Sesion 1
Arquitectura de Software

“Los principios de diseño permiten construir software escalable, modular y más fácil de mantener”

— Pressman & Maxim.

Objetivos de la sesión:

- Comprender qué es arquitectura de software desde una perspectiva académica.
- Distinguir entre diseño y arquitectura con base en la literatura.
- Analizar la relevancia del rol del arquitecto en contextos ágiles y tradicionales.
- Identificar implicancias prácticas de una arquitectura mal definida.
- Aplicar una visión crítica mediante un caso real documentado (HealthCare.gov).

¿Qué es la Arquitectura de Software?

¿Alguna vez trabajaste en un sistema difícil de mantener o escalar?

¿Por qué crees que fue así?

¿Cuál es la diferencia entre un sistema que lleva 30 años funcionando y otro que apenas 6 meses luego de construirse ya debe ser reescrito?

¿Qué es la Arquitectura de Software?

"Software architecture entails the set of significant decisions about the organization of a software system."

La arquitectura de software implica el conjunto de decisiones significativas sobre la organización de un sistema de software.

— Booch, IEEE Software (2009)

“La arquitectura de software de un sistema es la estructura o conjunto de estructuras que lo componen, incluyendo los componentes de software, sus propiedades visibles externamente y las relaciones entre ellos.”

— Bass, Clements, Kazman. *Software Architecture in Practice*, 4th ed., 2021.

“La arquitectura moderna se enfoca menos en estructuras estáticas y más en habilitar el cambio y la evolución, tratando la arquitectura como un conjunto de restricciones estratégicas y principios rectores.”

— Yoder et al., *Facilitating Software Architecture*, 2021.

El rol del arquitecto

Antes, el arquitecto era el que “dibujaba” el sistema y entregaba el diagrama al equipo, un planeador.

El rol del arquitecto en la era Post-Agilismo

En entornos ágiles, los arquitectos actúan más como mentores y facilitadores que como tomadores de decisiones jerárquicos. Su función es permitir que los equipos tomen decisiones arquitectónicas informadas, no dictarlas.

— Ambler, S. (2020). *Agile Modeling and Architecture Strategies*.

La arquitectura de software moderna consiste en alinear continuamente el diseño del sistema con los objetivos del negocio, bajo un cambio constante. Los arquitectos deben estar integrados en los equipos, tomando decisiones estratégicas en tiempo real.

— The New Stack (2022), *What Is the Role of Software Architect in an Agile World?*

En lugar de definir cada módulo o capa desde el inicio, los arquitectos en contextos ágiles definen principios rectores, límites y contratos que ayudan a los equipos a construir sistemas alineados y evolucionables.

— Red Hat (2021), *The Role of an Agile Architect*

El arquitecto como gestor del Cambio

Un arquitecto de software es gestor del cambio cuando sus decisiones estratégicas permiten que un sistema evolucione con seguridad, flexibilidad y coherencia, en entornos de incertidumbre tecnológica y organizacional.

1. **Cambio es inevitable:** el software moderno se encuentra en evolución constante por nuevas tecnologías, cambios de mercado, reorganización de equipos, etc.
2. **Cambio debe ser sostenible:** no toda evolución es positiva si genera deuda técnica o caos estructural.
3. **Cambio debe estar anticipado:** un buen diseño arquitectónico considera el cambio como una constante, no como una excepción.

