SOLID: La intención es aplicar estos principios en conjunto para que sea más probable obtener un software fácil de mantener y extender en el tiempo.
Los principios SOLID son guías, no son reglas inamovibles.

6. **Explicar qué es la arquitectura de un sistema:**

   LA ARQ ESTÁ GUIADA POR LA DEFINICIÓN DE LOS REQ NO FUNCIONALES
   VAMOS A DISEÑAR LA ARQ EN FUNCIÓN DE LOS REQ NO FUNC

   - Gran parte del éxito en la calidad del sistema depende de la arquitectura
   - Actúa como puente entre los requerimientos y la implementación
   - La arquitectura representa las decisiones de diseño significativas que le dan forma al sistema, aquellas que tienen gran impacto y un gran costo de cambio
   - La arquitectura establece restricciones sobre actividades subsiguientes
   - Las decisiones de arquitectura son las que permiten cumplir con el comportamiento requerido y con los atributos de calidad. Pueden ser generales o más específicas como por ej adoptar cierto protocolo de comunicación. Si es importante para cumplir con los objetivos del sistema, es una decisión de arquitectura
   - La arquitectura de software debe ser el núcleo del diseño y desarrollo de un sistema de software. Debe estar siempre en el primer plano durante todo el ciclo de vida del sistema
   - La arquitectura es un conjunto de las principales decisiones de diseño que determinan los elementos clave del sistema y sus interrelaciones. Estas decisiones son un nivel de abstracción por encima del código.
   - El estilo arquitectónico restringe al código en algunos aspectos (por ejemplo, en cómo debe interactuar con otros componentes), pero da libertad para la codificación interna del componente

   DEFINICIONES:

   - Una arquitectura es el conjunto de decisiones importantes sobre la organización de un sistema de software, la selección de los elementos estructurales y sus interfaces por los cuales el sistema está compuesto, junto con su comportamiento como es especificado en las colaboraciones entre estos elementos, la composición de estos elementos estructurales y de comportamiento en subsistemas progresivamente más grandes, y el estilo de arquitectura que guía a esta organización (estos elementos y sus interfaces, sus colaboraciones y su composición).
   - Cómo un sistema se puede descomponer de la mejor forma posible? o Cuáles son sus componentes? o Cómo se comunican dichos componentes? o Cómo fluye la información? o Cómo los elementos de un sistema pueden evolucionar independientemente? o Cómo se puede describir todo esto a través de notaciones formales e informales?

7. **Enumerar tres razones por las cuales es importante la arquitectura de un sistema**

a. Inhibe/habilita los atributos de calidad del sistema - si un sistema cumplirá con sus atributos de calidad es determinado (pero no garantizado) por la arquitectura
b. Las decisiones tomadas en una arquitectura permiten administrar los cambios a medida que el sistema evoluciona
    i. Aproximadamente el 80% del costo total de un sistema de software ocurre luego de la primera instalación.
c. El análisis de una arquitectura permite predicciones tempranas acerca de las cualidades del sistema
    i. Evaluando la arquitectura podríamos predecir si el sistema cumplirá con los atributos de calidad deseados.
    ii. Herramientas: técnicas de evaluación cuantitativas, prototipos, pruebas de concepto, etc.
d. Una arquitectura documentada mejora la comunicación entre los interesados
    i. La arquitectura provee una abstracción del sistema que los interesados pueden usar para entenderlo.
    ii. Cada interesado tiene interés en diferentes características del sistema
    iii. La arquitectura provee un lenguaje común en la cual los distinto intereses pueden ser expresados, negociados y resueltos a un nivel manejable para sistemas grandes y complejos.
e. Es el portador de las decisiones de diseño más importantes y difíciles de cambiar
    i. La arquitectura refleja las decisiones de diseño más tempranas que tienen gran influencia en el desarrollo, instalación y mantenimiento.
    ii. Cambiar alguna de estas decisiones causaría un efecto dominó sobre otras
f. Define un conjunto de restricciones sobre la implementación subsecuente
    i. La implementación de un sistema debe conformar las decisiones de diseño prescritas por la arquitectura

8. **Explicar tres tipos de actividades que debe realizar un Arquitecto de Software**

Roles:

a. Está al tanto de lo que está ocurriendo en el proyecto: detectando problemas importantes y resolviéndolos antes que se transformen en problemas serios.
b. Intensa colaboración en el equipo.
c. Es el mentor del equipo de desarrollo.
d. Es el guía de la aventura de desarrollar un software.

Actividades:

Definición de arquitectura:

e. Gestión de Requerimientos No Funcionales: Específicos Medibles Realizables Comprobables
f. Definición de Arquitectura:Trata de introducir estructura, lineamientos, principios y liderazgo a los aspectos técnicos de un proyecto de software.

g. Selección de Tecnologías teniendo en cuenta: Costo, Licenciamiento, Interoperabilidad, Soporte, Entrega, Políticas de actualización, Dependencias con otras tecnologías, Conocimiento del equipo, Curva de aprendizaje
h. Evaluación de Arquitectura: o La arquitectura funciona? o Satisface los requerimientos no funcionales o Provee los fundamentos necesarios para el resto del código o Funciona como una plataforma para resolver los problemas de negocio subyacentes. o ¿Cómo se verifica si funciona? Desarrollar test y pruebas para los componentes y atributos de calidad
i. Colaboración de Arquitectura: debe asegurar que la arquitectura ha sido entendida por todos los interesados en el sistema de software

Entrega de la arquitectura:

j. Propietario de la Visión Global
k. Liderazgo: asegurarse que todo es tenido en cuenta y que el equipo está siendo dirigido en la dirección correcta, de manera continua
l. Coaching y Mentoring: proveer asistencia a los desarrolladores para resolver problemas particulares, aportar su experiencia y promover que se comparta el conocimiento entre los miembros del equipo
m. Aseguramiento de la Calidad: definir estándares de codificación, delinear principios de diseño a cumplir, testeo unitario automático, herramientas de análisis de cobertura de tests
n. Diseño, Desarrollo y Testing: Muchos arquitectos son programadores experimentados, así que tiene sentido mantener esas habilidades al día.

