - Dividir roles dentro del grupo (e.g., arquitecto, desarrolladores).
  - Diseñar una solución arquitectónica, tomando decisiones y documentándolas.
- **Duración:** 1 hora
- **Resultados esperados:** Un esquema arquitectónico con decisiones justificadas.

7. Buenas Prácticas para el Arquitecto de Software

- **Documentar todo:** Mantener un registro de decisiones y cambios arquitectónicos.
- **Estar actualizado:** Seguir tendencias y evaluar nuevas tecnologías.
- **Colaborar:** Involucrar a los stakeholders en discusiones clave para garantizar alineación.

Referencias Bibliográficas

- Bass, L., Clements, P., & Kazman, R. (2021). **Software Architecture in Practice** (4th Edition). Addison-Wesley Professional.
  - Profundiza en el rol del arquitecto y cómo alinear las decisiones arquitectónicas con objetivos de negocio.
- Rozanski, N., & Woods, E. (2011). **Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives**. Addison-Wesley.
  - Presenta técnicas prácticas para manejar múltiples perspectivas en el diseño arquitectónico.
- Hohpe, G. (2010). **The Software Architect Elevator: Redefining the Architect's Role in the Digital Enterprise**. O'Reilly Media.
  - Un enfoque práctico sobre cómo los arquitectos pueden influir tanto en el nivel técnico como estratégico.
- Kruchten, P. (1995). **Architectural Blueprints—The "4+1" View Model of Software Architecture**. IEEE Software.
  - Expone cómo documentar y comunicar la arquitectura de manera efectiva.
- Fowler, M. (2002). **Patterns of Enterprise Application Architecture**. Addison-Wesley.
  - Incluye casos de estudio sobre la aplicación práctica de patrones en decisiones arquitectónicas.

Tema 4 Importancia de la Arquitectura en el Desarrollo de Software

Importancia de la Arquitectura en el Desarrollo de Software (1 hora)

- **Objetivo:** Reconocer cómo una buena arquitectura afecta la calidad, mantenimiento y escalabilidad de los sistemas.
- **Contenido:**
  - Beneficios de una arquitectura bien diseñada:
    - Reducción de costos a largo plazo.
    - Mejora en el tiempo de desarrollo.
    - Mayor adaptabilidad a cambios.
  - Consecuencias de una arquitectura débil o inexistente.
  - Ejemplos de éxito y fracaso relacionados con arquitectura.
- **Actividad:** Debate: ¿Es la arquitectura siempre necesaria en proyectos pequeños?

1. Introducción

- La arquitectura de software es fundamental en cualquier proyecto de desarrollo, ya que define cómo interactúan los componentes del sistema y garantiza que cumpla con los requisitos funcionales y no funcionales. Una arquitectura robusta no solo asegura un desarrollo eficiente, sino que también reduce riesgos y costos asociados al mantenimiento y escalabilidad.
- **¿Qué sucede sin una buena arquitectura?**
  - Sistemas difíciles de mantener y escalar.
  - Aumento de errores en producción.
  - Mayor costo técnico (deuda técnica).
  - Falta de alineación con las metas del negocio.

2. Beneficios de una Arquitectura Bien Diseñada

- **2.1 Mejora la Calidad del Software**
  - **Definición clara:** Una buena arquitectura establece reglas y estándares claros para el desarrollo.
  - **Impacto:** Reduce defectos y asegura que el software sea confiable y seguro.
- **2.2 Facilita el Mantenimiento**
  - **Modularidad:** Divide el sistema en componentes independientes.
  - **Impacto:** Permite que los cambios en un módulo tengan un impacto mínimo en otros.

…Beneficios de una Arquitectura Bien Diseñada

- **2.3 Asegura la Escalabilidad**
  - **Diseño para crecimiento:** Considera el aumento de usuarios y datos.
  - **Impacto:** Permite agregar recursos o mejorar el rendimiento sin rediseñar el sistema.
- **2.4 Reduce los Costos a Largo Plazo**
  - **Eficiencia en el desarrollo:** Evita problemas técnicos desde las primeras fases.
  - **Impacto:** Minimiza el tiempo invertido en correcciones y ajustes futuros.
- **2.5 Mejora la Toma de Decisiones**
  - **Documentación arquitectónica:** Proporciona una base sólida para evaluar impactos de cambios.
  - **Impacto:** Facilita la planificación estratégica y la gestión de riesgos.

3. Factores de Éxito Relacionados con la Arquitectura

- **3.1 Requisitos No Funcionales**
  - La arquitectura debe satisfacer criterios como:
  - **Escalabilidad:** Aumentar el número de usuarios o datos sin comprometer el rendimiento.
  - **Seguridad:** Proteger el sistema contra amenazas externas e internas.
  - **Disponibilidad:** Garantizar que el sistema esté operativo el mayor tiempo posible.
  - **Mantenibilidad:** Hacer que el sistema sea fácil de actualizar y corregir.

