***Arquitectura de Software.***

La arquitectura de software de un sistema, define de manera abstracta, el conjunto de estructuras que lo componen. Estas estructuras son elementos de tecnologías, relaciones y propiedades entre ellas. Es por esta razón, que la arquitectura trata de omitir ciertos detalles internos de cada uno de estos elementos, para abstraerse de su dificultad, y se ocupa más bien de lo exterior.

Interfaces: dividen lo privado de lo público, se centra en la complejidad de la interacción de los elementos.

Objetivos:

Los sistemas de SW son construidos para satisfacer los objetivos del negocio.

Importancia:

Usuario: el sistema cumpla con rapidez, disponibilidad, confiable.

Cliente: arquitectura seleccionada sea implementada de acuerdo al calendario y presupuesto.

Project Manager: arquitectura permita la independencia de los equipos que interactúen con disciplina.

Arquitectura: puntos anteriores funcionen correcta y sincrónicamente.

Interesados:

Clientes, Usuarios, Project Managers, Arquitectos, Desarrolladores, Testers, Subsistemas, otros.

Decisiones de diseño:

Monoprocesador vs Multiprocesador

SW en capas

Comunicación de componentes sincrónica o asincrónica

Sistema dependiente de características de HW y/o Sistema Operativo.

Contexto:

Técnico, Ciclo de vida del proyecto, Negocio, Profesional.

A pesar de que estos contextos no cambian, las especificaciones del sistema cambian constantemente. Uno de los desafíos del arquitecto es prever que estos cambios no repercutan en gran medida sobre los sistemas.

Actividades importantes en la creación de la Arquitectura de Software:

1. Crear casos de negocio para el sistema.

2. Entender, los requerimientos importantes para la arquitectura.

3. Crear o seleccionar una arquitectura.

4. Documentar y comunicar la arquitectura.

5. Analizar y evaluar la arquitectura.

6. Implementar y testear el sistema, basándose en la arquitectura.

7. Asegurarse que la implementación conforme a la arquitectura elegida.

**Atributos de Calidad:**

Propiedad de media o testeo que permite indicar qué tan bien funciona un sistema y como satisface las necesidades.

**Requerimientos**:

Funcional: qué tiene que hacer el sistema y cómo debe reaccionar o actuar en tiempo de ejecución.

Calidad de Sistema: definen las características de los requerimientos funcionales a lo largo de todo el producto.

Restricciones: decisión de diseño que debe tomarse (lenguaje de programación, reutilización de código)

**Orientar las decisiones de Diseño de Calidad:**

La arquitectura puede verse como el resultado de aplicar un conjunto de decisiones de diseño. Pueden dividirse en 7 secciones: Asignar responsabilidades, modelo de coordinación, modelo de datos, gestión de recursos, mapeo entre los elementos, decisiones sobre los tiempos, elección de tecnologías.

**Disponibilidad**:

Tiene que ver con minimizar las interrupciones de servicio y mitigar las posibles fallas que puedan ocurrir.

Las pérdidas significan desviaciones de las especificaciones del sistema, dónde esta desviación es externamente visible. La idea es entender el origen del problema para tener una estrategia de mitigación dentro del sw.

Técnicas: Detectar fallas, recuperación de fallas, prevención de fallas.

**Interoperabilidad**:

Que 2 o más sistemas puedan intercambiar información importante vía interfaces pudiendo entender dicha información.

Categoría “Locate”, táctica “discover service”: sistemas descubiertos en tiempo de ejecución.

Categoría “Manage Interface”:

Táctica “Orchestrale”: se utiliza un mecanismo de control que coordina, gestiona y secuencia la invocación de servicios particulares

Táctica “Tailor interface”: agrega o elimina capacidades en una interface.

**Adaptabilidad:**

Tiene que ver con el cambio, centrando el mismo en el costo y en el riesgo de realizarlo.

Consideraciones: ¿qué puedo cambiar?, ¿probabilidad del cambio?, ¿costo?, ¿cuándo y por quién?

**Performance**:

**Habilidad** que tienen los sistemas para contestar los requerimientos en **tiempo y forma**. Cuando ocurre un evento, el sistema debe responder a tiempo

Táctica “Demanda de control de recursos”: lado demanda, para producir menos demanda de los recursos que deben servir a eventos.

Táctica “Gestión de recursos”: lado repuesta, para que los recursos trabajen más eficientemente en gestionar las demandas que se suscriben sobre ellos.

**Seguridad**:

Proteger el acceso datos e información de un acceso no autorizado, mientras se provee acceso a gente y sistemas que sí está autorizados.

Tácticas: **Detectar Ataques** (la detección de intrusos, la detección de la denegación de un servicio, la verificación de integridad de mensajes, y la detección del atraso de mensajes) y **Resistir Ataques** (identificar actores).