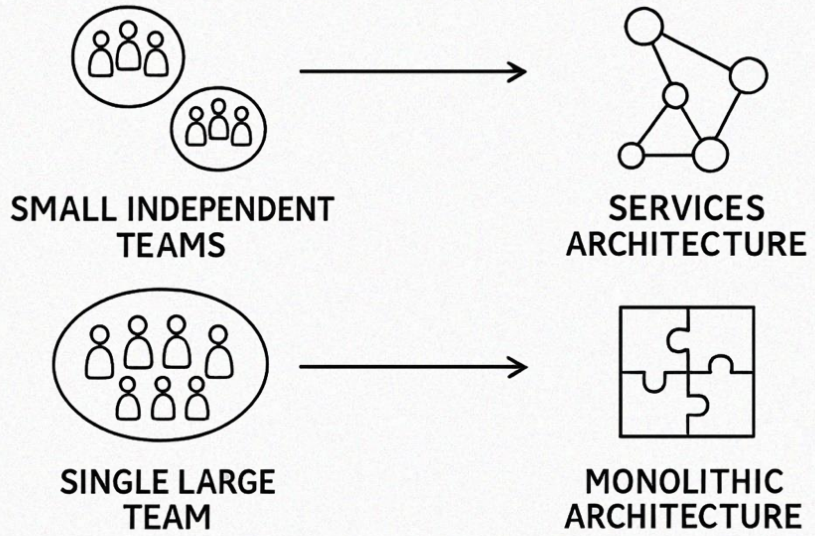
"Modern architecture is evolutionary architecture. It is defined not only by its structure but by how well it supports guided, incremental change."
— Ford, Parsons, Kua & McMullin (2017), *Building Evolutionary Architectures*

El arquitecto como estratega

Responsabilidades del arquitecto como estratega

- **Diseñar estructuras sociotécnicas** (no solo técnicas)
- Anticipar el impacto de las decisiones técnicas en la organización
- Alinear equipos con los límites de contexto (Domain-Driven Design)
- Facilitar cambios en cultura técnica (testing, CI/CD, documentación)
- Servir como **puente entre la dirección técnica y la dirección de negocio**

"¿Sabías que si decides usar microservicios… tal vez debas cambiar la estructura del equipo también?"



The Conway's Law

Diseño de Software
vs
Arquitectura de Software

¿Elegir una base de datos es arquitectura o diseño?

¿Cambiar una librería de validación es arquitectura o diseño?

¿Separar el sistema en módulos por dominio es arquitectura o diseño?

> El diseño de software abarca la construcción de módulos individuales y el diseño a bajo nivel de algoritmos y estructuras de datos.
>
> — Ian Sommerville, *Software Engineering*

> La arquitectura de software abarca el conjunto de decisiones significativas sobre la organización del sistema, incluyendo elementos estructurales, sus interfaces, comportamientos y composición.
>
> — Bass, Clements & Kazman

Diseño de Software

- Se enfoca en **detalles de implementación**: clases, funciones, algoritmos.
- Opera a un **nivel bajo o medio de abstracción**.
- Tiene **impacto localizado** en el sistema.
- Suele ser **modificable por un solo equipo**.
- Cambia con frecuencia durante el desarrollo.
- No siempre requiere coordinación inter-equipo.
- Aporta **eficiencia y legibilidad** al código.
- Se apoya en patrones de diseño (GoF, SOLID, etc.).

Arquitectura de Software

- Define la **estructura general del sistema** y sus componentes principales.
- Opera a un **nivel alto de abstracción**.
- Tiene **impacto estructural** y de largo plazo.
- Afecta a múltiples equipos y decisiones organizativas.
- Debe ser **estable pero adaptable**.
- Requiere **alineación con objetivos del negocio**.
- Asegura **calidad técnica** (escalabilidad, seguridad, mantenibilidad).
- Se expresa mediante vistas, estilos y decisiones arquitectónicas.

¿Un arquitecto debe saber programar?

¿Cuál es la diferencia entre un arquitecto y un líder técnico?

¿Quién cuida la arquitectura cuando no hay arquitecto? O ¿Todo equipo deberia tener un arquitecto?

Implicancias prácticas de una arquitectura mal definida

¿Qué pasa si un equipo no tiene arquitecto… o peor, si su arquitectura simplemente emerge sin nadie que la piense?

Sistemas difíciles de mantener

- Cambios simples (agregar un campo, modificar un flujo) requieren tocar múltiples archivos o capas no relacionadas.
- Ausencia de cohesión y alta dependencia entre módulos.

