UNIVERSIDAD PERUANA LOS ANDES

FACULTAD: **INGENIERÍA**
ESCUELA PROFESIONAL: **SISTEMAS Y COMPUTACIÓN**

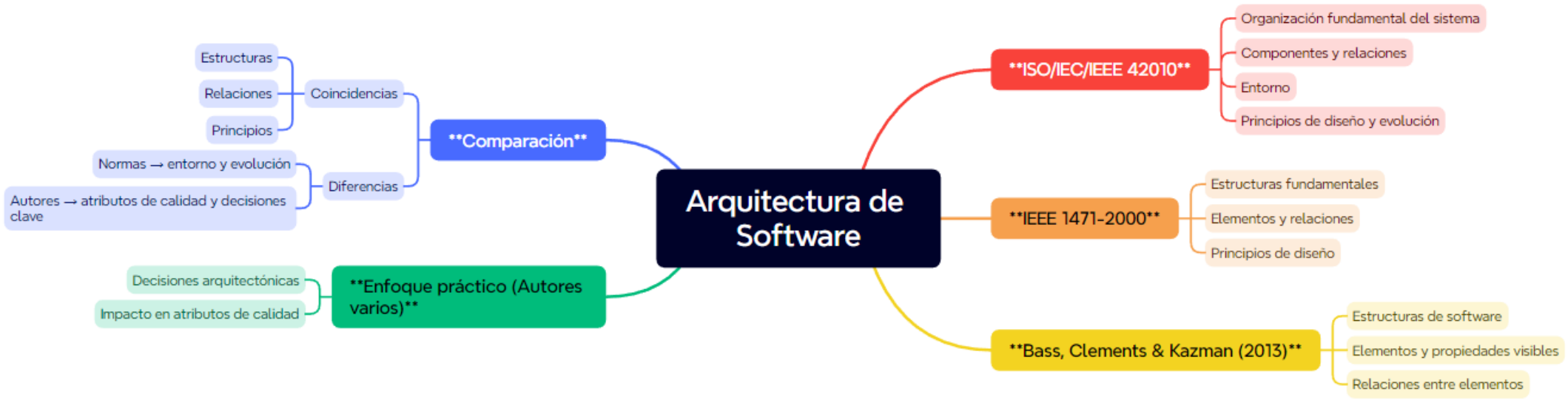
PROFESOR: **Fernandez Bejarano Raul Enrique**
ESTUDIANTE: **Limaylla Carhuallanqui Sebastian**

CICLO: VIII

HUANCAYO 2025

Tema 1: Introducción a la arquitectura de software

1.1 Subtema: definición de arquitectura de software



1.2 Subtema: Diferencias entre diseño de software y arquitectura

1. Introducción

El propósito de este documento es analizar un sistema conocido y diferenciar qué aspectos corresponden a la **arquitectura de software** y cuáles al **diseño de software**.
Se utilizará como ejemplo una **aplicación web educativa**.

2. Marco conceptual

- **Arquitectura de software**: Organización fundamental de un sistema en sus componentes principales, las relaciones entre ellos, y los principios que guían su diseño y evolución (ISO/IEC/IEEE 42010).
- **Diseño de software**: Actividad más detallada que traduce la arquitectura en soluciones técnicas específicas, algoritmos, estructuras de datos y diagramas que permiten la implementación.

3. Caso de estudio: Aplicación web educativa

Descripción breve: Plataforma en línea para gestionar cursos, estudiantes, materiales y evaluaciones.

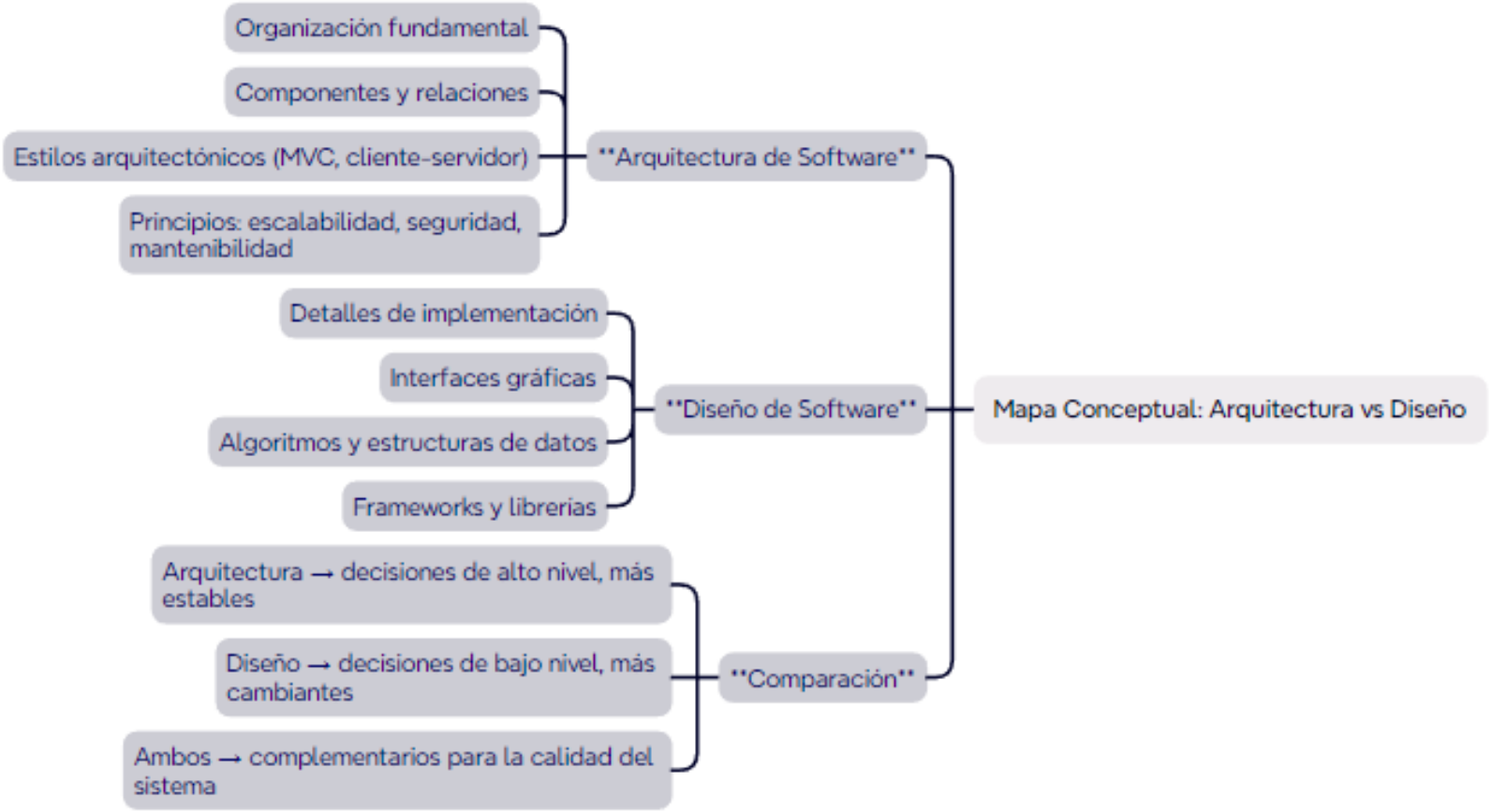
- **Elementos de la arquitectura**
  - Definición de capas (presentación, lógica de negocio, base de datos).
  - Elección del estilo arquitectónico (cliente-servidor, MVC, servicios web REST).
  - Componentes principales: módulos de autenticación, gestión de cursos, motor de calificaciones.
  - Decisiones sobre escalabilidad (uso de servidores distribuidos, nube).
  - Principios de seguridad (gestión de usuarios, roles, cifrado).
- **Elementos del diseño**
  - Interfaces gráficas: formularios de registro, panel del estudiante y del profesor.
  - Diagramas de clases: atributos y métodos en objetos como Usuario, Curso, Evaluación.
  - Algoritmos específicos: cálculo de notas, ordenamiento de contenidos, búsqueda de estudiantes.
  - Estilo visual: colores, tipografía, disposición de botones.
  - Elección de frameworks o librerías para implementación (React, Angular, Bootstrap).