**Reaccionar a un ataque**: revocar servicio (limitar accesos si se cree que puede originarse un ataque) o lockeo de computadora (bloquear pc tras varios intentos fallidos).

**Recuperación a un ataque**: consiste en restaurar los servicios dañados (servicios adicionales o conexiones de redes son reservados para este propósito).

**Capacidad de Prueba y Testeo:**

El objetivo de la táctica de la capacidad de prueba, es permitir un testeo sencillo cuando se completa un incremento del desarrollo de software.

Categorías: controlar y observar el sistema; limitar la complejidad del diseño del sistema.

**Usabilidad**:

Cuán fácil es para el usuario ejecutar una tarea deseada y cuál es el tipo de usuario del sistema. Es una de las formas más fáciles y baratas de mejorar la calidad de un sistema.

Características: aprendizaje, utilización eficiente, minimizar impacto de errores, adaptación a necesidades del usuario, incrementar confianza y la satisfacción.

**Iniciativa de apoyo al usuario:**

Cuando el sistema es ejecutado, se mejora la facilidad de uso brindando feedback de los usuarios de que es lo que el sistema está haciendo y habilitando al usuario a tener respuestas apropiadas.

Por ejemplo, las tácticas de “cancel”, “undo”, “pause”, “resume” y “aggregate” ayudan al usuario a corregir los errores y ser más eficiente.

**Iniciativa de apoyo al sistema:**

Cuando el sistema toma la iniciativa, debe basarse en un modelo del usuario, donde la tarea que se lleva a cabo, es por el usuario o el propio estado del sistema. Cada modelo requiere de distintos tipos de input, para lograr este tipo de iniciativas. Identifican los modelos que utilizan los sistemas para predecir su propio comportamiento, o la intención del usuario.

**Variabilidad** = Adaptación al contexto.

**Portabilidad** = Que pueda realizar cambio de plataforma.

**Desarrollo Distribuido** = Diseño de soft que soporta el desarrollo de SD.

**Escalabilidad** = Agregar más recursos como un server o más memoria.

**Capacidad de Ejecución** = Como los ejecutables llegan a la plataforma de host y como son invocados.

**Movilidad** = Tamaño, tipo de visualización, tipo de dispositivo de entrada.

**Monitoreo** = Monitorear el sistema mientras está trabajando.

**Integridad Conceptual** = Consistencia en el diseño de la arquitectura.

**Comerciabilidad** = No siempre se adaptan a lo que necesitamos.

**Tácticas de Arquitectura y Patrones:**

Los patrones y tácticas de arquitectura son los caminos para realizar un buen diseño de estructuras que pueden ser reutilizadas.

**Patrón de arquitectura:**

Es un paquete de decisiones de diseño, que se puede encontrar repetidamente en la práctica.

Conoce propiedades que permiten reutilización.

Describe una clase de arquitectura.

Establecen relaciones entre contexto (situación común que se repite y da lugar a un problema), problema y solución.

**Patrones de Módulo – Layered Patterns:**

Separar partes de los sistemas en capas para que puedan desarrollarse y evolucionar de forma independiente.

**Patrones Componente - Conector” – “Broker Pattern”:**

Algunos sistemas son construidos bajo una colección de servicios distribuidos sobre múltiples servers, y el problema está en cómo deben interoperar teniendo en cuenta la disponibilidad. La solución es separar los servicios de usuarios (clientes) de los servicios de proveedores (clientes) insertando un intermediario (Broker).

**Patrones Componente - Conector” – “MVC”:**

Modelo-Vista-Controlador.

**Patrones Componente - Conector” – “Client-Server Pattern”**

Un número de servicios son ofrecidos por proveedores de servicios y consumidos por servicios de consumidores, la solución es que el patrón describe una colección de componentes distribuidos que proveen y/o consumen servicios que son en gran parte independientes.

**“Patrones de alocación” – “Map-Reduce Pattern”**

Los negocios necesitan poder analizar grandes volúmenes de información que ellos generan o acceden. La solución radica en tener una infraestructura especializada, y luego mapeo y reducción.

**Arquitectura de SW en el Ciclo de Vida:**

**Arquitectura en proyectos ágiles:**

1. Responsabilidad sobre a las personas que están involucradas.

2. Realizar un desarrollo más rápido de su funcionalidad.

3. Mostrar un progreso más rápido en el ciclo de vida.

4. Realizar una documentación justa y necesaria.

**Puntos interesantes:**

•Poner una prioridad alta en la satisfacción del cliente cuando se entrega una versión al cliente.

• Si cambian los requerimientos, aunque sea tarde en el desarrollo, bienvenidos sean. Los procesos ágiles, aprovechan los cambios, para obtener ventajas competitivas del cliente.

