Resumen Arquitectura de Software - 1er Parcial

# Definiciones de Arquitectura de Software:

* “Es el concepto de mayor alto nivel de un sistema en su ambiente” **(IEEE Architecture Working Group)**.
* As = {*Elementos* (QUÉ), *Forma* (CÓMO), *Restricciones* (POR QUÉ)} **(Perry 92).**
* “La arquitectura de software se ocupa: **(Kruchten 95)**
  + Del diseño e implementación de la estructura de más alto nivel del sistema.
  + Abstracción, descomposición, composición y estilo”.
* La arquitectura de un programa o sistema de computación es la estructura o estructuras del sistema, que comprenden sus componentes de software, las propiedades externas de los componentes, y la relación entre ellos”. **(Kazmann)**
  + Más de una estructura
  + Información sobre los componentes y como interactúan entre ellos y con su entorno

# Conceptos generales:

* La arquitectura de un sistema se focaliza en:
  + Elementos estructurales de alto nivel.
  + Elementos que tienen impacto en la perfomance, confiabilidad, adaptabilidad, integrabilidad, costo, etc.
  + En la selección de plataformas, sistemas operativos, middleware, dbms, etc.
* La arquitectura del sistema se describe utilizando diagramas que cubren aspectos de:
  + funcionalidad y restricciones
  + diseño lógico
  + diseño físico
  + topología del hardware
  + otros

# Beneficios de la Arquitectura

* Comunicarse con los accionistas
  + Accionistas: Cliente, Usuario, Gerente de Proyecto, Desarrolladores, Arquitecto y otros.
    - Cada uno tiene intereses y visiones distintas sobre el sistema.
    - La arquitectura establece un lenguaje común.
* Comprender el sistema
  + Los sistemas de Software son:
    - Funcionalmente complejos
    - Involucran tecnologías complejas
    - Fácilmente cambiables
* Permite realizar decisiones estratégicas
  + Estrategia vs Táctica
    - Estratégica -visión a largo plazo
    - Táctica -visión a corto plazo
  + La arquitectura permite validar tempranamente los atributos de calidad del sistema.
* Organizar el desarrollo
  + La arquitectura divide el sistema en sus componentes y sus interfaces. (PBS)
  + Permite obtener tempranamente la división de las actividades del grupo de desarrollo (WBS).
  + Permite realizar estimaciones tempranas de esfuerzo y duración.
* Promover el reuso
  + La arquitectura permite:
    - determinar tempranamente componentes candidatos a ser reusados o comprados.
    - reutilizar la arquitectura en sistemas similares. (experiencia del arquitecto)
    - establecer líneas de productos
* Evolucionar el sistema
  + La arquitectura permite determinar el impacto de los cambios.
  + Aislar los potenciales centros de cambios del sistema.
  + Una buena arquitectura absorbe los cambios que se producen durante el ciclo de vida del producto

# Requerimientos

El diseño de la arquitectura de software comprende:

* Definir la colección de componentes que satisfagan el conjunto de funciones (requisitos funcionales) y atributos de calidad (requisitos no funcionales) definidos para el sistema. (Requerimientos de arquitectura)

# Requerimientos Funcionales

Especifican los servicios que el sistema debe proveer.

Para resolverlos se utilizan estilos de diseño y principios de forma de aislar los componentes que proveen dichas funcionalidades.

* Separación de intereses
* SRP
* Ley de Demeter
* los principios vistos en DA
* Evitar repetición de responsabilidades (DRY)

# Requerimientos No Funcionales

* Requerimientos técnicos
  + Sistema operativo
  + Bases de Datos
  + Lenguaje de programación
  + Etc.
* Características de calidad

Para resolveros se utilizan estilos y mecanismos o tácticas de arquitectura

* Clustering
* Gestión de recursos
* Autenticación
* Logging
* Etc.

# Atributos de calidad del Sistema

Los atributos de calidad al igual que los RFs se afectan a las distintas partes de un sistema.

Existe dependencia entre los atributos de calidad por lo que es imposible lograrlos todos para un sistema, es necesario priorizar.