LA CALIDAD ES el total de características y aspectos de un producto o servicio en los que se basa su aptitud para satisfacer una necesidad dada.
La calidad es un concepto relativo, está en los ojos del observador y es relativa a las personas, su edad y circunstancias, al espacio, tiempo
Multidimensional Referida a varias cualidades; por ejemplo, funcionalidad, oportunidad, costo,
Sujeta a restricciones Presupuesto disponible
Ligado a compromisos aceptables Plazos de fabricación

1 IDENTIFICAR LOS ATRIBUTOS DE CALIDAD
2 ESPECIFICAR ESOS REQ DEFINIENDO ESCENARIOS GRALES Y CONCRETOS
3 PLANTEAR ESTRATEGIAS EN EL DISEÑO DE LA ARQUITECTURA PARA ALCANZAR LA CALIDAD

9. Explicar para qué sirve un escenario de atributo de calidad

Mecanismo que permite capturar/definir/modelar aspectos de atributos de calidad de una forma que puede ser evaluado y utilizado en diseño.

Un escenario de atributo de calidad plantea la especificación de requisitos de calidad. Puede ser generales o concretos:

Generales: Son independientes del sistema y, potencialmente pueden pertenecer a cualquier sistema.

Ejemplo: Llega un requerimiento para hacer un cambio en la funcionalidad, y el cambio debe ser hecho en un momento particular dentro del proceso de desarrollo y dentro de un periodo de tiempo específico.

Concretos: Aquellos que son específicos al sistema en consideración.

Ejemplo: Llega un requerimiento para agregar soporte a un nuevo browser para un sistema web y el cambio debe ser hecho en, como máximo, 2 semanas.

Para cada atributo de calidad distinguen escenarios generales (aquellos que son independientes del sistema y potencialmente pueden pertenecer a cualquier sistema) de los escenarios concretos (específicos al sistema particular bajo consideración). Luego se presentan caracterizaciones como una colección de escenarios generales. Para traducir el atributo de calidad en requisitos para un sistema en particular, los escenarios generales deben traducirse en concretos.

Tiene 6 partes:
Estímulo - la condición a ser considerada cuando arriba al sistema(lo que sucede)
Fuente de Estímulo - es la entidad (humano, otro sistema, o procedimiento) que generó el estímulo
Ambiente - Entorno del sistema cuando recibe el estímulo (e.g. el sistema está en una condición de sobrecarga, está operativo, detenido, arrancando, etc.)
Artefacto - la parte del sistema (componente, equipo, sistema entero) que es estimulada
Respuesta - define las acciones que el artefacto debería realizar como respuesta al estímulo
Medida de la Respuesta -La respuesta, debe ser medible de alguna manera para probar que se cumple el requerimiento.

EJ: “Bajo una operación normal del sistema, las transacciones ejecutadas por los usuarios deberían resolverse, en promedio, en 2 segundos.

Fuente: Usuarios
Estímulo: Instanciar transacciones
Artefacto: Sistema
Ambiente: Bajo operación normal
Respuesta: Transacciones procesadas
Medida de Respuesta: Latencia promedio de 2 segundos

10. Explicar qué es una táctica de atributos de calidad y cómo se usa

Una táctica es una decisión de diseño que cuando la implementamos va a influir el control de la respuesta de un atributo de calidad. Va a influir en cómo respondemos a ese atributo de calidad.

Si quiere un sist modificable me interesaria que todos los módulos van a tener un tamaño razonable, una táctica sería que cada módulo no tenga más de 200 líneas de código.

Son estrategias que se utilizan para especificar atributos y así poder medirlos, esto a través de escenarios.

Cada táctica es una opción de diseño. Puede haber otras. Ej, que cada módulo tenga máx 200 líneas, o que haya un cierto nivel de cohesión
Una táctica puede ser refinada en otras.
Generalmente las tácticas para un atributo de calidad se organizan en una jerarquía

11. **Explicar tres atributos de calidad**
12. **Explique la diferencia entre estilos y patrones de arquitectura**

La diferencia es muy sutil, los límites son difusos. Son guías de alto nivel.

Un **estilo arquitectónico** es un colección de decisiones de diseño arquitectónico que:

1. son aplicables a un contexto de desarrollo(“sistemas altamente distribuidos”, o “sistemas intensivos en GUI”)
2. dado un sistema particular, restringe las decisiones de diseño de arquitectura sobre dicho sistema, y
3. garantiza ciertas calidades del sistema resultante

Un **patrón arquitectónico** es una colección de decisiones de diseño arquitectónico que son aplicables a problemas de diseño recurrentes, y que están parametrizados para tener en cuenta los diferentes contextos de desarrollo de software en el que surge el problema.

provee un conjunto de decisiones específicas de diseño que han sido identificadas como efectivas para organizar ciertas clases de sistemas de software o, más típicamente, subsistemas específicos.

estas decisiones de diseño pueden pensarse como “configurables”, ya que necesitan ser instanciadas con los componentes y conectores particulares a una aplicación.

