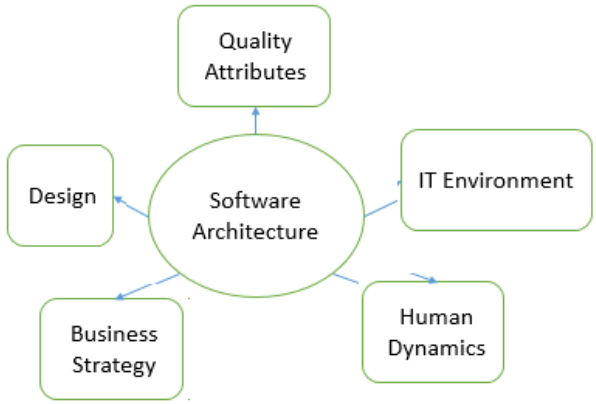
Arquitectura de Software
Refiere a las estructuras más grandes de un sistema de software, maneja la cooperación de múltiples procesos de software que cumplen una tarea.

Diseño de software
Se refiere a pequeñas estructuras (reducidas mediante un proceso de abstracción) que maneja un diseño con un alcance limitado referente a un solo proceso de software.

Entendiendo los alcances de los conceptos de arquitectura de software y diseño, estaremos en posición de elegir el modelo adecuado a cierto requerimiento de software.

	Arquitectura sirve como blueprint (plan o modelo), provee un modelo o abstracción de suficiente complejidad para establecer una coordinación entre diferentes componentes de un sistema de software. En arquitectura se toman decisiones no funcionales separadas en los requerimientos. En diseño todas las ideas se concretan en la implementación de requerimientos funcionales.

Diseño de software
Inicialmente (cuando diseño general) provee un plan que describe los elementos y como pueden trabajar juntos, también sirve para:
- Negociar los requerimientos y definir las expectaciones de los clientes
- Actúa como blueprint, guía la implementación de tareas, incluyendo diseño detallado.

Objetivos de arquitectura
- Identificar requerimientos que afecten la estructura de la aplicación,
- Reducción de riesgos asociados con la construcción de una solución técnica
- Es el puente las necesidades de negocio y requerimientos técnicos.

El objetivo principal de la arquitectura es
identificar los requerimientos que afecten la estructura de la aplicación y así reducir los riesgos y ser un puente de comunicación entre objetivos de negocio y requerimientos.

Objetivos secundarios
- Exponer la estructura del sistema sin los detalles de implementación
- Hacer los casos de uso
- Articular los requerimientos de diferentes protagonistas
- Manejo de requerimientos funcionales, no funcionales, y calidad
- Mejora la calidad y funcionalidad del sistema
- Aumenta la confianza en el sistema

Limitaciones

Es una disciplina emergente, establecida parcialmente por el tamaño de las empresas.
Falta de herramientas y técnicas estandarizadas para representar arquitecturas
Falta de herramientas para comprobar el impacto de la arquitectura en la implementación.
Falta de conciencia en la importancia del diseño arquitectural y desarrollo de software.
Falta de conciencia del rol disminuido del arquitecto de software y la comunicación con el resto de los protagonistas
Falta de conciencia del proceso de diseño, experiencia y evaluación de diseño.

Perfil del Arquitecto de Software

El arquitecto provee una solución (app) que el equipo técnico puede crear y diseñar, debe ser experto en diseño de software, incluye diferentes métodos y enfoques como POO y diseño dirigido por eventos.

Lidera al equipo de desarrollo y coordina los recursos para la integridad del diseño.
Debe ser capaz de revisar las propuestas de diseño y la negociación de estos.

Experto en el dominio (alcance)

Experto en el Sistema que se desarrolla y el plan para la evolución del sistema
Asistir en el proceso de investigación de los requerimientos, asegurando la consistencia y la totalidad de los recursos.
Coordinar la definición del dominio (alcance) del sistema a desarrollar

Experto en tecnología

Experto en tecnologías disponibles que ayude en la implementación del Sistema
Coordinar la selección del lenguaje de programación, framework, platforms, databases, etc

Experto en Metodologías

Experto en metodologías de desarrollo de software que puedan adoptarse
Seleccionar los enfoques apropiados para el desarrollo de todo el equipo de desarrollo

Perfil no técnico del arquitecto de software

Facilitar el trabajo técnico entre miembros y reforzar la confianza entre integrantes
Especialista en información que comparte su conocimiento y experiencia
Proteger al equipo de fuerzas externas que los puedan distraer y perder valor como equipo (cohesión)

Productos del Arquitecto

Un conjunto de requerimientos claros, completos y consistentes
Una descripción funcional del sistema, con al menos 2 capas de descomposición (perspectivas)
El concepto del sistema

Un diseño del sistema con al menos 2 capas de descomposición (perspectivas)
Noción del cronograma, atributos, y la implementación y planes de operación.
Documento o proceso que asegure la descomposición funcional, incluye la descripción de las interfaces.