Deuda técnica estructural

- No se trata de código mal escrito, sino de **decisiones mal tomadas o nunca tomadas**.
- Afecta rendimiento, seguridad, escalabilidad.

Equipos mal alineados

- Si la arquitectura no refleja límites de responsabilidad claros, los equipos se pisan entre sí.
- Sin contratos técnicos, surgen conflictos de integración constantes.

Without a coherent architecture, a system rapidly degenerates into a collection of disconnected patches."

— Bass et al., *Software Architecture in Practice*

"Structural debt is silent, accumulates invisibly, and usually explodes at scale."

— Ford, N. et al(2017). *Building evolutionary architectures*

Pérdida de velocidad en el desarrollo

- Tiempos de entrega largos debido a "re-trabajo" y regresiones constantes.
- Cada sprint involucra apagar incendios arquitectónicos.

Incapacidad para escalar

- El sistema no aguanta carga, ni en usuarios ni en funcionalidades.
- Cada nuevo requerimiento requiere una reestructuración mayor.

Una arquitectura mal definida ***no siempre se nota al inicio****, pero* ***siempre cobra factura con el tiempo****.*

Y lo más peligroso no es el mal diseño… ***es la ausencia de decisiones, y la ilusión de que "ya resolveremos cuando escale".***

Caso de estudio:
HealthCare.gov
Autopsia de una arquitectura mal diseñada

HealthCare.gov

Healthcare.gov es el portal web creado por el gobierno de Estados Unidos para facilitar la inscripción en planes de seguro médico bajo la Ley de Cuidado de Salud Asequible (Affordable Care Act, ACA), también conocida como Obamacare, promulgada en 2010. Este sitio, gestionado por los Centros de Servicios de Medicare y Medicaid (CMS), parte del Departamento de Salud y Servicios Humanos (HHS), tenía como objetivo ser un mercado en línea donde los ciudadanos pudieran comparar y adquirir seguros médicos, además de verificar su elegibilidad para subsidios basados en ingresos. El sitio fue lanzado oficialmente el 1 de octubre de 2013, cubriendo a los residentes de 36 estados que no crearon sus propios mercados de seguros.

Se esperaba que soportara un alto volumen de usuarios, dado que la ACA requería que todos los ciudadanos tuvieran seguro médico o enfrentaran una penalidad fiscal.

HealthCare.gov

- Registrar online personas (algunos de sus datos básicos)
- Capturar información de esas personas en diferentes bases de datos de diferentes organizamos del gobierno central
- Generar un perfil Enviar a compañías de seguro
- Recibir alternativas de planes y seguros (marketplace de seguros)