La **diferencia** es, los 2 aportan decisiones de diseño, en el estilo son decisiones de diseño más abiertas, más generales que se aplican a una gran variedad de casos/sistemas, en la caso del patrón son soluciones que se van a aplicar repetidamente y todas juntas cuando se da un determinado tipo de problema.

ESTILOS:

Un vocabulario para los elementos de diseño o Tipos de componentes y conectores
o Por ejemplo: clases, invocaciones, “pipes”, clientes, etc.

Reglas de composición o Un estilo tiene restricciones topológicas que determinan cómo se puede hacer la composición de los elementos Por ejemplo: los elementos de un “layer” se pueden comunicarse sólo con los del “layer” inferior
Semántica para los elementos - elementos con significado bien conocido

Permiten: reuso de diseños
soluciones maduras aplicadas a problemas nuevos
una parte importante del código que implementa la arquitectura puede pasarse de un sistema a otro
comunicación más efectiva
portabilidad de soluciones

	ESTILO	PATRÓN
Alcance	Aplican a un contexto de desarrollo: o “sistemas altamente distribuidos”, o “sistemas intensivos en GUI”	Aplican a problemas de diseño específicos: o El estado del sistema debe presentarse de múltiples formas o La lógica de negocio debe estar separada del acceso a datos
Abstracción	Son muy abstractos para producir un diseño concreto del sistema.	Son fragmentos arquitectónicos parametrizados que pueden ser pensados como una pieza concreta de diseño.
Relación	Un sistema diseñado de acuerdo a las reglas de un único estilo puede involucrar el uso de múltiples patrones	Un único patrón puede ser aplicado a sistemas diseñados de acuerdo a los lineamientos de múltiples estilos

13. Explicar las características que presenta una arquitectura cliente/servidor

PERTENECE AL ESTILO POR CAPAS (2 - cliente y server)

Los clientes conocen al servidor pero no al revés.

Separar físicamente los componentes de software usados para requerir servicios de aquellos usados para proveer servicios, con el objetivo de proveer una distribución y escalamiento apropiados (tanto en la cantidad de proveedores de servicio como de solicitantes de servicios)

Hacer que **los proveedores no tengan conocimiento sobre la identidad de los solicitantes** para permitir que los sirvan transparentemente a muchos (posiblemente cambiantes) solicitantes

Aislar a los solicitantes unos de otros para permitir que sean agregados o eliminados independientemente. Hacer que los solicitantes sólo dependan de los proveedores de servicio

Permitir que **múltiples proveedores de servicios se creen de forma dinámica** para sacar carga a los proveedores existentes si la demanda de servicios aumenta por encima de un determinado umbral

DESCRIPCIÓN Clientes le envían requerimientos al servidor, el cual los ejecuta y envía la respuesta (de ser necesario). La comunicación es iniciada por el cliente.

CUALIDADES Sencilla y muy utilizada Centralización de cómputos y datos en el server. Mantenible Un único servidor puede atenderá a múltiples clientes

USOS TÍPICOS Apps con datos y/ o procesamiento centralizados y clientes GUI Stacks de protocolos de red Aplicaciones empresariales

PRECAUCIONES Condiciones de la red vs crecimiento de clientes.

ESTILOS DE FLUJOS DE DATOS:

En esencia considera el movimiento de datos entre unidades de procesamiento independientes.
La estructura del sistema está basada en transformaciones sucesivas de los datos.
Los datos entran al sistema y fluyen a través de los componentes hasta su destino final.
Normalmente un programa controla la ejecución de los componentes (lenguaje de control)

Típico en sistemas financieros.

14. **Explicar las características que presenta una arquitectura batch/secuencial**

    DESCRIPCIÓN Programas separados, independientes y ejecutados en orden. Los datos son pasados como un **lote** de un programa al siguiente. El siguiente programa **espera hasta que su predecesor finaliza** con el procesamiento del lote de datos completo.

    CONECTORES Distintos tipos de interfaces: desde humana hasta webservices

    RESTRICCIONES ADICIONALES Se ejecuta un programa a la vez, hasta que termina.

    CUALIDADES Sencillez Ejecuciones independientes

    USOS TÍPICOS PROCESAMIENTO DE TRANSACCIONES en sistemas financieros, clearing bancario, hay un cierto orden en las operaciones para calcular los balances

    PRECAUCIONES Cuando se requiere interacción entre componentes. Cuando se requiere concurrencia entre componentes. Acceso random a datos es deseable

15. **Explicar las características que presenta una arquitectura pipes & filters**

El siguiente programa puede procesar los elementos de datos tan pronto como empiezan a estar disponibles.
Los programas se ejecutan concurrentemente y hay mayor performance
Los datos son considerados como “streams ”

DESCRIPCIÓN Programas separados y ejecutados, potencialmente de manera concurrente. Datos son pasados como un stream de un programa al siguiente.

CONECTORES Routers explícitos de streams de datos

TOPOLOGÍA Pipeline (conexiones en T son posibles)

CUALIDADES Filtros mutuamente independientes. Estructura simple de streams de entrada/salida facilitan la combinación de componentes. Flexibilidad: Agregar, eliminar, cambiar y reusar filtros

USOS TÍPICOS Aplicaciones sobre sistemas operativos Procesamiento de audio, video Web servers (procesamiento de requerimientos HTTP)

PRECAUCIONES Cuando estructuras de datos complejos deben ser pasadas entre filtros complica la implementación. Cuando se requiere interacción entre filtros.