>>>>>>>

Las Presentaciones

1 Arquitectura Jerárquica Leslie Miroslava

Introducción

Las arquitecturas, en general, se refieren a estructuras y patrones de diseño que definen cómo se organizan, comunican y operan los componentes de un sistema o una aplicación (hardware y software).

Buscan resolver problemas específicos de diseño y operación al proporcionar guías y patrones que optimizan el rendimiento, la escalabilidad, la reutilización y la mantenibilidad de los sistemas y las aplicaciones.

Hay varios tipos de arquitectura, en esta presentación analizaremos la Arquitectura jerárquica.

Conclusiones
Se basa en la idea de dividir un sistema complejo en componentes más pequeños y manejables, organizados en niveles o capas, cada uno con funciones y responsabilidades específicas.
Cada nivel o capa tiene un propósito específico y brinda servicios a los niveles adyacentes.

Ventajas A.K.A. dónde usarla?

- Estructura organizada: La arquitectura jerárquica brinda claridad en roles y responsabilidades, facilitando la toma de decisiones y la comunicación en sistemas y organizaciones.
- Escalabilidad eficiente: Permite agregar elementos nuevos sin alterar la estructura general, útil en sistemas de información y redes, además de simplificar la gestión en empresas y equipos.
- Simplificación: Divide sistemas complejos en partes manejables, promoviendo diseños más claros y mantenibles en sistemas, y facilitando la comunicación en organizaciones.

Desventajas A.K.A. dónde usarla?

- Falta de agilidad: La jerarquía puede ser rígida ante cambios rápidos, afectando la adaptación en entornos cambiantes.
- Comunicación ineficiente: La información pasa por varios niveles, lo que puede causar demoras y distorsiones en la comunicación.
- Centralización y restricción: Puede concentrar el poder en niveles superiores, limitando la autonomía y la innovación en niveles inferiores.

Aplicaciones en redes informáticas

- Segmentación eficiente: Divide redes en capas para dirigir el tráfico según funciones y necesidades.
- Escalabilidad: Permite expandir la infraestructura de red sin afectar otras capas.
- Reducción de congestión: Evita cuellos de botella al dirigir el tráfico adecuadamente.
- Seguridad y control: Facilita la aplicación de políticas de seguridad en capas específicas.
- Administración simplificada: Simplifica la gestión y el mantenimiento al separar las funciones y los cambios por capas.

Algunos diseños clásicos

1 **Main-Submain:** El objetivo de este estilo es reutilizar los módulos y desarrollar libremente módulos o subrutinas individuales.
En este estilo, un sistema de software se divide en subrutinas mediante el uso de un refinamiento de arriba hacia abajo según la funcionalidad deseada del sistema.

2 Master-Slave: Este enfoque aplica el principio de "divide y vencerás" y respalda el cálculo de fallas y la precisión computacional. Es una modificación de la arquitectura de subrutina principal que proporciona confiabilidad del sistema y tolerancia a fallas.

3 **Virtual Machine:** La arquitectura de la máquina virtual pretende alguna funcionalidad que no es nativa del hardware y/o software en el que se implementa. Una máquina virtual se construye sobre un sistema existente y proporciona una abstracción virtual, un conjunto de atributos y operaciones.

4 **Layer style:** En este enfoque, el sistema se descompone en una serie de capas superiores e inferiores en una jerarquía, y cada capa tiene su propia responsabilidad exclusiva en el sistema.

Desafíos actuales en la implementación de la arquitectura

- Flexibilidad en Entornos Cambiantes: Adaptar la jerarquía a cambios rápidos en negocios y
- tecnología.
- Colaboración Transversal: Fomentar la colaboración entre equipos y departamentos.
- Empoderamiento de los Niveles Inferiores: Equilibrar el poder entre niveles jerárquicos.

Impacto de la arquitectura jerárquica en la toma de decisiones organizacionales

- Centralización de Decisiones: Autoridad concentrada en niveles superiores.
- Eficiencia en Decisiones Rutinarias: Agilidad en decisiones operativas.
- Flujo de Información: Posible filtrado de información.
- Consistencia y Coherencia: Decisiones alineadas con objetivos.