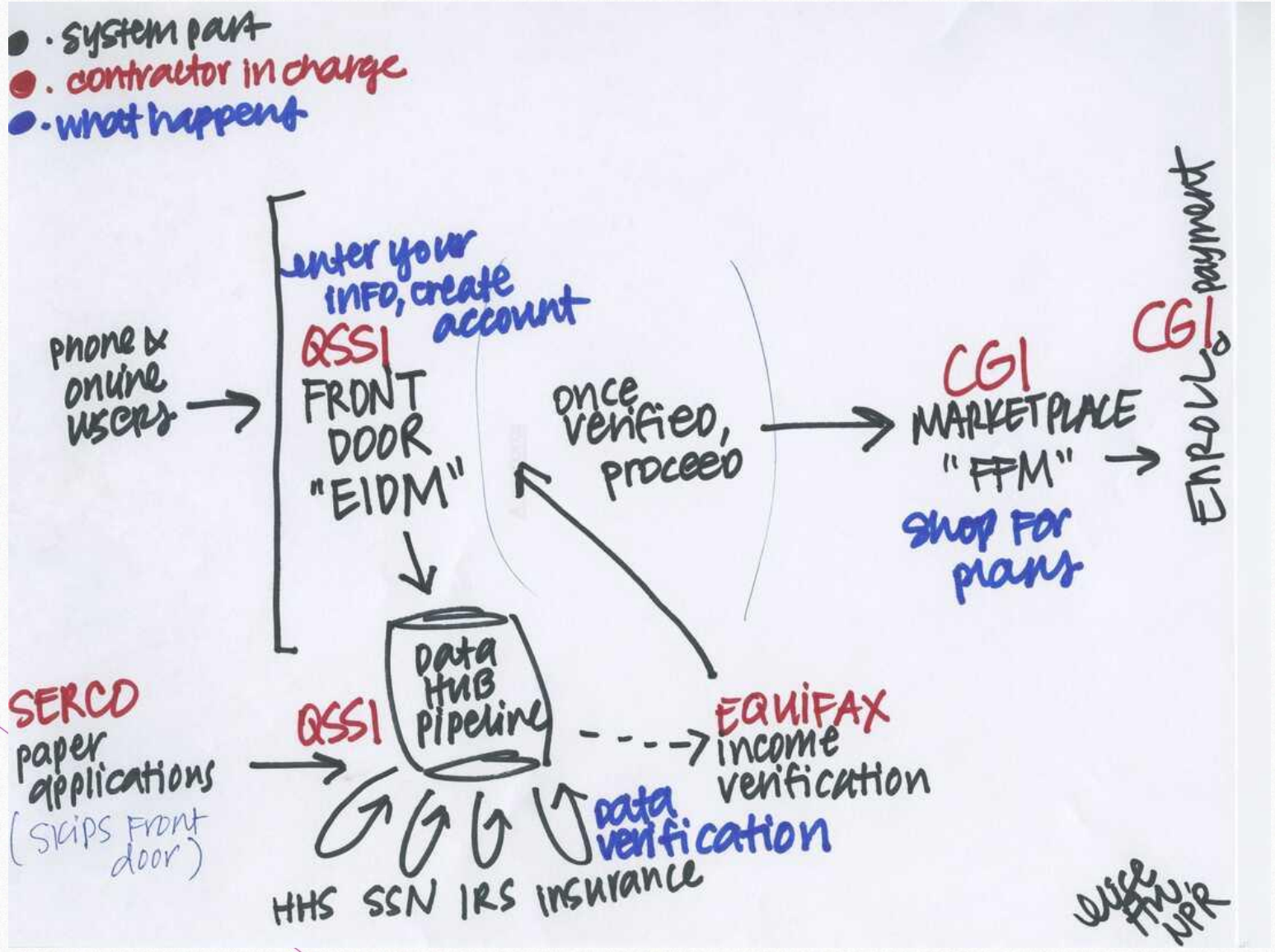
Métrica	Valor
Monto Total del Proyecto (subcontratos)	US$ 323.000.000
Costo gestión de Proyecto (PMO)	US$ 22.000.000
Cantidad de contratistas involucrados	55
Tamaño del Software (LOC)	500.000.000 LOC

Problemas en el lanzamiento:

¿Qué creen que pasó aquí?

Fallas técnicas masivas:

- El sitio colapsó en menos de dos horas tras su lanzamiento debido a un volumen de usuarios mucho mayor al esperado (250,000 usuarios simultáneos, cinco veces más de lo previsto).
- Solo seis personas en todo el país lograron inscribirse en un plan de salud el primer día. En los primeros días, con 8 millones de visitantes, solo el 1% completó el proceso de inscripción.
- Problemas específicos incluyeron menús desplegables incompletos, datos de usuarios enviados incorrectamente a las aseguradoras y un sistema de inicio de sesión que no podía manejar el tráfico.



Basado en testimonios de contratistas

Falta de liderazgo claro

El desarrollo de Healthcare.gov involucró a múltiples entidades, incluyendo el Departamento de Salud y Servicios Humanos (HHS), los Centros de Servicios de Medicare y Medicaid (CMS), la Oficina de Gestión y Presupuesto (OMB) y más de 55 contratistas privados, como CGI Federal, que lideró gran parte del desarrollo del sitio. Sin embargo, no había una autoridad central que coordinara eficazmente a todas las partes.

