1. Introducción

¿Por qué este ramo?

- Aumenta la complejidad del software actual.
  - Sistemas distribuídos, con escalabilidad masiva, sobre diversas plataformas, transparente para el usuario, ...
- Existe conocimiento de arquitecturas de referencia, frameworks y patrones para usar como base y mejorar la calidad del diseño.
- Garantizar calidad, reducir costos, permitir flexibilidad.

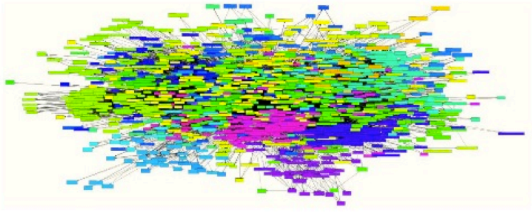
Motivación

- Típicamente los sistemas desarrollados atienden intereses puntuales.
  - Lista de requisitos.
  - Modelo de dominio.
  - Desarrollo de software.
- Aparecen cambios!
  - Modificaciones, adaptaciones, extensiones, parches, …
- Resultado : Big ball of Mud

Big ball of mud

JDK 1.5

- 1315 classes in 229 packages all depend on each other



Problemas de la Ingeniería de Software

- Accidentales.
  - Existe una solución (que espera ser descubierta).
  - Mejora de productividad.
    - Programación y abstracciones.
      - Lenguajes de programación de alto nivel.
    - Resultados de las decisiones de programación toma mucho tiempo.
    - Programas heterogéneos.
      - IDEs (Integrated Development Environment).
      - Apoyo al desarrollo.

Problemas de la Ingeniería de Software

- Esenciales.
  - Solo existen soluciones parciales.
    - Complejidad.
      - Crece de manera no lineal.
    - Conformidad.
      - Sistema operativo, hardware, …
    - Cambiabilidad.
      - Nuevas aplicaciones, usuarios, máquinas, estándares, leyes, hardware, …
    - Intangibilidad.
      - No hay leyes físicas, no hay una presentación obvia.

Atacando la complejidad

- Procesos de desarrollo ágil:
  - Iterativo : En cada iteración se corrigen errores.
  - Incremental: En cada iteración se construye un nuevo incremento funcional.
- Y el diseño arquitectural?
  - Crece la complejidad.
  - "Se puede enseñar buen diseño, pero no gran diseño"

Arquitectura de software

- Conjunto de las principales decisiones de diseño sobre el sistema.
  - Estructura.
  - Conducta.
  - Interacción.
  - Propiedades no funcionales.
- Principal: Decisiones que afectan a todo el sistema.

Similar (pero diferente) a la Arquitectura en Ing. Civil

- Requisitos, planos, construcción, uso.
- Satisfacción del cliente.
- Tareas especializadas.
- Puntos de revisión.
- El proceso no es tan importante como la arquitectura.
- Pero el software es más maleable que el material sólido.
- El software tiene un aspecto temporal y cambiará.

Arquitectura de Software

- No se especifican detalles –de implementación– pero si los componentes que son cargan con la responsabilidad de la implementación.
  - Componentes de grano grande y subsistemas, no clases y algoritmos.
- Va del análisis a la implementación.
- Objetivo: Manejar la complejidad.
- Tomar decisiones que afectan a todo el sistema.

Arquitectura de software

Arquitectura de software = {Elementos, *qué* Forma, *cómo* Lógica *por qué* }

- Involucra:
  - Elementos a partir de los cuáles se construye un sistema.
  - Interacciones entre esos elementos.
  - Patrones que guían su composición.
  - Restricciones sobre esos patrones.
  - Abstracciones, de-composición, composición, estilo y *estética*.

Arquitectura de software

- "La organización fundamental de un sistema, reflejada en sus componentes, relaciones entre ellos y el ambiente, y los principios que gobiernan su diseño y evolución."