ESTILOS DE MEMORIA COMPARTIDA:

Múltiples componentes acceden al mismo almacenamiento de datos.
Los componentes se comunican entre ellos a través de dicha memoria
El diseño se centra especialmente en los repositorios compartidos
Se originaron con el uso de datos globales en aplicaciones en C y Pascal.

16. Explicar las características que presenta una arquitectura tipo blackboard

Programas independientes que acceden y se comunican a través de un repositorio de datos global (blackboard).
Cada uno modifica una parte de los datos y actualiza el contenido
Luego, otro programa puede utilizar estos nuevos datos
El estado de la información en el pizarrón determina el orden de ejecución de los distintos programas expertos.
Hay un agente activo que monitoriza los datos y notifica a todos que se produjo un cambio

DESCRIPCIÓN Programas independientes que acceden y se comunican a través de un repositorio de datos global (blackboard).

CONECTORES Acceso al blackboard Referencia directa a memoria, llamada a procedimiento, consultas a la base de datos, ...

TOPOLOGÍA Estrella, con el blackboard al medio

RESTRICCIONES ADICIONALES Detección de cambios en el blackboard:

1.Polling sobre el blackboard
2.Blackboard Manager se encarga de notificar cambios

CUALIDADES La solución completa a un problema no tiene que ser pre-planificada. La evolución del estado determina las estrategias a ser adoptadas.

USOS TÍPICOS Resolución de problemas heurísticos en inteligencia artificial Compiladores, IDEs, Chatroom con muchos chat clients

PRECAUCIONES Si la interacción entre programas “independientes” necesita de reglas de regulación compleja Cuando los datos en el blackboard están sujetos a cambios frecuentes y se requiere propagarlos entre todos componentes participantes. Existe otra estrategia más simple

EVITARLO SI… Los programas tratan con partes independientes de los datos comunes La interface de los datos comunes es susceptible a cambiar Las interacciones entre los programas independientes requieren complejas regulaciones

Especialización: Repositorio, repo compartido recibe y procesa todos los pedidos pero no hay ninguna notificación a los componentes. Ej: SIU

17. Explicar las características que presenta una arquitectura basada en reglas

Es un tipo de arquitectura de memoria compartida altamente especializada.

La memoria compartida se denomina base de conocimiento. Contiene “hechos” (sentencias de valores de variable) y “reglas de producción” que consisten en condicionales sobre los hechos

Almacena y manipula conocimiento para interpretar información de manera útil

El estilo es especialmente usado en inteligencia artificial :

Diagnóstico médico basado en síntomas
Movimiento de juegos deducido

3 COMPONENTES CLAROS:

interfaz de usuario Provee dos modos: Ingresar hechos y reglas Ingresar consultas (goals)

motor de inferencias Opera sobre la base de conocimiento en respuesta a una entrada de usuario: Hechos y reglas son agregados a la base de conocimiento Las consultas son comparadas contra los hechos existentes Match exacto - retorna true Si no hay match exacto - evalúa las reglas correspondientes para determinar la validez de la consulta

base de conocimiento Memoria compartida que contiene: Hechos: sentencias de valores de variables Reglas: Cláusulas “if... then” sobre el conjunto de variables

DESCRIPCIÓN El motor de inferencias parsea la entrada del usuario Si es un hecho/regla, la agrega a su base de conocimiento Si es una consulta (goal), obtiene las reglas aplicables desde la base de conocimiento e intenta resolverla.

COMPONENTES Interfaz de usuario, motor de inferencias, base de conocimiento

CONECTORES Los componentes están estrechamente conectados a través de: llamadas a procedimientos acceso a datos compartidos

ELEMENTOS DE DATOS Reglas/hechos y consultas

TOPOLOGÍA 3 capas altamente acopladas: interfaz de usuario motor de inferencia base de conocimiento

CUALIDADES Modificabilidad - el comportamiento de la aplicación puede ser modificado agregando o eliminando reglas dinámicamente Facilita el prototipado de sistemas pequeños

USOS TÍPICOS Cuando el problema puede ser entendido como una cuestión de resolver repetidamente un conjunto de predicados

PRECAUCIONES Cuando se tiene una gran cantidad de reglas, entender las interacciones entre múltiples reglas afectadas por los mismos hechos llega a ser difícil.

EVITARLO SI… El número de reglas es extenso Se presentan interacciones entre las reglas Se requiere alta performance

ESTILOS DE INTÉRPRETE

Interpretación dinámica y on-the-fly de comandos

Los comandos son definidos en términos de comandos primitivos predefinidos.

Proceso de interpretación

Comienza con un estado de ejecución inicial (datos iniciales)
Obtiene el primer comando a ejecutar
Ejecuta el comando sobre el estado de ejecución actual (probablemente modificando dicho estado)
Identifica el próximo comando a ejecutar (probablemente afectado por el resultado del comando anterior – por ejemplo condicionales)
Ejecuta el siguiente comando

El estilo arquitectónico intérprete básico implica la ejecución uno a uno de comandos

18. **Explicar las características que presenta una arquitectura basada en intérprete**

    Se puede decir que es similar al estilo basado en reglas: el motor de inferencia parsea el comando de entrada y lo resuelve en base a la base de conocimiento.