Cada entidad operaba con objetivos y prioridades propias, lo que generó conflictos y falta de alineación. Por ejemplo, el CMS, encargado de supervisar el proyecto, carecía de experiencia en gestionar proyectos tecnológicos de esta magnitud, y su enfoque estaba más orientado a políticas de salud que a tecnología.

En lugar de un liderazgo técnico sólido, las decisiones a menudo recaían en funcionarios de alto nivel con poca experiencia en desarrollo de software, lo que resultó en una supervisión inadecuada.

¿Qué creen que pasó aquí?

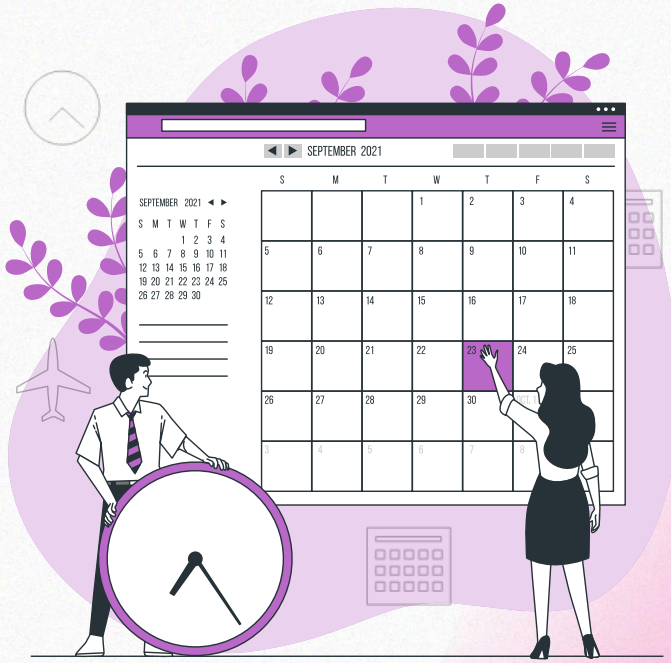
Planificación deficiente:

Durante el desarrollo del sitio, puestos críticos, como el de un director de tecnología (CTO) para el CMS o un gerente de proyecto con experiencia en TI a gran escala, estaban vacantes o no se definieron claramente. Esto dejó al proyecto sin un líder técnico que pudiera tomar decisiones rápidas y basadas en datos.

Se recibieron 18 advertencias escritas durante los dos años previos al lanzamiento, incluyendo informes de consultores externos que señalaban riesgos como planificación inadecuada y desviaciones de estándares de TI. Sin embargo, no se consideró posponer el lanzamiento

La presión por cumplir con la fecha de lanzamiento establecida por la ACA (1 de octubre de 2013) llevó a un desarrollo apresurado, con poco tiempo para pruebas y corrección de errores.

Los contratistas trabajaban con plazos vagos y expectativas poco realistas. Por ejemplo, las pruebas de integración (que verifican cómo los diferentes componentes del sitio funcionan juntos) se realizaron solo semanas antes del lanzamiento



Problemas de escalabilidad y diseño:

Las pruebas de carga, que evalúan la capacidad del sitio para manejar grandes volúmenes de usuarios, fueron insuficientes.

Las pruebas de integración también fueron limitadas, lo que permitió que errores críticos, como menús desplegables incompletos y datos enviados incorrectamente a las aseguradoras, pasaran desapercibidos hasta el lanzamiento.

El sistema dependía de servidores físicos con capacidad fija, lo que limitaba su flexibilidad.

El sistema de inicio de sesión era especialmente problemático, ya que no podía manejar múltiples solicitudes simultáneas

Las pruebas de no simularon escenarios realistas. Por ejemplo, no se probaron picos de tráfico cercanos a los 250,000 usuarios simultáneos.