• Las entregas de software trabaja con una frecuencia que puede ir desde un par de semanas a un par de meses, con una preferencia a la escala de tiempo más corto.

• La gente de negocio, y los desarrollos, deben trabajar conjuntamente a lo largo del proyecto.

**Requerimientos**:

Los requerimientos de arquitectura significativos (ASR) son aquellos que tienen un efecto profundo sobre la arquitectura, es decir, la arquitectura bien podría ser diferente en ausencia de tal requisito.

Los ASR generalmente (aunque no siempre) dan forma a los requerimientos de atributos de calidad como ser (la performance, la seguridad, la adaptabilidad, la disponibilidad, la usabilidad, etc.) que la arquitectura provee a un sistema

• Reunir ASRs de los documentos de requerimientos

• Reunir ASRs entrevistando a los interesados

• Reunir ASRs entendiendo los objetivos de Negocio

**Gestión y Gobierno**

**Planificación:**

En un proyecto sucede todo el tiempo.

**Organización**:

Organización de equipos, división de responsabilidades entre el Project manager, y el arquitecto de software, y la planificación global o distribuida del desarrollo.

**Roles típicos**:

Team Leader – Gestionan las tareas en el equipo.

Developer – Diseñan e implementan los subsistemas de código.

Configuration Manager – Ejecutan y construyen test de integración.

System test manager – Testeo de sistema, y testing de aceptación.

Product Manager – Representan el marketing. Definen características

**Coordinación**:

Contactos informales, Documentación, Reuniones, Comunicación electrónica.

**Implementación**:

El Project manager, y el arquitecto, tienen que tomar decisiones importantes. Se deben priorizar los riesgos, y para riegos los altos desarrollar una estrategia de mitigación.

**Arquitectura en la Nube:**

Definiciones básicas del Cloud:

1. Servicio a demanda: Consumidor utiliza servicio, según necesite de forma automática.

2. Acceso único de red: Los servicios y recursos cloud se utilizan heterogéneamente entre todos los clientes.

3. Pool de recursos: Los recursos se agrupan, para que el proveedor pueda brindar a los consumidores acorde a la demanda.

4. Independencia de ubicación: Locación independiente del acceso a red, a veces puede producir latencia.

5. Elasticidad rápida: Puesta en común de recursos, brinda capacidades rápidas de escalar rápidamente hacia afuera o hacia dentro del cloud (parece ilimitada).

6. Servicios medidos: Los sistemas cloud automáticamente controlan y optimizan los recursos. La utilización de los mismos puede ser monitoreada, controlada y reporteada, para que los consumidores de los servicios, solo sean facturados según su uso.

**Modelos de Servicios y Opciones de Desarrollo**

1. Software como un Servicio (SaaS)

El consumidor es el usuario final que utilizan aplicaciones que están en la nube como mail, calendario, streaming de video. No gestionan la infraestructura del cloud.

2. Plataforma como un Servicio (PaaS)

El consumidor es administrador/desarrollador del sistema. La plataforma provee una variedad de servicios (opciones de base de datos, balanceo de carga, disponibilidad, entorno de desarrollo). El consumidor desplegará aplicaciones en la infraestructura del Cloud. No gestionan la infraestructura del cloud, pero sí tienen sus propias aplicaciones y otras que configuren el entorno.

3. Infraestructura como un Servicio (IaaS)

El consumidor es administrador/desarrollador del sistema. La capacidad que tiene el consumidor es de proveer, procesamiento, almacenamiento, networking. No gestiona ni maneja la infraestructura del cloud pero tiene el control de los sistemas operativos, el almacenamiento, las aplicaciones desplegadas, y posiblemente pueda limitar el control de los componentes de redes (como ser un firewall).

**Modelos de Desarrollos**

Cloud privado. La infraestructura de cloud es comprado por una organización y operado solamente por aplicaciones compradas por esa organización. El principal objetivo de la organización no es vender servicios de cloud.

Cloud público. La infraestructura del cloud está disponible para el público, o para un gran grupo de una industria, y es comprada por la organización para vender servicios de cloud.

**Justificación Económica**

**Economía de escala**

Utilización de equipamiento

Multi-tenencia

**Arquitectura en un entorno de “Cloud”**

El Cloud es una plataforma, y construir el sistema para ejecutar en el Cloud, especialmente utilizando el IaaS no es muy diferente que construir una plataforma distribuida. Esto significa, que el arquitecto necesita prestar atención en la adaptabilidad, la usabilidad, la interoperabilidad, y el testeo, como haría en cualquier otra plataforma.

Los atributos de calidad que tienen algunas diferencias, son la seguridad, la performance y la disponibilidad.