4. Análisis crítico

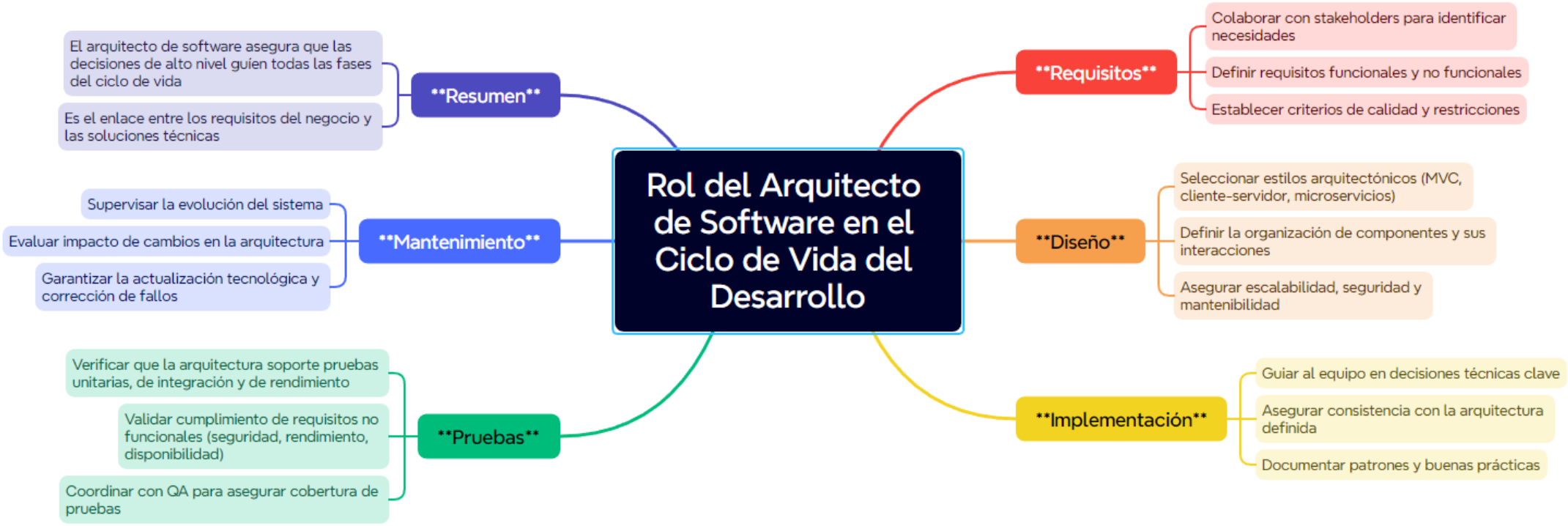
- La arquitectura define las decisiones de más alto nivel que impactan la calidad del sistema (escalabilidad, seguridad, mantenibilidad).
- El diseño concreta esas decisiones en implementaciones detalladas, más susceptibles de cambio en el corto plazo.
- Ambos niveles son complementarios: una arquitectura sólida no garantiza un buen diseño de interfaces, y un diseño atractivo no asegura un sistema escalable.

5. Conclusiones

- La arquitectura responde al *qué* y al *cómo general* del sistema.
- El diseño responde al *cómo detallado*.
- Diferenciarlos permite documentar y planificar correctamente el desarrollo de software.


Mapa Conceptual: Arquitectura vs Diseño
Arquitectura de Software
Organización fundamental
Componentes y relaciones
Estilos arquitectónicos (MVC, cliente-servidor)
Principios: escalabilidad, seguridad, mantenibilidad
Diseño de Software
Detalles de implementación
Interfaces gráficas
Algoritmos y estructuras de datos
Frameworks y librerias
Comparación
Arquitectura → decisiones de alto nivel, más estables
Diseño → decisiones de bajo nivel, más cambiantes
Ambos → complementarios para la calidad del sistema

1.3 Subtema: rol de arquitecto de software en el ciclo de vida del desarrollo



1.4 Subtema: impacto de la arquitectura en la calidad y eficiencia de los sistemas

1. Introducción
La arquitectura de software no solo define la estructura del sistema, sino que impacta directamente en la calidad, eficiencia y sostenibilidad de las soluciones tecnológicas. Un buen diseño arquitectónico permite alcanzar niveles altos de rendimiento, seguridad, escalabilidad y mantenibilidad.

2. Caso de estudio: Sistema de biblioteca digital
El sistema seleccionado es una plataforma digital de gestión bibliotecaria que permite:

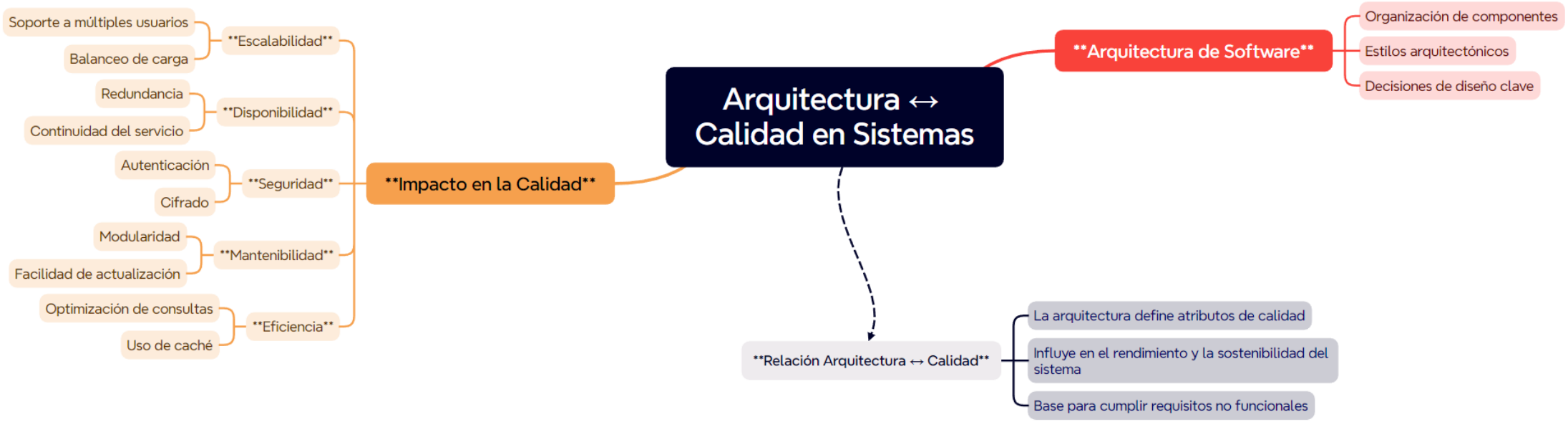
- Registro de usuarios.
- Préstamo y devolución de libros digitales.
- Búsqueda avanzada por autor, título y tema.
- Gestión de catálogos y estadísticas de uso.

3. Impacto de la arquitectura en la calidad

- **Escalabilidad**: Gracias al uso de servicios web distribuidos, el sistema soporta gran cantidad de usuarios concurrentes.
- **Disponibilidad**: Arquitectura basada en redundancia asegura continuidad del servicio.
- **Seguridad**: Implementación de capas de autenticación y cifrado protege datos sensibles.
- **Mantenibilidad**: El uso de una arquitectura modular facilita actualizaciones y correcciones de errores.
- **Eficiencia**: Optimización en consultas a la base de datos y uso de caché mejora la velocidad de respuesta.