* **Confiabilidad**
  + ***Tolerancia a fallas (Disponibilidad)***
    - Capacidad de mantener el nivel de performance especificado en caso de fallas en el software
    - Trata con las fallas del sistema y sus consecuencias asociadas. (Falla: es observable por el usuario, Defecto: es la causa de la falla. Puede recuperarse sin que se haga observable)
  + ***Capacidad de recuperación (Recoverability)***
    - Capacidad e restablecer el nivel de performance y los datos involucrados
* **Modificabilidad (Escalabilidad)**
  + Es la capacidad del sistema de permitir realizar cambios (anticipados o no anticipados) con bajo impacto.
  + Que puede cambiar
    - Código / funcionalidad.
    - Plataforma (Hardware, SO, Middleware): Portabilidad.
    - Atributos de calidad (eficiencia, seguridad, etc).
    - Capacidad (usuarios, procesamiento): Escalabilidad.
* **Eficiencia**
  + Es la cualidad que refleja la habilidad del sistema de lograr los requerimientos de performance al tiempo que minimiza los recursos de su ambiente computacional (R. Taylor).
  + Refiere al tiempo requerido para responder a determinados eventos o al número de eventos procesados en un intervalo dado de tiempo.
  + Subatributos
    - ***Latencia***: Especifica el tiempo requerido por parte del sistema para responder al algún estimulo.
    - ***Capacidad***: Especifica cuantos usuarios, terminales, nodos, registros, etc. debe soportar el sistema manteniendo las restricciones de tiempo.
    - ***Troughput***: Especifica cuantos eventos puede responder el sistema en un intervalo dado de tiempo
* **Seguridad**
  + Es la habilidad del sistema de resistir intentos de uso no autorizados mientras que se continua ofreciendo los servicios a los usuarios legítimos.
  + Ataque o amenaza: Intento de violar la seguridad
  + ***No repudiación***: Implica que una transacción no pueda negarse por ninguna de las partes involucradas.
  + ***Confidencialidad***: Datos o servicios están protegidos del acceso no autorizado.
  + ***Integridad***: Datos o el servicio se entregan según lo acordado.
  + ***Aseguramiento***: Implica que se garantice la identidad de quienes participan en una transacción.
  + ***Disponibilidad***: Un sistema esta disponible para su uso legitimo.
  + ***Auditoria***: El sistema lleva registro de sus actividades lo que permite reconstruir una transacción.
* **Usabilidad**
  + El esfuerzo requerido por parte del usuario para realizar sus tareas con el sistema.
  + El tipo de asistencia que brinda el sistema a los usuarios.
* **Testeabilidad**
  + Facilidad con la cual se puede demostrar las fallas de un sistema de software a partir de pruebas (testing)
  + Testing representa el 40% del costo de desarrollo de un sistema.
  + Sistema testeable: debe ser posible controlar el estado interno y entradas de cada componente y observar las salidas/resultados.
  + Suelen utilizarse herramientas, librerías externas o incluso metodologías especificas para realizar las pruebas de software.
* **Portabilidad**
  + Capacidad para transferir un producto de software de un ambiente a otro.
    - El ambiente puede ser organizacional, hardware o software.

# Estilos

* Un estilo de arquitectura es una descripción de tipos de componentes y de sus patrones de control y transferencia de datos en tiempo de ejecución (bass).
* Están determinados por:
  + Un conjunto de tipos de componentes.
  + La distribución topología de los componentes indicando sus relaciones en tiempo de ejecución.
  + Un conjunto de conectores que realizan la comunicación, coordinación entre los componentes.
* Normalmente un sistema se compone de la combinación de varios estilos.
* Los componentes y conectores de un estilo normalmente pueden describirse utilizando otros estilos.
* Cada estilo es más apropiado para satisfacer algunos atributos de calidad.

## Estilo Repositorio

Descripción del Problema: Productores y Consumidores de Datos compartidos no conocen los detalles de implementación del otro.

Características:

* Utiliza un mediador.
* Se reduce el acoplamiento entre productores y consumidores.



Los clientes:

* Ejecutan en threads o procesos de control independientes.
* Pueden ser productores, consumidores o ambos.
* Conocen la estructura del repositorio.

El repositorio

* Puede estar en memoria o disco (bd o archivos).
* Debe tener un esquema especifico y conocido por los clientes.
* Es pasivo.

**Motivo: Almacenar grandes volúmenes de información.**

**Cualidades: Escalabilidad, Mantenibilidad, Confiabilidad**

**Vista: Componentes y Conectores**

**Implementación: Gestor de Base de Datos**

## Estilo Repositorio Abstracto

Las diferencias con el anterior son:

* Esconde a los clientes los detalles de implementación del repositorio mediante una interfaz abstracta.
* Los clientes NO conocen la estructura del repositorio.
* El repositorio provee interfaces para acceder al mismo.