Toma un programa escrito en un lenguaje y lo interpreta a otro lenguaje para ejecutar una serie de comandos

Facilita la codificación en un lenguaje de más alto nivel

Ventajas Portabilidad y flexibilidad de aplicaciones o lenguajes, Soporte dinámico de cambios

Ejemplos LISP, Perl Fórmulas de Excel

DESCRIPCIÓN Parsea y ejecuta comandos de entrada, actualizando el estado mantenido por el intérprete

COMPONENTES intérprete de comandos estado del programa interpretado estado del intérprete la interfaz de usuario

CONECTORES Los componentes están estrechamente conectados a través de: llamadas a procedimientos acceso a datos compartidos

ELEMENTOS DE DATOS Comandos

TOPOLOGÍA 3 capas altamente acopladas. El estado puede estar separado del intérprete.

CUALIDADES Es posible obtener un comportamiento altamente dinámico, donde el conjunto de comandos se va modificando. La arquitectura del sistema puede ser constante mientras se crean nuevas capacidades a partir de primitivas existentes

USOS TÍPICOS end-user programing

PRECAUCIONES Cuando se necesita un procesamiento rápido (el código interpretado tarda mucho más que el código ejecutable).

EVITARLO SI… Se requiere alta performance

19. Explicar las características que presenta una arquitectura de código móvil

Algunas veces la interpretación no se puede resolver de manera local.

Variantes:
1) Código en demanda
    El cliente cuenta con los recursos y el poder de procesamiento
    El servidor mantiene el código a ser ejecutado
    El cliente le requiere el código al servidor y lo ejecuta
2) Ejecución remota/evaluación
    El cliente mantiene el código pero no tiene recursos para ejecutarlo (no tiene el “intérprete” de software o no tiene capacidad de procesamiento)
3) Agente móvil
    El iniciador tiene el código a ejecutar, pero no todos los recursos
    En forma autónoma decide migrar a otro nodo para obtener recursos adicionales

DESCRIPCIÓN El código se mueve para ser interpretado en otro host.

COMPONENTES dock de ejecución (recepción y deployment de código y estado) intérprete/compilador de código

CONECTORES Protocolos de red y elementos para empaquetar código y datos para transmisión

CUALIDADES Adaptabilidad dinámica Toma ventaja del poder de procesamiento del host Confianza - se incrementa al permitir migrar a un nuevo host de manera simple.

USOS TÍPICOS Cuando se procesan grandes conjuntos de datos en locaciones distribuidas (es más eficiente que el código se mueva al lugar donde se encuentran esos datos)

PRECAUCIONES Seguridad - La ejecución de código importado abre la puerta a malware. Cuando el costo de transmisión excede al costo de ejecución Cuando las conexiones de red no están disponibles

EVITARLO SI… La seguridad del código móvil no puede ser asegurada Se requiere estricto control de las versiones del software desplegado

ESTILOS DE INVOCACIÓN IMPLÍCITA

Se anuncian los eventos en vez de invocarse métodos
Los “listeners” se registran como interesados y asocian métodos (callbacks) con eventos.
Al producirse un evento, el sistema invoca a todos los métodos registrados
Quien anuncia el evento, no sabe a quién afectará
No hay suposiciones sobre el orden de procesamiento en respuesta a eventos

20. **Explicar las características que presenta una arquitectura de tipo publish/subscriber**

    DESCRIPCIÓN Subscribers se registran/desregistran para recibir mensajes o contenidos específicos. Cuando el Publisher publica, el mensaje es enviado a los Subscribers
    Es usado para enviar eventos y mensajes a un conjunto desconocido de receptores.
    El conjunto de receptores es desconocido para el productor del evento, la correctitud del productor no puede depender de los receptores.
    Nuevos receptores pueden agregarse sin cambios en el/los productor/es

    1) Basados en Listas :

        Cada Publisher mantiene una lista de suscripciones

    2) Basados en Broadcast:

        Los Publishers tienen poco (o ningún) conocimiento de los Subscribers.

Todos los eventos son emitidos a todos los Subscribers
Los Subscribers deben filtrar los eventos que son de su interés

3) Basados en Contenido

Los tópicos son tipos de eventos o mensajes predefinidos

VENTAJAS Desacoplamiento Escalabilidad

DESVENTAJAS Se agrega una capa de direccionamiento afectando la latencia, No se garantiza la entrega de mensajes, ni el orden en el que llegan, Disponibilidad

EJEMPLOS Interfaces de usuarios gráficas, Aplicaciones basadas en MVC o Ambientes de desarrollo extensibles, Listas de correo, Redes sociales

CONECTORES Llamadas a procedimientos pueden ser usadas dentro de un programa. Protocolos de red son más frecuentes.

ELEMENTOS DE DATOS o suscripciones o notificaciones o información publicada

TOPOLOGÍA Subscribers se conectan a Publishers en forma directa o pueden recibir notificaciones de intermediarios

CUALIDADES Altamente eficiente para distribuir información en un solo sentido con muy bajo acoplamiento de componentes.

USOS TÍPICOS o Distribución de noticias o GUIs o Juegos en red multi-player

PRECAUCIONES Cuando la cantidad de Subscribers para un tópico es muy grande, un protocolo especial puede ser necesario.