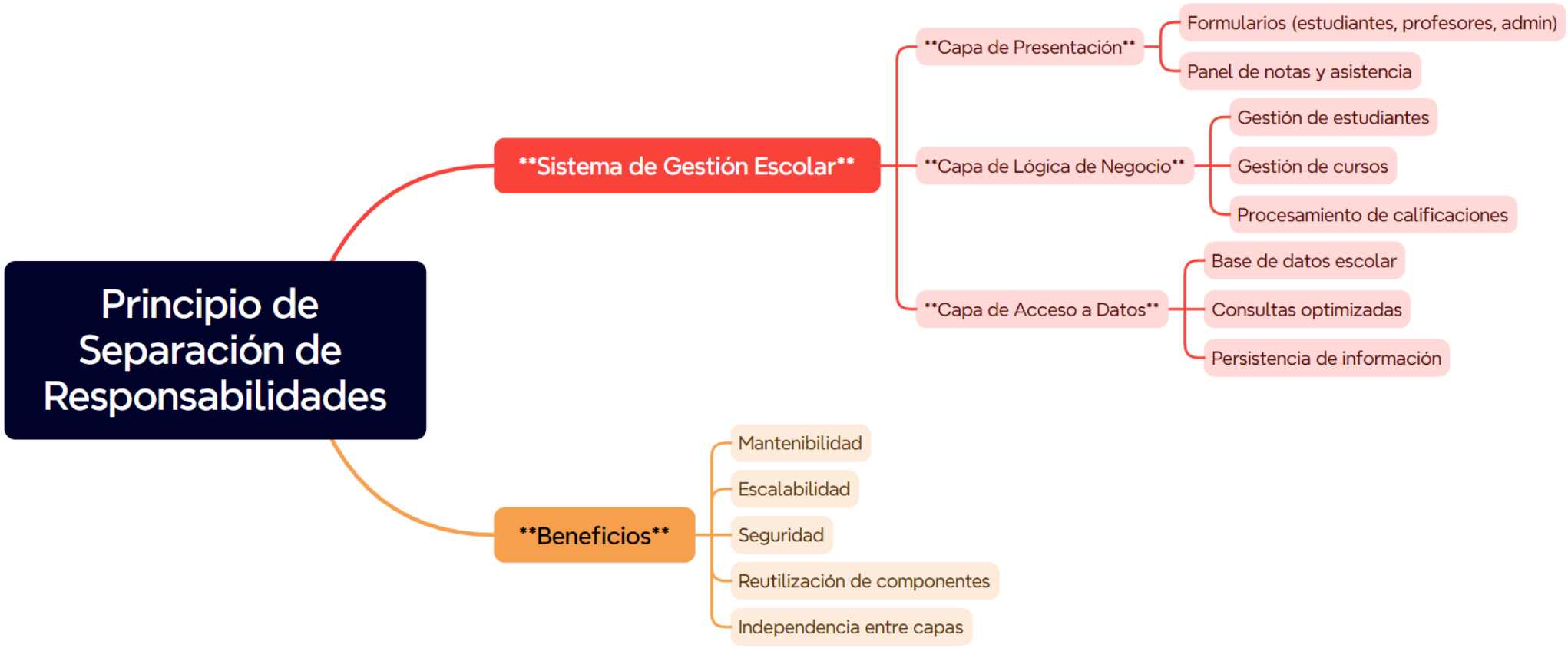
4. Conclusiones

- Una arquitectura bien definida es un factor determinante en la calidad global de un sistema.
- La eficiencia no depende únicamente de la programación, sino de decisiones arquitectónicas tempranas.
- Sistemas críticos, como una biblioteca digital, requieren arquitecturas robustas que integren seguridad, escalabilidad y mantenibilidad.


Arquitectura ↔ Calidad en Sistemas
Arquitectura de Software
Organización de componentes
Estilos arquitectónicos
Decisiones de diseño clave
Impacto en la Calidad
Escalabilidad
Soporte a múltiples usuarios
Balanceo de carga
Disponibilidad
Redundancia
Continuidad del servicio
Seguridad
Autenticación
Cifrado
Mantenibilidad
Modularidad
Facilidad de actualización
Eficiencia
Optimización de consultas
Uso de caché
Relación Arquitectura ↔ Calidad
La arquitectura define atributos de calidad
Influye en el rendimiento y la sostenibilidad del sistema
Base para cumplir requisitos no funcionales

Tema 2: Principios y conceptos fundamentales de la arquitectura de software

2.1 Subtema: principios de separación de responsabilidades



2.2 Subtema: cohesión y acoplamiento en la arquitectura

1. Introducción
La calidad de la arquitectura de software se mide en gran parte por el nivel de cohesión y acoplamiento de sus módulos.

- **Cohesión**: Grado en que los elementos de un módulo trabajan juntos para cumplir una única responsabilidad.
- **Acoplamiento**: Grado de dependencia entre módulos; mientras más alto, más difícil de mantener y escalar.

2. Análisis de un sistema académico
Caso: **Sistema de gestión académica en una universidad** con módulos para matrícula, notas, asistencia y biblioteca digital.

- Problema detectado:
  - El módulo de **matrícula** está fuertemente acoplado con el módulo de **notas**, porque ambos comparten directamente estructuras de datos sin intermediación.
  - El módulo de **asistencia** depende del de **matrícula** para identificar estudiantes, generando dependencia circular.

3. Impacto en la calidad del sistema

- **Baja mantenibilidad**: Un cambio en un módulo provoca ajustes en varios otros.
- **Escalabilidad limitada**: Es difícil separar módulos para ejecutarlos en servicios distribuidos.
- **Riesgo en la confiabilidad**: Un fallo en un módulo puede propagarse a todo el sistema.

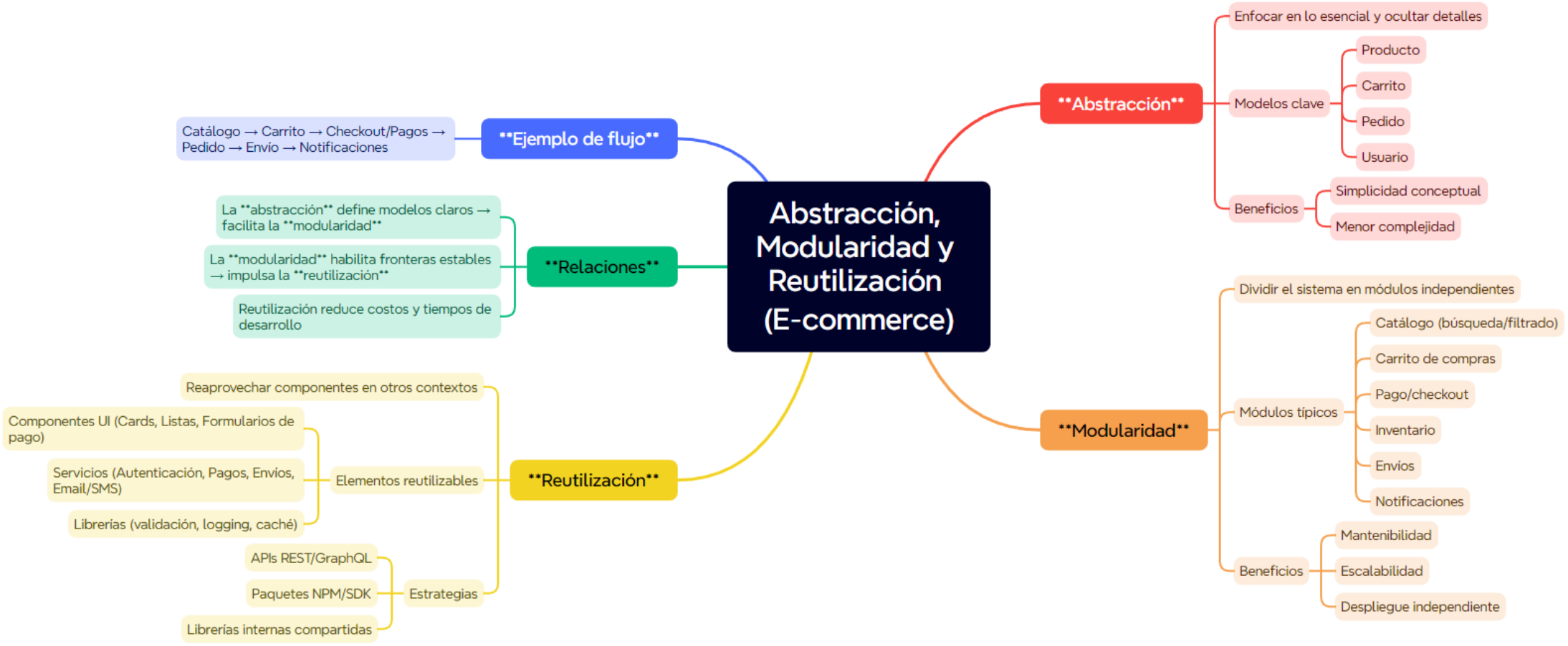
4. Propuestas de mejora

- Introducir **interfaces o APIs** para desacoplar módulos.
- Aplicar el principio de **responsabilidad única** para aumentar cohesión.
- Implementar **capas de servicios** para que los módulos no se llamen directamente.
- Adoptar un **estilo arquitectónico orientado a servicios (SOA o microservicios)**.