## Estilo –Capas (lógicas)

Descripción del Problema: Se utilizan para proveer niveles acumulativos de abstracción. Cada nivel oculta los detalles de implementación a las capas adyacentes.

Características:

* Util en aplicaciones que involucran distintas clases de servicios que se pueden agrupar en forma jerárquica.
* Se quiere:
  + **Separar interfaces/organizar codigo**
  + localizar los cambios para minimizar su impacto
  + Separar las responsabilidades entre módulos
  + Minimizar el acoplamiento entre módulos
  + Reusar módulos

Los componentes en cada capa:

* deben pertenecer al mismo nivel de abstracción
* deben ser cohesivos

Los elementos de una capa se comunizan con sus pares y con elementos de las capas subyacentes:

* Estricto –solo con capas inmediatamente inferiores
* Relajado –con capas de cualquier nivel inferior

Las capas se deben comunicar mediante interfaces bien definidas

Invocaciones:

* De arriba hacia abajo
  + La interacción comienza con la capa superior y se propaga a las capas inferiores (situación más común)
* De abajo hacia arriba
  + La interacción comienza en la capa de menor nivel de abstracción y se propaga hacia las capas superiores (por ejemplo captura de eventos de bajo nivel)

Heurísticas de diseño

* Las capas deben agrupar funcionalidades de la solución en niveles de servicios bien definidos
* Se deben definir la interfaces de cada capa en función de las necesidades de la capa superior
* Las interfaces se pueden implementar utilizando distintos tipo de conectores
  + Llamas a procedimientos, mensajes, WS, callbacks, etc.

**Cualidades: Favorece la mantenibilidad, reuso de paquetes, portabilidad, testeabilidad, división del trabajo.**

**Vista: Lógica**

**Implementación: Paquetes/namespaces**



## Estilo –Tiers (capas físicas)

Descripción del Problema: Se quiere distribuir el procesamiento de los componentes de una solución entre distintos procesadores.



Se opta por este estilo cuando un sistema tiene categorías de funcionalidad bien definidas que:

* Son internamente cohesivas.
* Estables con respecto al cambio.
* Dependen entre ellas en forma predecible.
* No tienen dependencias cíclicas (esto es deseable)
* Tiene bajo acoplamiento con otras capas.

**Motivo: Distribuir componentes en el hardware**

**Cualidades: Eficiencia, Reuso, Mantenibilidad.**

**Vista: Componentes y Conectores, Despliegue**

**Implementación: JEE**

## Publicador/Subscriptor

Descripción del Problema: Es necesario sincronizar automáticamente el estado de productores (Publicadores) y consumidores (Suscriptores) de datos.

Características:

* Puede utilizar un mediador (o las funciones pueden estar distribuidas entre los propios consumidores y productores).
* Cuando un publicador publica un dato se informa a todos los suscriptores (o a los interesados en ese dato) y reciben el dato.
* Se reduce el acoplamiento entre productores y consumidores.

Los clientes:

* Ejecutan en threads o procesos de control independientes.
* Pueden ser productores, consumidores o ambos.
* No conocen la estructura del repositorio.

El Mediador:

* Provee interfaces para publicar y suscribirse.
* Puede utilizar reglas particulares para notificar a varios suscriptores interesados en el mismo dato.
* La notificación puede ser sincrónica o asincrónica

Basados en listas (tópicos)

* Los suscriptores se suscriben a un tópico (subject)
* El publicador o el mediador:
  + mantienen una lista de suscriptores
  + Proveen interfaces para suscribirse o borrarse de la lista
  + Cuando se recibe o genera un mensaje (evento) con el tópico se envía a todos los suscriptores de la lista
* El suscriptor
  + Provee una interfaz para ser notificado del evento



Basados en Broadcast (tópicos)

* El publicador o el mediador:
  + Cuando se recibe o genera un mensaje (evento) con el tópico se envía a la red
* Todos los suscriptores
  + Reciben los mensajes y deciden si lo procesan basándose en el tópico

**Motivo: Notificar eventos que se publican a suscriptores sin que se conozcan**

**Cualidades: Mantenibilidad (separar responsabilidades), escalabilidad, eficiencia (sincronización)**

**Vista: Componentes y Conectores**

**Implementación: JMS (Tópicos), WCF/Biztalk**