Conclusión

La arquitectura jerárquica ofrece claridad en roles y eficiencia en la toma de decisiones, pero enfrenta desafíos como la rigidez y la limitación en la colaboración.

>>>>>>>>>>>>>>>>>

2 ARQUITECTURA ORIENTADA A LA INTERACCIÓN José Sandoval

Introducción

La arquitectura orientada a la interacción es una forma de separar la interacción del usuario de la abstracción de datos y el procesamiento de datos empresariales.
Esto se logra mediante la división de la aplicación en tres partes interconectadas: Modulo de datos, Modulo de control fijo, Modulo de presentación y visita.

La interacción orientada a arquitecturas tiene dos estilos:
Presentation-Abstraction-Control (PAC); Model-View-Controller (MVC).

Model-View-Controller (MVC).

El principal objetivo de esta arquitectura MVC es separar la interacción del usuario de la abstracción de datos y el procesamiento de datos comerciales.
Se utiliza cada vez más en aplicaciones de software que requieren la interacción del usuario tanto para entrada como para salida de datos

PARTICIONES PRINCIPALES

Módulo de datos	Provee la abstracción de datos y el núcleo lógico comercial.
Módulo de control de flujo	Determina los controles de flujo, la comunicación entre módulos, distribución de trabajo y algunas configuraciones de datos.
Módulo de presentación y vista	Se encarga de la presentación visual y auditiva de la salida de datos, además de proveer una interfaz para la entrada del usuario.

2 Modelos de MVC

MVC - I The Controller-View - La vista del controlador actúa como interfaz de entrada / salida y se realiza el procesamiento.
The Model - El modelo proporciona todos los datos y servicios de dominio.

MVC-II es una mejora de la arquitectura MVC-I en la que el módulo de vista y el módulo del controlador están separados.

El módulo de modelo juega un papel activo como en MVC-I al proporcionar toda la funcionalidad principal y los datos admitidos por la base de datos.
El módulo de vista presenta datos mientras que el módulo controlador acepta la solicitud de entrada, valida los datos de entrada, inicia el modelo, la vista, su conexión y también distribuye la tarea.

Las aplicaciones MVC son efectivas para aplicaciones interactivas donde se necesitan múltiples vistas para un solo modelo de datos y es fácil conectar una vista de interfaz nueva o cambiar.

Las aplicaciones MVC son adecuadas para aplicaciones en las que existen claras divisiones entre los módulos, de modo que se pueden asignar diferentes profesionales para trabajar en diferentes aspectos de dichas aplicaciones al mismo tiempo

ventajas MVC	desventajas MVC
Código fácil de mantener y ampliar Componentes se pueden probar por separado Los componentes se pueden desarrollar simultáneamente Reduce la complejidad de la aplicación Funciona bien para aplicaciones web con grandes equipos de trabajos Compatible con optimización de motores de búsqueda	Es difícil de leer, cambiar, probar y reutilizar este modelo. No es adecuado para crear aplicaciones pequeñas Ineficiencia del acceso a los datos a la vista La división entre Vista y Controlador no está clara en algunos casos.

3 PRESENTACIÓN-ABSTRACCION-CONTROL (PAC)

1 Controlador: Presentación-Abstracción-Control:

No tiene una estructura jerárquica clara.
Es una arquitectura jerárquica basada en agentes.
Su sistema está organizado en una jerarquía de muchos agentes cooperantes (triadas).
Desarrollada a partir de MVC.

APLICACIONES

Útil para cualquier sistema distribuido en donde cada agente remoto tiene sus propias funcionalidades con datos e interfaz interactiva.
Sistema interactivo que se pueda descomponer en muchos agentes cooperantes de manera jerárquica.

Ventajas PAC	Desventajas PAC
Soporte para múltiples tareas y vistas Soporte para la reutilización y extensión Fácil de integrar o cambiar a nuevos agentes	Gasto debido al control del puente entre interfaz y abstracción y la comunicación de los agentes.

Soporte de concurrencia de agentes en Paralelo	Es difícil determinar el número correcto de agentes debido al débil acoplamiento y la alta independencia entre los agentes. Cada agente generado aumenta la complejidad del desarrollo debido a la comunicación entre estos mismos.

Conclusión

La arquitectura orientada a la interacción es una forma efectiva de separar la interacción del usuario de la abstracción de datos y el procesamiento de datos empresariales.

Tanto Model-View-Controller (MVC) como Presentation-Abstraction-Control (PAC) son ejemplos de esta arquitectura, y ofrecen beneficios como mayor modularidad, reutilización de componentes, facilidad para realizar pruebas y depuración, escalabilidad y flexibilidad.