5. Conclusiones

- La cohesión debe ser alta dentro de cada módulo, asegurando que cumpla una función clara.
- El acoplamiento debe ser bajo, para que los módulos evolucionen de forma independiente.
- La mejora en estos aspectos eleva la mantenibilidad, escalabilidad y confiabilidad del sistema académico.

2.3 Subtema: abstracción, modularidad y reutilización.



2.4 Subtema: escalabilidad, mantenibilidad y confiabilidad.

1. Introducción
En los sistemas de *streaming*, la demanda de usuarios crece de forma acelerada, lo que obliga a diseñar arquitecturas capaces de escalar, mantenerse a lo largo del tiempo y garantizar un servicio confiable.

2. Caso de estudio: Plataforma de streaming de video
Características:

- Transmisión en tiempo real de video y audio.
- Gestión de cuentas y perfiles de usuario.
- Recomendaciones personalizadas.
- Soporte para millones de conexiones simultáneas.

3. Análisis de atributos de calidad

- **Escalabilidad**
  - Uso de **microservicios** para permitir crecimiento independiente de módulos.
  - **Balanceadores de carga** para distribuir tráfico entre servidores.
  - **CDN (Content Delivery Network)** para reducir latencia.
  - Escalado automático en la nube para soportar picos de demanda.
- **Mantenibilidad**
  - Código modular con principios SOLID.
  - Documentación actualizada de servicios y APIs.
  - Pruebas automatizadas (unitarias, integración, regresión).
  - Pipeline CI/CD para despliegues continuos.
- **Confiabilidad**
  - Replicación de datos en múltiples regiones.
  - Monitoreo y alertas en tiempo real.
  - Tolerancia a fallos mediante redundancia de servidores.

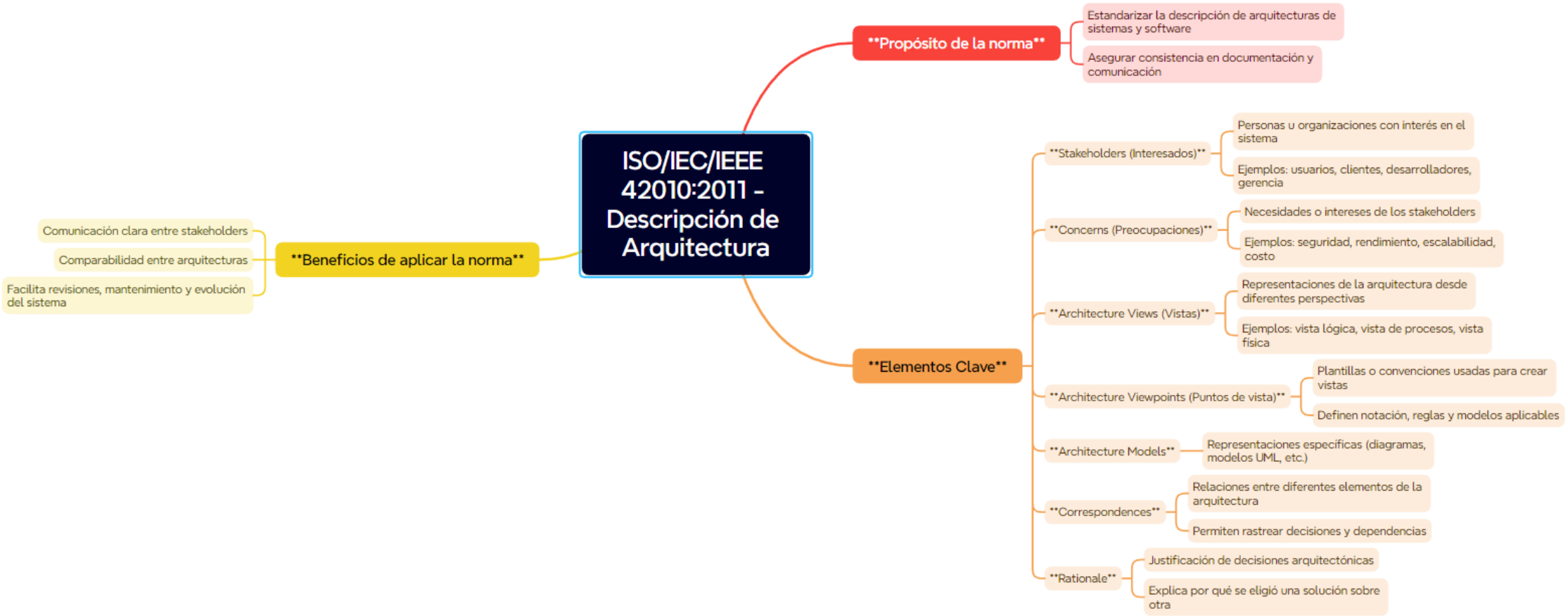
- Backups regulares y planes de recuperación ante desastres.

4. Conclusiones

- La **escalabilidad** asegura la capacidad de respuesta frente al crecimiento exponencial de usuarios.
- La **mantenibilidad** reduce costos y facilita la evolución del sistema.
- La **confiabilidad** garantiza un servicio estable y continuo para el usuario final.
- Una arquitectura robusta en estos tres atributos es esencial para plataformas de streaming competitivas.

Tema 3: ISO/IEC/IEEE 42010:2011:

3.1 Subtema:



3.2 Subtema: estándares de calidad de software ISO/IEC 25010.

1. Introducción

El estándar **ISO/IEC 25010** define un modelo de calidad de software que permite evaluar y comparar sistemas en función de características y subcaracterísticas específicas. Su aplicación facilita el aseguramiento de la calidad y la mejora continua en los proyectos de software.

2. Características principales del modelo ISO/IEC 25010

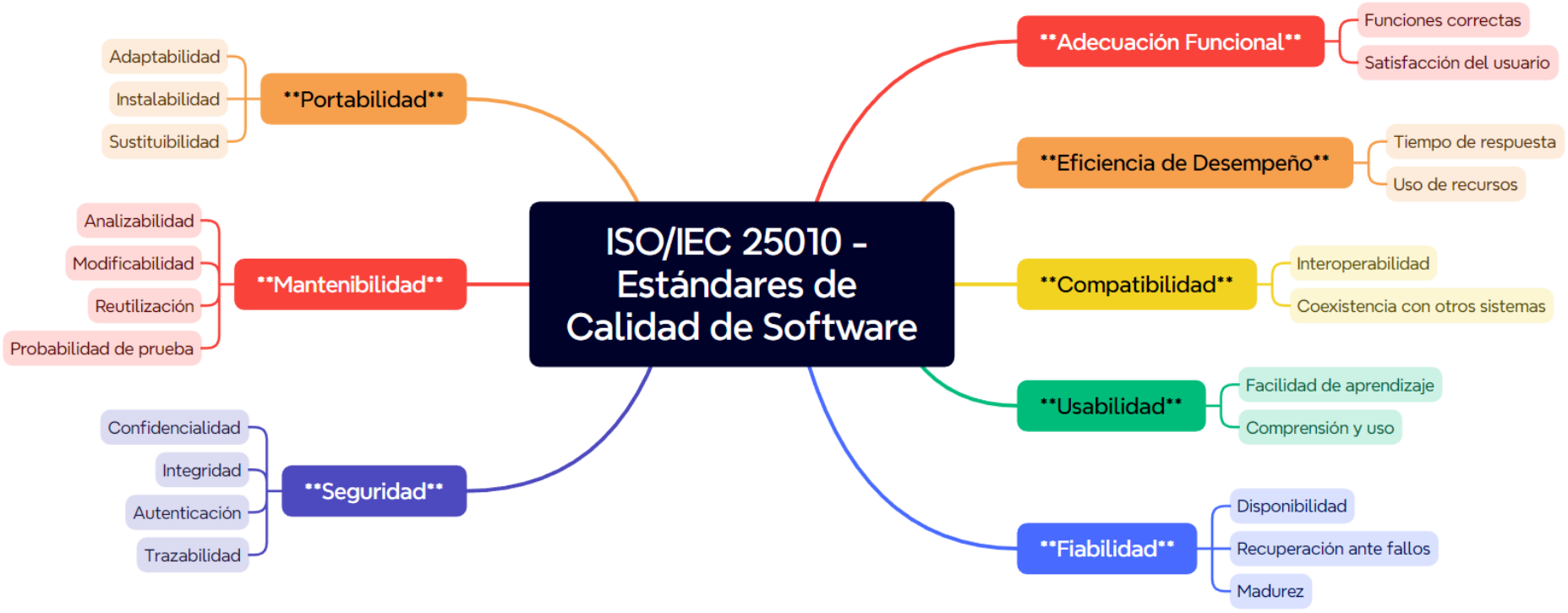
- **Adecuación funcional**: capacidad del software de proporcionar funciones que satisfagan necesidades explícitas e implícitas.
- **Eficiencia de desempeño**: utilización de recursos y tiempo de respuesta en diferentes condiciones de uso.
- **Compatibilidad**: grado en que el software puede operar con otros sistemas o compartir información.
- **Usabilidad**: facilidad de aprendizaje, comprensión y uso por parte de los usuarios.
- **Fiabilidad**: capacidad del sistema de mantener un nivel de desempeño bajo condiciones específicas.
- **Seguridad**: protección de la información y datos frente a accesos no autorizados.
- **Mantenibilidad**: facilidad con la que se puede modificar, mejorar o reparar el software.
- **Portabilidad**: capacidad del software para ser transferido de un entorno a otro.

