ARQUITECTURA DE SOFTWARE

ANA MARIA MORENO CASADIEGO 1152073
YEINER DANIEL ANAYA 1152086
MOISES OSORIO 1152082

PRESENTADO
RODRÍGUEZ TENJO JUDITH DEL PILAR

UNIVERSIDAD FRANCISCO DE PAULA SANTANDER
1155705- ANALISIS Y DISEÑO DE SISTEMAS
CÚCUTA

ARQUITECTURA DE SOFTWARE

La arquitectura se entiende comúnmente como la disciplina que planifica y diseña edificios y espacios de esparcimiento, pero también se aplica a un nivel abstracto en la toma de decisiones antes de la realización de cualquier estructura. En el ámbito del software, la arquitectura surgió en los años 60 y se refiere a la planificación basada en modelos, patrones y abstracciones teóricas para desarrollar software complejo, proporcionando una guía detallada para entender cómo se integrarán las distintas partes del producto o servicio. En este contexto, un patrón es una solución general y reutilizable para problemas recurrentes en ingeniería del software, orientado a la estructura y a un nivel más genérico.

Importancia de la arquitectura:

- Permite planificar el desarrollo por adelantado y elegir las mejores herramientas.
- Determina en gran medida el ritmo del desarrollo y los factores económicos/humanos.
- Se deben considerar aspectos como costos, tiempo de desarrollo, número de usuarios y nivel de aislamiento.

Se describen varios patrones arquitectónicos comunes:

Cliente-Servidor: Un servidor proporciona servicios a múltiples clientes. Ventajas: centralización, escalabilidad, mantenimiento. Desventajas: disponibilidad, requisitos, distribución.

Capas: El software se divide en capas separadas que proporcionan servicios a las capas superiores. Ventajas: facilidad de pruebas y desarrollo. Desventajas: rendimiento, escalabilidad.

Maestro-Esclavo: Un nodo maestro orquesta tareas que son ejecutadas por nodos esclavos. Ventajas: gestión centralizada, control, escalabilidad. Desventajas: implementación, dependencia.

MVC: Separa la lógica en Modelo (datos), Vista (presentación) y Controlador (entrada). Ventajas: colaboración, múltiples vistas. Desventajas: complejidad, a veces lento.

Broker: Componentes desacoplados interactúan a través de un broker central. Ventajas: escalabilidad, rendimiento. Desventajas: coste, mantenimiento.

La elección del patrón arquitectónico es crítica y determinará en gran medida el éxito del proyecto, por lo que requiere un cuidadoso análisis previo.

puntos para convertirte en un mejor arquitecto de software

1. Diseñar: Conocer patrones de diseño, medidas de calidad, probar diferentes tecnologías y analizar patrones aplicados.
2. Decidir: Saber qué es importante, priorizar decisiones, conocer tu ámbito de competencia y evaluar múltiples opciones.
3. Simplificar: Cuestionar soluciones, dar un paso atrás, dividir problemas, aceptar la refactorización.
4. Programar: Tener un proyecto paralelo para experimentar, elegir las tecnologías correctas para probar.
5. Documentar: Aplicar principios de código limpio, generar documentación automáticamente, ser conciso.
6. Comunicar: Comunicar ideas de forma estructurada, dar presentaciones, encontrar el nivel adecuado, ser transparente.
7. Estimar y evaluar: Conocer gestión de proyectos, evaluar arquitecturas desconocidas.
8. Equilibrar: Balancear requisitos, resolver objetivos contradictorios, manejar conflictos.
9. Ser proactivo: Tener una visión, construir comunidades de práctica, sesiones de puertas abiertas.

10. Marketing: Motivar y convencer con prototipos, videos; ser persistente con tus ideas.

PATRONES DE ARQUITECTURA DE SOFTWARE: TIPOS Y CARACTERÍSTICAS

Cuando escuchamos el término arquitectura, pensamos inmediatamente en construir un edificio o diseñar un plano gigantesco para una gran obra. La experiencia, como siempre ganada con el tiempo, permitió crear comportamientos comunes que debía tener un software para poder desarrollarse más organizado y favorecer la interacción entre las partes, sin que perdieran su independencia.

1. Programación en capas (o N-capas): Divide la estructura del software en capas separadas como presentación, lógica y datos. Cada capa sólo se comunica con la anterior y siguiente. Previene el acoplamiento.

2. Tres niveles: Similar a N-capas, pero cada capa o grupo de capas se almacena en diferentes ubicaciones físicas, proporcionando más autonomía.

3. Arquitectura de microservicios y orientada a servicios: El software se desarrolla como un conjunto de servicios individuales que se comunican entre sí. Los microservicios se enfocan en la aplicación, mientras que SOA es un concepto más amplio.

4. Modelo-Vista-Controlador (MVC): Separa la lógica de negocio de la presentación en tres componentes: Modelo (datos), Vista (interfaz) y Controlador (lógica). Muy utilizado en frameworks modernos.

5. Arquitectura de microkernel: Divide el sistema en un núcleo mínimo y módulos/servicios independientes para manejar fallas y reducir complejidad.

6. Arquitectura en pizarra: Resuelve problemas complejos mediante agentes independientes que leen/escriben en una "pizarra" compartida.

7. Arquitectura dirigida por eventos: Las acciones se desencadenan por cambios de estado u otros eventos, con agentes emisores y consumidores.

Referencias bibliográficas

"Software Architecture in Practice" por Len Bass, Paul Clements y Rick Kazman (3ra edición, 2012) "Software Architecture in Practice" - https://www.pearson.com/us/higher-education/program/Bass-Software-Architecture-in-Practice-3rd-Edition/PGM333923.html

"Design Patterns: Elements of Reusable Object-Oriented Software" por el "Gang of Four" (Gamma, Helm, Johnson, Vlissides) (1994) https://en.wikipedia.org/wiki/Design_Patterns