Recommended Practice for Architectural Description of Software-Intensive Systems
ANSI/IEEE Std 1471-2000

- Captura la **estructura** del sistema en términos de componentes y cómo interactúan.
- Define reglas de diseño y evolución globales al sistema.
- Componentes, modulos, objetos o cualquier otra unidad de software interrelacionada.

Contenidos y plan de curso

Mes	Tópico	Lectures
Agosto	Conceptos básicos	Introduccion, Análisis, Atributos de calidad de software, Conceptos básicos
	Análisis y Diseño	Diseño de arquitecturas, Conectores, Modelación y visualización
Septiembre	Diseño	Estilos arquitectónicos: Cliente/Servidor, Pipe&Filters, Shared memory, Event-based, Message Oriented Middleware
		Implementación
		Patrones de arquitecturas empresariales (Dominio, Data y Presentación)
Octubre	Casos de Estudio	Arquitecturas aplicadas: REST (Web), Peer 2 Peer (Skype), Centrada en los datos (Facebook)
	Evaluación	Evaluación
	Estrategias avanzadas	Model-Driven architecture, Software product line, Cloud
Noviembre	Conceptos avanzados	Reusabilidad, Concurrencia, Estado y Distribución, Escalabilidad masiva

Tareas

- Construir un prototipo siguiendo un estilo arquitectónico predeterminado.

- Tarea1: Requisitos no funcionales.
- Tarea2: Diseño conceptual 1.
- Tarea3: Prototipo 1.
- Tarea4: Diseño conceptual actualizado.
- Tarea5: Prototipo 2.
- Tarea6: Evaluación.

Evaluación

- Teoría (NT)
  - 3 Interrogaciones (30%).
  - Examen final (obligatorio) (30%).
- Práctica (NP)
  - 6 Tareas, en grupos de a ~4 (mini-proyecto) (40%).
    - **Evaluación individual.**
- NE≥4.0, NP≥4.0, NT≥4.0

Bibliografía

- Richard N. Taylor, "Software Architecture, Foundations, Theory and Practice", Wiley, 2010.
- Spinellis & Gousios, "Beautiful Architecture", O'Reilly, 2009.
- Ian Gorton, "Essential Software Architecture", Springer, June, 2006.
- Martin Fowler et al., "Patterns of Enterprise Application Architecture", Addison Wesley, 2002.

1. Introducción

Diseño arquitecturas.

Requisitos No Funcionales.

Diseño de la Arquitectura de Software

Tradicionalmente: Análisis no debe mancharse con consideraciones de diseño.

Plan, costo, riesgo?

1. Identificar requisitos no funcionales (-ilities) y restricciones.
2. Diseñar y/o elegir un framework / metáfora / estilo.
3. Validar.
4. Documentar desiciones de diseño.

Arquitectura de Software

- No es un capricho tecnológico: **Requisitos (No Funcionales)** y Restricciones.

Atributo de Calidad	Requisito Arquitectónico
Desempeño	La aplicación debe entregar tiempos de respuesta menores a 4 segundos para el 90% de las solicitudes.
Seguridad	Toda la comunicación debe ser autenticada y encriptada usando certificados.
Gestión de recursos	El servidor debe correr como un proceso no prioritario en un PC con 512MB de memoria.
Usabilidad	La interfaz de usuario debe correr en un browser de Internet para los usuarios remotos.
Disponibilidad	El sistema debe correr 24x7x365, con una disponibilidad promedio de 0.99.
Confiabilidad	No se acepta pérdida de mensajes, el resultado de la entrega de mensajes debe conocerse en 30segs.
Escalabilidad	La aplicación debe manejar un carga peak de 500 usuarios concurrentes durante el periodo de inscripción.
Flexibilidad	La arquitectura debe permitir una migración en fases del lenguaje 4GL a .NET

Arquitectura de Software

- Restricciones: Técnicas o tecnológicas, de negocio, de calidad.

Restricción	Requisito Arquitectónico
Negocio	La aplicación debe correr como un plugin para MS BizTalk (queremos venderselo a MS).
	Queremos aliarnos (y obtener fondos) de Xcorp, necesitamos usar su tecnología.
Desarrollo	El sistema debe estar escrito en Java (la mantención y testing lo hará nuestro equipo).
Tiempos	La primera version debe entregarse en 6 meses.