3. Cuadro comparativo (ejemplo resumido)

Característica	Enfoque principal	Beneficio esperado
Adecuación funcional	Funciones cumplen con lo esperado	Mayor satisfacción del usuario
Eficiencia	Tiempo de respuesta y uso de recursos	Mejor desempeño del sistema
Compatibilidad	Integración con otros sistemas	Interoperabilidad
Usabilidad	Facilidad de uso	Aceptación y adopción por usuarios
Fiabilidad	Operación estable	Confianza del usuario
Seguridad	Protección de datos	Cumplimiento de normas y confianza
Mantenibilidad	Facilidad de actualización y corrección	Reducción de costos a largo plazo
Portabilidad	Adaptación a nuevos entornos	Mayor alcance del software

4. Conclusiones

- ISO/IEC 25010 proporciona un marco normativo que asegura la calidad integral del software.
- El cumplimiento de estas características mejora la satisfacción de los usuarios y la competitividad del sistema.
- Su aplicación debe guiar tanto la fase de diseño como la evaluación post-implementación.



3.3 Subtema: marcos de referencia internacionales (TOGAF, Zachman).

1. Introducción
Los marcos de referencia internacionales como **TOGAF** y **Zachman** brindan lineamientos para estructurar la arquitectura empresarial y de software. Permiten organizar los procesos de diseño, implementación y gestión tecnológica en las organizaciones, tanto públicas como privadas.

2. Caso de estudio: Empresa privada de e-commerce
La empresa analizada gestiona un portal de ventas en línea con miles de usuarios diarios. Presenta desafíos de integración entre sistemas de inventario, logística y atención al cliente.

3. Aplicación de TOGAF

- **Fase preliminar**: Identificar stakeholders y principios de arquitectura.
- **ADM (Architecture Development Method)**:
  - **Visión**: Definir objetivos estratégicos (ej. experiencia de usuario fluida, disponibilidad 24/7).
  - **Arquitectura empresarial**: Alinear procesos de logística, ventas y postventa.
  - **Arquitectura de sistemas de información**: Unificación de bases de datos y APIs para integración.
  - **Arquitectura tecnológica**: Uso de nube híbrida para escalabilidad.
  - **Plan de migración**: Estrategia por fases para modernizar los sistemas heredados.
- **Beneficios esperados**: mayor interoperabilidad, reducción de costos de integración, alineamiento TI-negocio.

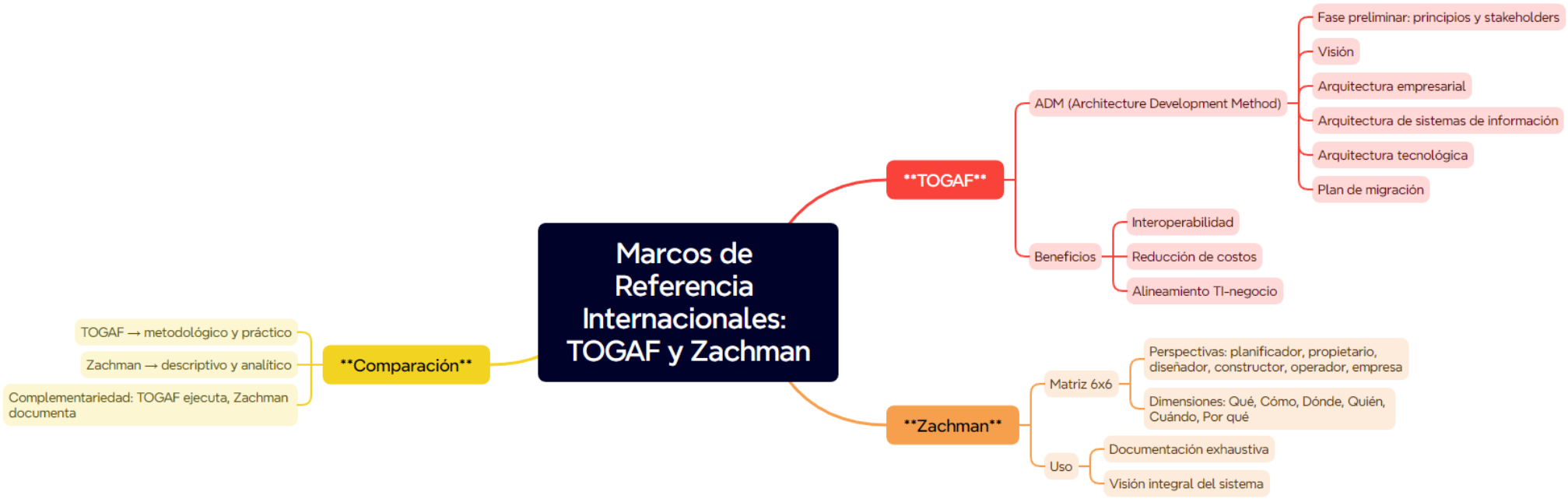
4. Aplicación de Zachman

- Proporciona una **matriz de 6x6** que organiza la arquitectura en base a *perspectivas* (planificador, propietario, diseñador, constructor, etc.) y *dimensiones* (qué, cómo, dónde, quién, cuándo, por qué).
- En este caso, permite documentar exhaustivamente cada aspecto:
  - **Qué**: inventario de productos.
  - **Cómo**: procesos de venta y despacho.
  - **Dónde**: infraestructura distribuida en nube.
  - **Quién**: roles (clientes, operadores, proveedores).

- **Cuándo**: tiempos de entrega.
- **Por qué**: objetivos estratégicos (crecimiento, satisfacción del cliente).

5. Conclusiones

- **TOGAF** es más práctico y metodológico, orientado a guiar la transformación organizacional.
- **Zachman** es más descriptivo y analítico, útil para documentar la arquitectura de manera completa.
- La empresa e-commerce se beneficiaría combinando ambos: **TOGAF para planificar y ejecutar** y **Zachman para documentar y comunicar**.