La arquitectura orientada a la interacción se utiliza en:

- aplicaciones web interactivas
- sistemas de control de procesos industriales,

>>>>>>

3 Data centered Architecture Héctor Alejandro Robles Pérez

introducción

Arquitectura Centrada en Datos se presenta como un enfoque esencial para el diseño de sistemas y aplicaciones.
Se enfoca a la forma en que las organizaciones gestionan sus datos, situándolos en el epicentro de sus operaciones y estrategias.

Que es ?
Arquitectura en la cual los datos están centralizados y son accedidos con frecuencia por otros componentes, que modifican los datos.
El objetivo principal de este estilo es lograr la integridad de los datos.

Características

Decisión de diseño orientada a centralización de datos
Uso de un almacén centralizado
Elementos funcionales de acceso (agentes)

Tipos de Componentes

Acceso de datos	Colección de componentes independientes que operan en el almacén central de datos, realizan cálculos y pueden retrasar los resultados.

Datos centrales	Estructura o almacén de datos o repositorio de datos, que es responsable de proporcionar almacenamiento permanente de datos. Representa el estado actual.

Estilos de esta arquitectura

Estilo de repositorio	• El almacén de datos pasivo • Clientes (componentes de software o agentes) activos • Flujo lógico está determinado por agentes • Los componentes participantes verifican el almacén de datos en busca de cambios.
Estilo de pizarra	• El almacén de datos activo • Clientes/agentes pasivos. • Flujo lógico determinado por el estado actual de los datos en el almacén de datos. • Tiene componente de pizarra, que actúa como un depósito central de datos, • Diferentes elementos computacionales construyen y actúan sobre una representación interna.

LO BUENO Y LO MALO DEL ESTILO DE REPOSITORIO

VENTAJAS REPOSITORIO	**DESVENTAJAS REPOSITORIO**
Proporciona funciones de integridad de datos, copia de seguridad y restauración. Proporciona escalabilidad y reutilización de agentes ya que no tienen comunicación directa entre sí. Reduce la sobrecarga de datos transitorios entre componentes de software.	Es más vulnerable a fallas y es posible la replicación o duplicación de datos. Alta dependencia entre la estructura de datos del almacén de datos y sus agentes. Los cambios en la estructura de datos afectan en gran medida a los clientes. La evolución de los datos es difícil y costosa. Costo de mover datos en la red para datos distribuidos.

Características del estilo de pizarra

• La pizarra almacena una serie de componentes que actúan de forma independiente sobre la estructura de datos común.
• Los componentes interactúan únicamente a través de la pizarra. El almacén de datos alerta a los clientes cada vez que hay un cambio en el almacén de datos.
• El estado actual de la solución se almacena en la pizarra y el estado de la pizarra activa el procesamiento.

Ventajas y desventajas del estilo de pizarra

VENTAJAS pizarra	DESVENTAJAS pizarra

Es útil cuando el problema a resolver es complejo. Proporciona escalabilidad que proporciona una fuente de conocimiento fácil de agregar o actualizar. Proporciona concurrencia que permite que todas las fuentes de conocimiento trabajen en paralelo, ya que son independientes entre sí. Comunicación más efectiva entre los sistemas. Admite la reutilización de los agentes de origen de conocimiento	Problemas de carga al revisar la pizarra. Grandes costos en el traslado de información. Es difícil obtener una traza de los pasos que llevaron a la solución, es decir, no ofrece explicaciones. Vulnerable a la falta de datos Peligro de que los agentes dupliquen datos

>>>>>>>>>

4 Arquitectura de software DataFlow Rogelio Yahir Seañez Ochoa

INTRODUCCIÓN
El término "data flow" se refiere al flujo de datos, es decir, al movimiento y la transformación de información o datos a través de un sistema o proceso.

¿Que es la arquitectura de software?
La arquitectura de software se refiere a la estructura fundamental y el diseño organizacional para manejar el data flow.

decisiones arquitecturales comprenden:
● Selección de elementos estructurales y sus interfaces mediante los cuales se compone el sistema.
● Composición de estos elementos estructurales y de comportamiento en subsistemas más grandes.
● Elección de patrones de diseño, lenguajes de programación, tecnologías, enfoques de seguridad y más.
● Permita futuras modificaciones y ampliaciones sin rehacer todo el sistema

¿Que es la arquitectura DataFlow?