Atributos de calidad

- Performance – Desempeño

  Carga de trabajo que una aplicación debe ejecutar en una unidad de tiempo, y/o deadlines que debe cumplir para una correcta operación.

  Carga de trabajo: Transacciones, mensajes, lectura a la DB, actualización de una DB distribuída en 2 fases.

  - Throughput – Salida: Carga de trabajo en una unidad de tiempo (transacciones por segundo - tps). Peak y promedio.
  - Tiempo de respuesta: Latencia que una aplicació exhibe al procesar una carga de trabajo. Garantizado (límite) y promedio.
  - Deadline – Plazo: Ventana de tiempo para la compleción de una carga de trabajo.

Atributos de calidad

- Escalabilidad

  ¿Qué también trabaja una solución cuando el tamaño del problema aumenta?

  - Request – Cantidad de Solicitudes: Multi-thread, balance de carga (repartir las solicitudes entre varios servidores/CPUs).
  - Cantidad de Conexiones concurrentes
  - Tamaño de datos: Cantidad de datos que puede procesar la aplicación en cada transacción (incremento en el tamaño de los paquetes de datos).
  - Deployment – Implantación: Cantidad de esfuerzo requerido en distribuir, actualizar, configurar una aplicación (incremento en cantidad de usuarios).

Atributos de calidad

- Flexibilidad

  Esfuerzo requerido para cambiar una aplicación (modificar funcionalidad o arquitectura).

  - Requiere identificar escenarios de cambio probables.
  - Cohesión dentro los componentes y subsistemas débilmente acoplados facilitan el cambio.

- Seguridad
  - Autenticación, autorización, encriptación, integridad, no repudiación.

- Disponibilidad

  Porcentaje de tiempo en que una aplicación puede ser usada.

  - Recuperación ante fallas (automática o manual – tiempo de respuesta o reacción) … Sábados? Vacaciones?, 3am?

Atributos de calidad

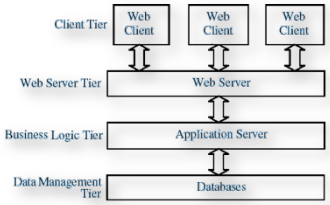
- **Integración**

  Esfuerzo requerido para integrar una aplicación en un contexto mayor.

  - A nivel de datos (DB accesible a otras aplicaciones).
    - Flexible, simple y peligrosa.
  - API (Application Programming Interface).
    - Conjunto de funciones (parametrizadas) que controlan el acceso a los datos. Garantizan integridad de datos y reglas de negocio.
    - Compleja, costosa y segura.

N-Capas

- Modularidad: Separación de temas.
- Comunicación sincrónica entre capas (request – wait – answer).
- Implantación flexible: capas independientes, distribuidas, heterogéneas, etc.
- Gran número de clientes, solicitudes concurrentes, requiere poco tiempo de proceso.



Cliente-Servidor n-capas

Atributo de Calidad	Ventajas
Disponibilidad	Réplica de servidores en cada capa, si uno falla, los otros responden. La QoS disminuye hasta restaurar el servidor fallado.
Manejo de fallas	La mayoría de los servidores web y de aplicación implementan una recuperación transparente (el cliente se redirecciona a un servidor replicado para responder a la solicitud).
Flexibilidad	La modularidad facilita el cambio, debido la encapsulación (presentacion, negocios y datos). El impacto del cambio al interior de una capa debería ser mínimo en las otras.
Desempeño	Alto desempeño mediante replicaccción de software (threads concurrentes), velocidad de la conexión entre capas y la cantidad de datos a ser transferidos. Es recomendable minimizar las llamadas entre capas al procesar cada solicitud.
Escalabilidad	Como los servidores pueden replicarse, la arquitectura escala bien. En la práctica, la gestión de datos se convierte en el cuello de botella.