EVITARLO SI… No se dispone de un middleware para soportar un alto volumen de datos

21. Explicar las características que presenta una arquitectura basada en eventos

Componentes independientes comunicándose sólo enviando eventos a través de conectores a un event-bus

Los componentes emiten eventos al event-bus(puede haber mas de uno) en forma asincrónica, el cual luego los transmite a los otros componentes. Cada componente puede reaccionar ante la recepción de un evento, o ignorarlo.

Similar a Publish-Subcriber, pero en Event-Based no hay clasificación como en Publisher y Subcriber, todos pueden emitir y recibir eventos.

Los conectores se encargan de:
optimizar la distribución de eventos
la replicación de eventos (transparente al emisor y receptor)

OPTIMIZACIÓN Solamente distribuir eventos a quienes expresan interés en ellos

TIPOS DE DISTRIBUCIÓN

Pull (polling) - los componentes consultan al conector por eventos disponibles (bloqueante o no bloqueante)

Push - los eventos arriban a los componentes ni bien se producen

DESCRIPCIÓN Componentes independientes que asincrónicamente emiten y reciben eventos comunicados a través de event-buses.

ELEMENTOS DE DATOS Eventos

TOPOLOGÍA Los componentes se comunican con el event-bus, no directamente entre ellos.

CUALIDADES • Altamente escalable • Fácil de evolucionar • Efectivo para aplicaciones heterogéneas altamente distribuidas.

USOS TÍPICOS • Software de UI • Aplicaciones de área amplia que involucran partes independientes (mercados financieros, logística, redes de sensado)

PRECAUCIONES No existen garantías que un evento sea procesado, ni cuando lo será.

EVITARLO SI… Se requiere garantizar el procesamiento en tiempo real de los eventos

22. Explicar las características que presenta una arquitectura de tipo peer-to-peer

Consiste de una red de componentes autónomos y débilmente acoplados (pares) que colaboran para proveer un servicio.
Todos los componentes son iguales y ninguno puede ser crítico para la salud del sistema
Cada componente provee y consume los mismos servicios y usa el mismo protocolo
La información, por lo general, es mantenida localmente en cada componente.

Un componente puede interactuar con cualquier otro componente. La comunicación es típicamente una interacción requerimiento/respuesta. La interacción puede ser iniciada por cualquier parte (en el sentido clientserver) y cada componente es tanto cliente como servidor

La ausencia de centralización hace que la búsqueda de recursos sea un tema importante en P2P.

Tres opciones:

P2P Puro:

Una consulta es puesta en la red
El requerimiento se propaga hasta que la información es descubierta o hasta que algún umbral de propagación es alcanzado.
Si la información es localizada, el componente obtiene la dirección del otro componente y lo contacta

P2P Híbrido:

Ciertos componentes juegan un rol especial, o bien localizando otros pares o proveyendo directorios para localizar información

Servidor Índice:

Utiliza un servidor centralizado para indexar información y componentes

VENTAJAS

Escalabilidad: Los componentes pueden ser agregados o removido de la red sin un impacto significativo.

Disponibilidad: Si un componente deja de estar disponible, otros aún pueden proveer el servicio para completar la tarea.

Performance: La carga de cualquier componente actuando como servidor es reducida, ya que dicha carga es distribuida entre los componentes de la red.

DESVENTAJAS

Como un sistema peer-to-peer es descentralizado, algunas tareas son más complejas: manejar seguridad, consistencia de datos, disponibilidad de datos y servicios, backup y recuperación, etc.

Es difícil dar garantías porque los componentes van y vienen.

CUALIDADES Altamente robusto de cara a la falla de un nodo. Escalable, en términos de acceso a los recursos y poder de cómputo. Computación distribuida

USOS TÍPICOS File sharing Mensajería instantánea Grid computing

PRECAUCIONES Cuando la recuperación de la información es crítica en tiempo y no puede hacer frente a la latencia propuesta por el protocolo. Seguridad

EVITARLO SI… La confianza de los “peers” independientes no puede ser asegurada o administrada No se dispone de nodos designados para dar soporte al descubrimiento de recursos

23. Explicar las características que presenta el patrón de arquitectura MVC

Problema: Cómo mantener separadas la funcionalidad que corresponde a la interfaz, de la funcionalidad de la aplicación; y sin embargo, no dejar de responder a la interacción del usuario o a los cambios en los datos de la aplicación.

Separa la funcionalidad de la aplicación en tres clases de componentes:

MODELO: es la representación del estado (datos) de la aplicación y contiene (provee una interfaz) la lógica de la aplicación.

> Encapsula el estado de la aplicación
> Responde a consultas del estado
> Expone la funcionalidad de la aplicación
> Notifica a las vistas de cambios

VISTA: es un componente UI que muestra una representación del modelo al usuario y/o permite algún tipo de entrada de usuario. Interactúa con el controlador

> Renderiza el modelo
> Solicita actualizaciones a los modelos
> Envía los gestos del usuario al controlador
> Permite al controlador seleccionar vistas

CONTROLADOR: maneja la interacción entre el modelo y las vistas, traduciendo las acciones de usuario en cambios al modelo o cambios a la vista. Atiende los eventos

> Define comportamiento de la aplicación
> Mapea acciones de usuarios a actualizaciones del modelo
> Selecciona vistas para las respuestas
> Uno por cada funcionalidad

El componente modelo no debe interactuar directamente con el controlador

SEPARACION VISTA - MODELO: es uno de los más importantes principios de diseño, generalmente tienen intereses distintos

> Vista – se ocupa por mecanismos de UI y por cómo diseñar una buena UI
> Modelo – se piensa en términos de reglas de negocio y, quizás, de interacciones con la base de datos.

se podría querer ver la misma información del modelo de distintas formas
objetos no visuales son más fáciles de testear que objetos visuales
la presentación depende del modelo pero el modelo no depende la presentación

SEPARACIÓN 2 – CONTROLADOR – VISTA: No es tan importante, pero igualmente brinda beneficios. Permitiría tener más de un controlador por vista, o distintas vistas usar el mismo controlador

Ejemplo: Soportar comportamiento de edición y visualización con una vista. Podríamos tener 2 controladores, uno para cada caso, donde los controladores son Strategies (GoF) de la vista.