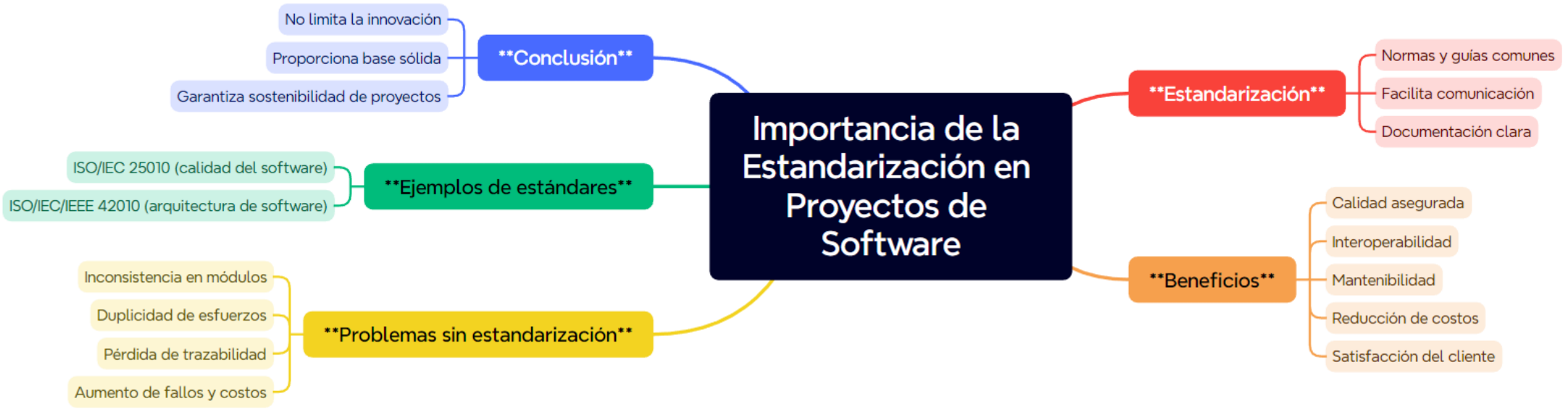
3.4 Subtema: importancia de la estandarización en proyectos de software.

La estandarización en proyectos de software constituye un pilar esencial para garantizar calidad, interoperabilidad y sostenibilidad en el tiempo. La ausencia de estándares puede derivar en sistemas poco consistentes, difíciles de mantener y con altos costos de corrección.

Un ejemplo común es el fracaso de proyectos de software donde cada equipo de desarrollo utiliza metodologías, herramientas y formatos diferentes sin un marco común. Esto genera duplicidad de esfuerzos, incompatibilidad entre módulos y pérdida de trazabilidad en los requisitos.

Los estándares internacionales, como **ISO/IEC 25010** o **ISO/IEC/IEEE 42010**, no solo formalizan prácticas, sino que sirven como guía para alinear al equipo hacia objetivos comunes. Además, permiten que las organizaciones documenten de forma clara los procesos y facilitan la comunicación entre clientes, desarrolladores y usuarios finales.

En conclusión, la estandarización no limita la creatividad, sino que proporciona una base sólida para la innovación. Su ausencia compromete la calidad del producto, mientras que su aplicación asegura la coherencia y la sostenibilidad del proyecto.


Importancia de la Estandarización en Proyectos de Software
Conclusión
No limita la innovación
Proporciona base sólida
Garantiza sostenibilidad de proyectos
Ejemplos de estándares
ISO/IEC 25010 (calidad del software)
ISO/IEC/IEEE 42010 (arquitectura de software)
Problemas sin estandarización
Inconsistencia en módulos
Duplicidad de esfuerzos
Pérdida de trazabilidad
Aumento de fallos y costos
Estandarización
Normas y guías comunes
Facilita comunicación
Documentación clara
Beneficios
Calidad asegurada
Interoperabilidad
Mantenibilidad
Reducción de costos
Satisfacción del cliente

Tema 4: Estilos y patrones arquitectónicos:

4.1 Subtema: estilos arquitectónicos (monolítico, en capas, cliente-servidor, microservicios)

1. Introducción
Los estilos arquitectónicos representan enfoques de organización de un sistema de software. Cada estilo tiene ventajas y limitaciones, y la elección depende del contexto y los objetivos del sistema.

2. Caso de estudio: Sistema de ventas

- **Arquitectura Monolítica**
  - Todo el sistema (ventas, clientes, facturación, inventario) se implementa como una única aplicación.
  - **Ventajas**: simplicidad de despliegue, menor costo inicial.
  - **Desventajas**: difícil escalabilidad, complejidad en mantenimiento, riesgo de fallo total.
- **Arquitectura en Capas**
  - Divide el sistema en capas (presentación, negocio, datos).
  - **Ventajas**: separación de responsabilidades, mantenibilidad, pruebas más claras.
  - **Desventajas**: posibles problemas de rendimiento, rigidez en cambios entre capas.
- **Arquitectura Cliente-Servidor**
  - Cliente solicita servicios al servidor central.
  - **Ventajas**: distribución de carga, centralización de datos.
  - **Desventajas**: dependencia del servidor, problemas de latencia si hay muchos clientes.
- **Arquitectura de Microservicios**
  - El sistema se divide en servicios pequeños, independientes y desplegables de forma autónoma (ventas, inventario, facturación).
  - **Ventajas**: escalabilidad flexible, resiliencia, despliegue independiente.
  - **Desventajas**: mayor complejidad, necesidad de orquestación, sobrecarga de comunicaciones.

3. Conclusiones

- La **arquitectura monolítica** es adecuada para proyectos pequeños.

- La **arquitectura en capas** es ideal para sistemas medianos con equipos reducidos.
- La **arquitectura cliente-servidor** es práctica para entornos con centralización de datos.
- La **arquitectura de microservicios** es la más robusta para sistemas grandes y escalables.
- La decisión depende del tamaño del proyecto, presupuesto, equipo y proyección de crecimiento.

4.2 Subtema: patrones Arquitectónicos Más Utilizados.

4.3 Subtema: criterios de selección de un estilo o patrón arquitectónico.

1. Introducción

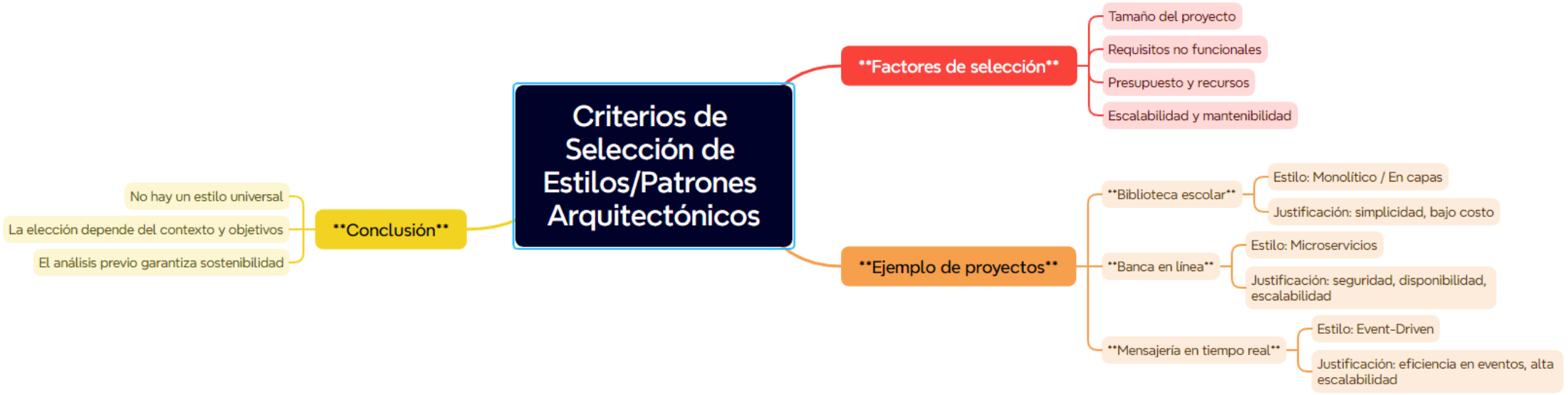
La elección de un estilo o patrón arquitectónico depende de múltiples factores: tamaño del proyecto, requisitos no funcionales (rendimiento, seguridad, escalabilidad), recursos disponibles y expectativas de los usuarios.