…Factores de Éxito Relacionados con la Arquitectura

- **3.2 Alineación con el Negocio**
  - Una arquitectura bien diseñada asegura que las soluciones técnicas respalden los objetivos estratégicos del negocio.
  - Ejemplo: Si una empresa planea expandirse globalmente, la arquitectura debe soportar múltiples regiones y zonas horarias.
  - **3.3 Reducción de Riesgos**
  - Identificar problemas potenciales antes de la implementación.
  - Implementar estrategias para mitigar fallos, como redundancia y tolerancia a fallos.

4. Ejemplos Prácticos

- **Ejemplo 1: Sistema Bancario**
  - **Desafío:** Manejar millones de transacciones diarias.
  - **Solución arquitectónica:** Arquitectura basada en eventos para procesar transacciones en tiempo real.
  - **Impacto:** Alta disponibilidad, escalabilidad y seguridad.

…Ejemplos Prácticos

- **Ejemplo 2: Plataforma de Streaming**
  - **Desafío:** Gestionar una base de usuarios global con miles de visualizaciones simultáneas.
  - **Solución arquitectónica:** Uso de CDNs (Content Delivery Networks) y microservicios.
  - **Impacto:** Respuesta rápida y experiencia de usuario óptima.

…Ejemplos Prácticos

- **Ejemplo 3: Comercio Electrónico**
  - **Desafío:** Adaptarse rápidamente a picos de demanda (e.g., Black Friday).
  - **Solución arquitectónica:** Escalado horizontal en la nube.
  - **Impacto:** Disponibilidad garantizada durante periodos críticos.

5. Actividad

- **Estudio de caso: Evaluando una Arquitectura Existente**
- **Objetivo:** Analizar cómo una arquitectura existente satisface las necesidades del negocio y los requisitos no funcionales.
- **Instrucciones:**
  - Dividir a los participantes en equipos.
  - Proporcionar un caso práctico, como el diseño de un sistema de reservas de vuelos.
  - Cada equipo debe:
    - Evaluar si la arquitectura cumple con criterios clave como escalabilidad y mantenibilidad.
    - Proponer mejoras.
- **Duración:** 1 hora.
- **Resultados esperados:** Informe con análisis y recomendaciones.

6. Buenas Prácticas para Implementar una Arquitectura Sólida

- **Documentar desde el inicio:** Utilizar herramientas como diagramas UML para definir claramente los componentes y sus relaciones.
- **Evaluar tecnologías:** Seleccionar herramientas y frameworks alineados con los objetivos del proyecto.
- **Diseñar para el cambio:** Anticipar posibles evoluciones y construir una arquitectura flexible.
- **Adoptar principios sólidos:** Aplicar principios como separación de preocupaciones y modularidad.

7. Retos Comunes en la Arquitectura

- **Sobrecarga inicial:** Diseñar una arquitectura demasiado compleja puede ralentizar el desarrollo.
- **Subestimación de requisitos no funcionales:** Ignorar aspectos como seguridad y rendimiento puede causar problemas a largo plazo.
- **Deuda técnica:** Sacrificar calidad por velocidad puede ser costoso en el futuro.

Referencias Bibliográficas

- Bass, L., Clements, P., & Kazman, R. (2021). **Software Architecture in Practice** (4th Edition). Addison-Wesley Professional.
  - Profundiza en cómo la arquitectura afecta la calidad y sostenibilidad de los sistemas.
- Rozanski, N., & Woods, E. (2011). **Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives**. Addison-Wesley.
  - Explica cómo diseñar arquitecturas alineadas con los requisitos de negocio.
- Fowler, M. (2002). **Patterns of Enterprise Application Architecture**. Addison-Wesley.
  - Presenta patrones arquitectónicos que impactan positivamente la calidad del software.
- Kruchten, P. (1995). **Architectural Blueprints—The "4+1" View Model of Software Architecture**. IEEE Software.
  - Proporciona un enfoque para estructurar y comunicar arquitecturas complejas.
- IEEE. (2000). **IEEE Recommended Practice for Architectural Description of Software-Intensive Systems** (IEEE 1471).
  - Estándar reconocido sobre cómo documentar y evaluar arquitecturas.
- Hohpe, G. (2010). **The Software Architect Elevator: Redefining the Architect's Role in the Digital Enterprise**. O'Reilly Media.
  - Detalla cómo los arquitectos pueden influir en las estrategias empresariales a través de decisiones técnicas.