Usarlo si…. las separaciones indicadas anteriormente (especialmente la separación de la presentación y el modelo) son útiles.

Evitarlo si… se tiene una aplicación muy simple donde el modelo no tiene comportamiento. las tecnologías a utilizar no brindan la infraestructura necesaria.

24. Explicar las características que presenta el patrón de arquitectura SENSE-COMPUTE-CONTROL

Idea básica

Una computadora es embebida en alguna aplicación
Los sensores de distintos dispositivos están conectados a la computadora
Los sensores son interrogados para determinar su valor
La computadora también tiene asociados Actuators (dispositivos)
La computadora envía señales a los dispositivos a través de sus Actuators, logrando así controlar el sistema.

Dónde se usa?

Típicamente usado en aplicaciones de control embebidas, Simples electrodomésticos, Sistemas sofisticados para la industria automovilística Control robótico.

Cicla a través de los siguientes pasos:

Leer los valores de los sensores
Ejecutar un conjunto de leyes o funciones de control
Enviar la salida a los Actuators

Típicamente un ciclo está relacionado a un reloj

Frecuencia del reloj: Tasa máxima en que los valores de los sensores pueden cambiar, o en la sensibilidad con la que los Actuators pueden recibir actualizaciones.

25. Explicar las características que presenta el patrón de arquitectura BROKER

Algunos sistemas se construyen a partir de una colección de servicios distribuidos en múltiples servidores. Se requiere ocuparse de definir cómo los servicios van a interactuar (conectarse e intercambiar información) y manejar la disponibilidad de los servicios

PROBLEMA Cómo estructurar el software distribuido y que los usuarios de los servicios no necesitan conocer la naturaleza y ubicación de los proveedores de los servicios y hacer posible que cambien dinámicamente.

El broker es un intermediario que separa a los usuarios del servicio (clientes) de los proveedores del servicio (servidores)

FUNCIONAMIENTO

Cuando el cliente requiere de un servicio, envía la consulta al broker

El broker reenvía el requerimiento del cliente a un servidor para que lo procese. El server envía el resultado al broker

El broker retorna el resultado al cliente

El cliente sólo se puede asociar con un broker (potencialmente via proxy-cliente). El servidor solo se puede asociar con un broker (potencialmente via proxy-server)

COMPONENTES:

Client - consumidor de servicio
Server -proveedor de servicio
Broker - intermediario que localiza un server apropiado para cumplir el requerimiento de un cliente, reenvía el requerimiento al server y retorna el resultado al cliente
Client-side proxy - un intermediario que maneja la comunicación real entre el cliente y el broker
Server-side proxy - un intermediario que maneja la comunicación real entre el broker y el server

BENEFICIOS:

Los clientes no tienen que conocer ni la identidad, ni la dirección, ni las particularidades de comunicación de los servidores
Si un servidor se deja de estar disponible, el broker podría re-direccionar las peticiones a otro servidor que lo reemplace (disponibilidad)
Si un servidor es reemplazado por otro, solamente el broker deberá conocer sobre dicho cambio (modificabilidad)

DEBILIDADES

Agrega una capa de redirección y, por lo tanto, latencia entre clientes y servidores.
Dicha capa podría transformarse en un cuello de botella
El broker podría ser un único punto de falla
uede ser un objetivo de ataques de seguridad
Puede ser difícil de testear

26. Explicar las características que presenta una arquitectura SOA

SOA es una arquitectura de software donde todos los servicios y procesos implementados en software están diseñados como servicios a ser consumidos a través de una red.

• SOA describe una colección de componentes distribuidos que proveen y/o consumen servicios.
• Los servicios son, en gran medida, instalados independientemente y pertenecen, frecuentemente, a diferentes sistemas y organizaciones.

ENFOQUE:

El foco del diseño es la interfaz del servicio.

Un servicio:
- tiene una interfaz bien definida
- puede ser potencialmente invocado en una red
- puede ser reusado en múltiples contextos de negocio

Una aplicación:
- es integrada a nivel de interface y no a nivel de implementación
- es construida para trabajar con cualquier implementación de un contrato, resultando en un sistema menos acoplado y más flexible

ROLES

• proveedor de servicio:

- Crea una descripción de servicio
- Despliega el servicio en un entorno de ejecución para hacerlo disponible a otras entidades sobre la red
- Publica la descripción del servicio en uno o mas registros de servicios
- Recibe mensajes invocando el servicio por parte de los consumidores del servicio
- Cualquier entidad que aloja a un servicio web disponible a través de la red es un proveedor de servicios

EJ: Librería de apis de google

• consumidor de servicio

- encuentra la descripción de un servicio publicada en un registro de servicios
- aplica la descripción del servicio para ligar e invocar al servicio web alojado en el proveedor del servicio
- Un consumidor de servicio puede ser cualquier “entidad” que requiera de un servicio disponible en la red.