2. Proyectos ficticios analizados

- **Proyecto A: Sistema de gestión de biblioteca escolar**
  - Contexto: sistema pequeño, con pocos usuarios concurrentes.
  - Estilo recomendado: **Monolítico** o **en capas**.
  - Justificación: simplicidad, menor costo inicial, fácil despliegue.
- **Proyecto B: Plataforma de banca en línea**
  - Contexto: sistema crítico, con requisitos de seguridad, disponibilidad 24/7 y miles de usuarios concurrentes.
  - Estilo recomendado: **Microservicios**.
  - Justificación: permite escalabilidad, redundancia y despliegue independiente de servicios.
- **Proyecto C: Aplicación de mensajería en tiempo real**
  - Contexto: sistema de comunicación instantánea con millones de eventos por segundo.
  - Estilo recomendado: **Event-Driven**.
  - Justificación: responde eficientemente a eventos en tiempo real, altamente desacoplado y escalable.

3. Conclusiones

- No existe un estilo único válido para todos los proyectos.
- La selección depende de las características del sistema y de los atributos de calidad requeridos.
- Un análisis previo de riesgos, costos y objetivos es clave para tomar la decisión adecuada.



4.4 Subtema: beneficios de aplicar patrones reconocidos en la eficiencia del sistema.

1. Introducción
El uso de patrones arquitectónicos reconocidos permite diseñar sistemas más robustos, escalables y sostenibles. Netflix es un ejemplo emblemático de cómo los **microservicios** mejoran la eficiencia en la entrega de servicios digitales a escala global.

2. Caso de estudio: Netflix

- Netflix pasó de un **sistema monolítico** a una arquitectura basada en **microservicios** para enfrentar la creciente demanda de usuarios y contenidos.
- Este cambio permitió que la empresa atendiera millones de usuarios simultáneamente con mínima latencia.

3. Beneficios obtenidos por los microservicios

- **Escalabilidad**: cada servicio se escala de manera independiente según la demanda.
- **Resiliencia**: los fallos en un servicio no afectan a todo el sistema, gracias al aislamiento.
- **Agilidad en el desarrollo**: los equipos pueden trabajar en servicios específicos sin bloquear a otros.
- **Mantenibilidad**: facilita la actualización y despliegue continuo de nuevas funciones.
- **Eficiencia operativa**: optimiza el uso de recursos en la nube mediante balanceo de carga dinámico.
- **Innovación constante**: el desacoplamiento permitió incorporar IA para recomendaciones sin interrumpir otros servicios.

4. Conclusiones

- Los microservicios han sido un factor clave en el éxito de Netflix.
- Aplicar patrones reconocidos como este reduce riesgos, mejora la eficiencia y asegura continuidad del servicio.
- La experiencia de Netflix demuestra que la adopción de un estilo arquitectónico adecuado impacta directamente en la competitividad de las empresas digitales.

Tema 5: Importancia de la arquitectura de software en el desarrollo de sistemas.

5.1 Subtema: la arquitectura como guía para el desarrollo y mantenimiento

1. Introducción

La arquitectura de software funciona como una hoja de ruta que orienta las decisiones de diseño, implementación y mantenimiento de un sistema. Sirve como marco de referencia para asegurar que los objetivos funcionales y no funcionales se cumplan a lo largo de todo el ciclo de vida.

2. Caso simulado: Sistema de gestión hospitalaria

Este sistema permite la gestión de pacientes, citas, historiales médicos y facturación.

- **Guía en el desarrollo**
  - Definición de **capas** (presentación, lógica de negocio, datos).
  - Selección de **estilo arquitectónico** (en capas con servicios REST).
  - Establecimiento de **estándares de codificación** y uso de frameworks.
  - Integración con sistemas externos (farmacia, laboratorio) mediante APIs.
- **Guía en el mantenimiento**
  - Modularidad facilita actualizaciones sin afectar otras áreas.
  - Documentación arquitectónica ayuda en la incorporación de nuevos desarrolladores.
  - Arquitectura planificada asegura **escalabilidad** frente a mayor número de pacientes.
  - Soporte a nuevas funcionalidades como telemedicina sin rediseñar todo el sistema.

3. Análisis crítico

- Una arquitectura clara reduce riesgos de desviación en el proyecto.
- La falta de una guía arquitectónica puede ocasionar sobrecostos, retrabajos y problemas de calidad.
- El mantenimiento se beneficia al contar con estructuras documentadas y decisiones de diseño registradas.

4. Conclusiones

- La arquitectura guía tanto el desarrollo inicial como la evolución posterior del sistema.
- Es esencial documentarla y revisarla continuamente.
- Actúa como un punto de referencia que asegura alineamiento entre negocio, usuarios y tecnología.

5.2 Subtema: impacto en la eficiencia, escalabilidad y seguridad del sistema

1. Introducción
En los sistemas bancarios, la arquitectura de software es determinante para asegurar eficiencia operativa, escalabilidad frente al crecimiento de usuarios y seguridad en la protección de datos sensibles.

2. Caso analizado: Sistema bancario en línea
Características: gestión de cuentas, transferencias, pagos en línea, autenticación de usuarios y reportes financieros.

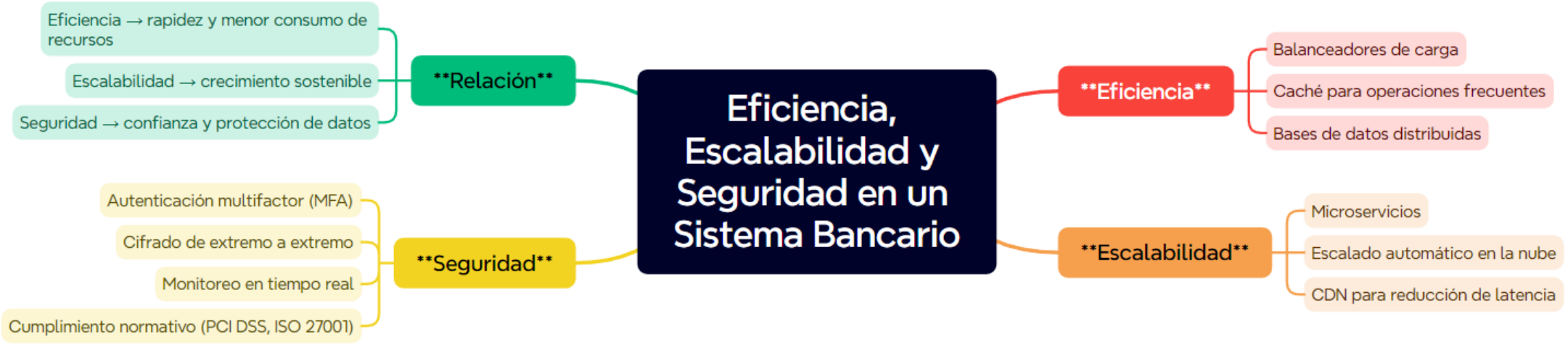
3. Análisis de propiedades

- **Eficiencia**
  - Optimización de transacciones mediante balanceadores de carga.

    - Uso de caché para operaciones frecuentes (consultas de saldo).
    - Bases de datos distribuidas para reducir tiempos de respuesta.
- **Escalabilidad**
    - Arquitectura basada en microservicios para escalar módulos específicos (transferencias, pagos).
    - Uso de infraestructura en la nube con escalado automático.
    - CDN para distribuir contenido y reducir latencia.
- **Seguridad**
    - Autenticación multifactor (MFA) para acceso a cuentas.
    - Cifrado de extremo a extremo en transacciones.
    - Monitoreo en tiempo real para detección de fraudes.
    - Cumplimiento de normativas internacionales (PCI DSS, ISO 27001).

4. Conclusiones

- La eficiencia garantiza una experiencia fluida para el cliente.
- La escalabilidad asegura que el sistema soporte millones de usuarios concurrentes.
- La seguridad protege la confianza del usuario y el cumplimiento normativo.
- La combinación de estas propiedades fortalece la competitividad y sostenibilidad de los sistemas bancarios modernos.



5.3 Subtema: reducción de riesgos y costos a través de una arquitectura bien definida

1. Introducción

La ausencia de una arquitectura de software sólida genera riesgos elevados y costos adicionales en proyectos de gran escala. Un ejemplo emblemático es el caso de **Healthcare.gov** en Estados Unidos, cuyo lanzamiento inicial estuvo marcado por múltiples fallos técnicos.