En la arquitectura DataFlow todo el sistema de software se ve como una serie de transformaciones en fragmentos consecutivos o conjuntos de datos de entrada, donde los datos y las operaciones son independientes entre sí.
En este enfoque, los datos ingresan al sistema y luego fluyen a través de los módulos uno a la vez hasta que se asignan a algún destino final.

Tipos de ejecuciones entre módulos

● Secuencial por Lotes
● Flujo no-secuencial de datos
● Control de Procesos

Secuencial por Lotes
En este modelo de procesamiento de datos, los lotes de información pasan por módulos o

funciones independientes, modificando, alterando o, si acaso, solamente leyendo la información hasta llegar al final de la ejecución.

Flujo no-secuencial de datos
La arquitectura de flujo no secuencial de datos o “Pipe and Filter”, hace énfasis en la transformación incremental de los datos por componentes sucesivos.
Este flujo está conformado por filtros, que son módulos de procesamiento de información

Control de Procesos
El control de procesos es un tipo de arquitectura de flujo de datos en la cual los datos no se agrupan en lotes secuenciales ni se transmiten de manera secuencial en forma de tuberías.
El flujo de datos proviene de un conjunto de variables que controlan la ejecución del proceso.
Descompone todo el sistema en subsistemas o módulos y los conecta entre sí

Partes de la unidad de control
- Variables controlada: Es la variable que el sistema toma en cuenta en todo momento
- Variable de entrada: Variables medidas para el proceso
- Variable manipulada: La variable que el sistema va a manipular según las entradas y otros procesos
- Definición de proceso: Procesa las variables actuales en el sistema para obtener una respuesta

>>>>>>>>>>

5 Arquitectura Basada en Componentes

Introducción
En la era actual de la tecnología, donde la escalabilidad, la reutilización y la mantenibilidad son esenciales, es crucial.

Definición de Arquitectura Basada en Componentes:
• La Arquitectura Basada en Componentes es un enfoque de diseño de software en el que las aplicaciones se dividen en módulos autónomos y reutilizables llamados "componentes".

Beneficios de la Arquitectura Basada en Componentes:
- Reusabilidad: Los componentes pueden ser utilizados en múltiples proyectos, reduciendo el esfuerzo de desarrollo y mejorando la coherencia.
- Escalabilidad: Los sistemas se pueden escalar de manera más eficiente al agregar o reemplazar componentes según sea necesario.
- Mantenibilidad: Los cambios en un componente afectan solo a ese componente, lo que facilita las actualizaciones y correcciones sin afectar al sistema en su conjunto

Ejemplos de Arquitecturas Basadas en Componentes:
- Modelo Vista Controlador (MVC): División en tres componentes: Modelo (lógica de negocio), Vista (interfaz de usuario) y Controlador (gestión de eventos y comunicación).
- Microservicios: Aplicación dividida en componentes pequeños e independientes, cada uno con su propia funcionalidad y comunicación a través de APIs.
- Bibliotecas y Frameworks: Uso de bibliotecas y frameworks para incorporar componentes predefinidos y acelerar el desarrollo.

Pasos de Implementación:

- Identificar Funcionalidades: Dividir la aplicación en funcionalidades discretas y definir los componentes que las implementarán.
- Definir Interfaces: Establecer interfaces claras y definidas para la comunicación entre componentes.
- Desarrollo y Prueba Individual: Desarrollar y probar cada componente por separado para garantizar su funcionamiento correcto.
- Integración: Conectar los componentes según las interfaces definidas y realizar pruebas de integración.
- Mantenimiento Continuo: Actualizar, mejorar o reemplazar componentes según sea necesario durante el ciclo de vida de la aplicación.

Conclusión:
• En resumen, la Arquitectura Basada en Componentes es una estrategia poderosa para construir sistemas de software escalables, mantenibles y eficientes.

>>>>>>>>>>>

6 DISTRIBUTED ARCHITECTURE

INTRODUCCIÓN
Para incrementar el rendimiento de los ordenadores se utiliza los sistemas distribuidos donde un conjunto de ordenadores independientes funciona como uno solo a ojos del usuario.

definición

La arquitectura de sistemas distribuidos se refiere a un método que consiste en hacer que varias computadoras trabajen juntas para resolver un problema común.

De este modo, una red de computadoras forma una única computadora potente que brinda recursos a gran escala para afrontar desafíos complejos.

VENTAJAS	DESVENTAJAS
mayor tolerancia a errores mayor velocidad Flexibilidad y escalabilidad Costo reducido Seguridad	Mayor nivel de complejidad Mayor esfuerzo