2. Análisis del caso Healthcare.gov

- Problema: el sitio colapsó en su lanzamiento en 2013, impidiendo que millones de ciudadanos se registraran en el sistema de salud.
- Causas arquitectónicas:
  - Integración deficiente entre módulos.
  - Alta dependencia de sistemas heredados.

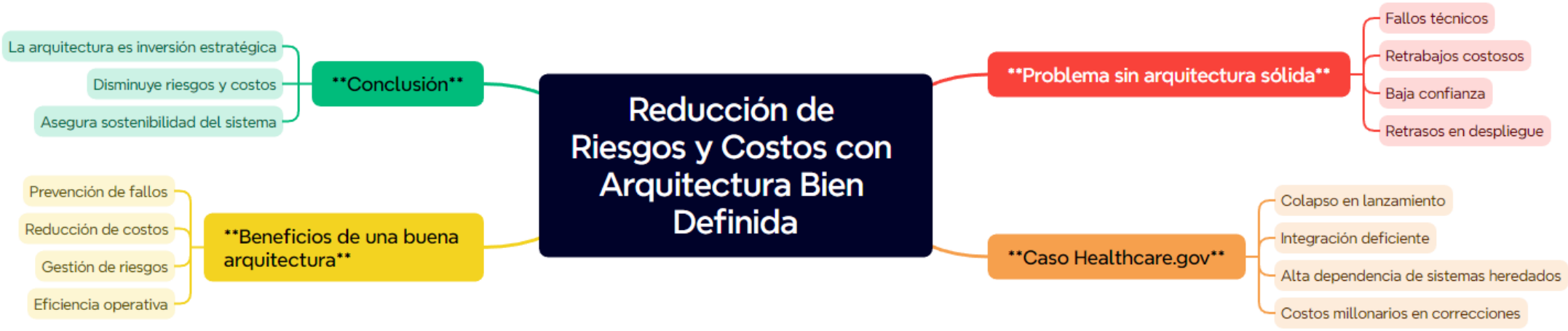
    - Falta de pruebas de escalabilidad.
    - Carencia de un diseño modular y planificado.
- Consecuencias:
    - Pérdida de confianza ciudadana.
    - Costos de reparación millonarios para rehacer módulos.
    - Retrasos en la implementación de la reforma sanitaria.

3. Impacto de una arquitectura bien definida

- **Prevención de fallos**: una arquitectura modular y escalable hubiera permitido detectar errores en etapas tempranas.
- **Reducción de costos**: al evitar retrabajos y rediseños masivos.
- **Gestión de riesgos**: mejor documentación y trazabilidad hubieran facilitado la toma de decisiones.
- **Eficiencia operativa**: garantizar disponibilidad del sistema en momentos críticos.

4. Conclusión

La arquitectura de software no es solo un aspecto técnico, sino una inversión estratégica que reduce riesgos y costos. Casos como Healthcare.gov muestran que la improvisación arquitectónica puede llevar al fracaso de proyectos de impacto nacional.



5.4 Subtema: casos prácticos de éxito y fracaso relacionados con la arquitectura de software

1. Introducción
La arquitectura de software ha sido un factor determinante en el éxito o fracaso de grandes proyectos tecnológicos. Analizar experiencias reales permite comprender cómo decisiones arquitectónicas impactan directamente en la eficiencia, escalabilidad y sostenibilidad de los sistemas.

2. Casos de éxito

- **Amazon**
  - Migración de una arquitectura monolítica a microservicios.
  - Beneficios: escalabilidad global, resiliencia ante fallos, agilidad en despliegues.
  - Resultado: líder mundial en comercio electrónico y servicios en la nube (AWS).
- **Netflix**

- Transición de servidores dedicados a una arquitectura basada en la nube y microservicios.
- Beneficios: alta disponibilidad, tolerancia a fallos, personalización mediante IA.
- Resultado: plataforma de streaming capaz de atender millones de usuarios en tiempo real.

3. Caso de fracaso

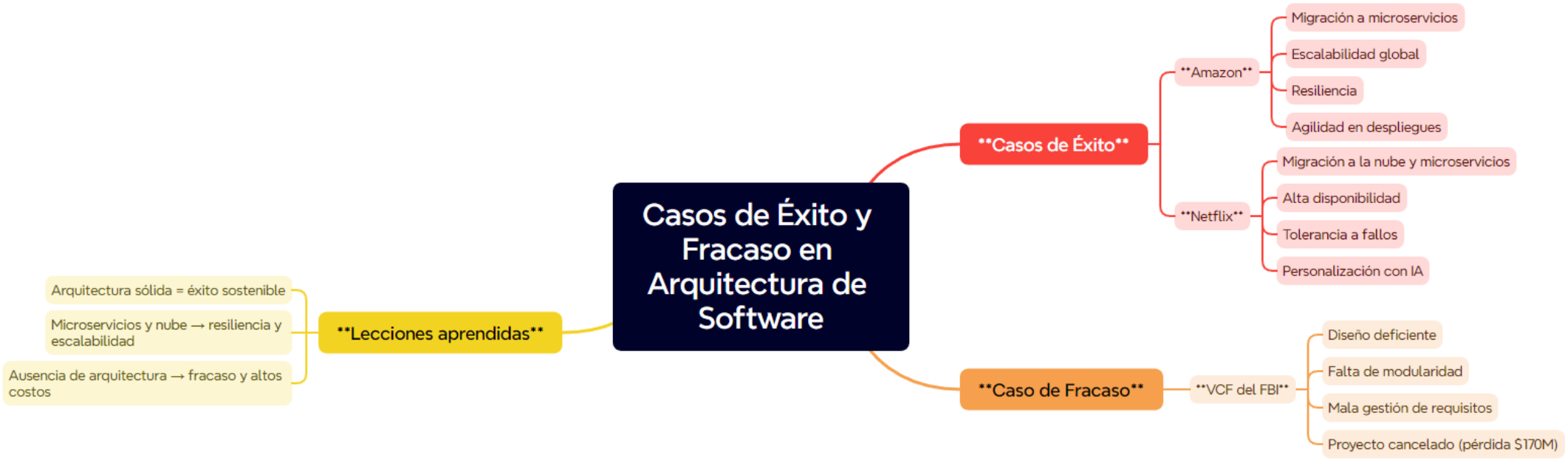
- **VCF del FBI (Virtual Case File)**
  - Proyecto de sistema de gestión de casos iniciado en 2000.
  - Problemas:
    - Diseño arquitectónico poco definido.
    - Alta complejidad sin modularidad.
    - Falta de pruebas integrales y mala gestión de requisitos.
  - Consecuencias:
    - Inversión perdida de más de 170 millones de dólares.
    - Proyecto cancelado en 2005 sin implementación exitosa.

4. Análisis crítico

- La falta de una arquitectura clara y escalable fue el origen del fracaso del FBI VCF.
- Por el contrario, Amazon y Netflix demostraron que los microservicios y la nube ofrecen resiliencia y crecimiento sostenible.
- Una arquitectura adecuada no solo respalda la operación actual, sino que permite evolucionar con las demandas del mercado.

5. Conclusión

La historia muestra que el éxito tecnológico depende de una arquitectura planificada y flexible. Empresas que priorizan decisiones arquitectónicas logran ventaja competitiva; aquellas que las descuidan, enfrentan altos costos y posibles fracasos.


Casos de Éxito y Fracaso en Arquitectura de Software
Casos de Éxito
Amazon
Migración a microservicios
Escalabilidad global
Resiliencia
Agilidad en despliegues
Netflix
Migración a la nube y microservicios
Alta disponibilidad
Tolerancia a fallos
Personalización con IA
Caso de Fracaso
VCF del FBI
Diseño deficiente
Falta de modularidad
Mala gestión de requisitos
Proyecto cancelado (pérdida $170M)
Lecciones aprendidas
Arquitectura sólida = éxito sostenible
Microservicios y nube → resiliencia y escalabilidad
Ausencia de arquitectura → fracaso y altos costos