EJ: programa en html que consume la api de maps para mapa embebido

• registro de servicio

- acepta pedidos de los proveedores de servicios para publicar y difundir las descripciones de los servicios web

• permite que los consumidores de servicios busquen en la colección de descripciones de servicios contenida dentro del registro
• El rol del registro de servicios es permitir ligar a los proveedores de servicios con los consumidores.
• Una vez que se han ligado, las interacciones se realizan directamente entre el proveedor y el consumidor del servicio.

EJ: guia de la api?

OPERACIONES • publicar - • encontrar - • ligar

PROPIEDADES

Visión lógica: Los servicios son una abstracción de lo que los programas, bases de datos y procesos de negocio son capaces de hacer.

Abstracción: SOA esconde los detalles subyacentes de implementación – e.g. de los lenguajes, procesos, estructuras de base de datos, etc.

Relevancia de mensajes: Un servicio es definido en término de los mensajes que se intercambian entre el agente proveedor y el consumidor, no en términos de las propiedades de los agentes en sí mismos.

Operaciones: Un servicio tiende a depender de un número pequeño de operaciones con mensajes relativamente largos y complejos.

Red: Los servicios están definidos para ser usados sobre una red.

Independencia de plataforma: Los mensajes son enviados en un formato estandarizado definido, usualmente XML, enviados a través de interfaces.

BENEFICIOS

SOA permite que los agentes que participan en el intercambio de mensajes estén débilmente acoplados; esto permite una mayor flexibilidad:

o un cliente sólo se acopla a un servicio, no a un servidor - la integración del servidor se lleva a cabo fuera del ámbito de la aplicación cliente

o las componentes funcionales y sus interfaces están separadas - nuevas interfaces pueden ser fácilmente añadidas

o funcionalidad vieja y la nueva se puede encapsular como componentes de software que proporcionan y solicitan servicios

o los servicios se pueden incorporar de forma dinámica en tiempo de ejecución

POTENCIALES PROBLEMAS

o complejidad en el diseño e implementación, debido al binding dinámico y el uso de metadatos

o overhead en la performance del middleware (intermediario) que se interpone entre servicios y clientes

o carencia de garantía de performance, ya que los servicios se comparten y no están bajo el control del cliente

27. Explicar las características que presenta una arquitectura MAP-REDUCE

en la actualidad, las organizaciones necesitan analizar rápidamente grandes volúmenes de datos que generan en un orden de petabyte (1015 bytes)

- interacciones en redes sociales
- repositorios de datos o documentos masivos
- pares para un motor de búsqueda.

se debe asegurar eficiencia y resiliencia con respecto a fallas en el hardware

Map-Reduce provee un marco de trabajo para analizar un conjunto de datos grande y distribuido, que se ejecuta en paralelo sobre un conjunto de procesadores. El paralelismo permite baja latencia y alta disponibilidad

Requerimientos: Una infraestructura especializada que se encargue de:

- alocar los componentes de software a los nodos de hardware en un ambiente de computación masivamente paralelo, asegurando disponibilidad y recuperación ante caídas
- manejar el ordenamiento de los datos como se necesite

dos funciones: map y reduce - definidas en términos de tuplas

Map(k1,v1) -> list(k2,v2):

toma una lista de pares en un dominio de datos y devuelve una lista de pares en un dominio diferente (datos intermedios)

su propósito es filtrar el dataset, determinando si un registro va a estar involucrado en el procesamiento posterior

k2 es importante ya que será usada para ordenar (agrupar) los datos de salida de esta función

es aplicada en paralelo a cada ítem (o lote de ítems) de la entrada de datos

el resultado de esta función es tomado por la infraestructura que junta los pares con la misma clave y los agrupa, creando un grupo por c/u de las claves

Reduce(k2,list (v2)) -> list(v3))

es aplicada en paralelo a cada grupo, produciendo una colección de valores para cada dominio

el conjunto de datos de salida es siempre mucho más chico que el de entrada

¿Cuándo NO es apropiado este patrón?

• si no existen grandes volúmenes de datos – ya que el overhead de MapReduce no se justifica
• si no se puede dividir el conjunto de datos inicial en subconjuntos de tamaño uniforme – en este caso, las ventajas del paralelismo se diluyen
• si se tienen operaciones que requieren múltiples “reduce” - aunque es posible, es más difícil de orquestar.

EJ: MongoDB, Hadoop, CouchDB

VISTAS: Una vista es un punto de vista/ una forma de ver el sistema:

Determinados atributos de calidad son expuestos mejor por ciertas vistas. Cada vista enfatiza ciertos aspectos del sistema, mientras des-enfatiza o ignora otros, con el objetivo de hacer el problema tratable.

ES MUY DIFÍCIL DOCUMENTAR UN SISTEMA ENTERO COMO UNA ÚNICA PIEZA, ES MAS FACIL PARTICIONARLO EN VISTAS

CUANDO VEMOS UNA SOLA DE LAS VISTAS NO NOS DAMOS CUENTA DE LA DIMENSIÓN DEL SISTEMA y entender en forma completa e integral todo el sistema, por eso necesitamos todas las vistas.

¿Qué contiene la documentación de un vista?

- una presentación primaria, usualmente gráfica, mostrando los principales elementos y las relaciones de la vista
- un catálogo de elementos que explica y define los elementos mostrados en la vista y lista sus propiedades
- una especificación de las interfaces y comportamiento de dichos elementos.
- indicaciones sobre los mecanismos incorporados que permitirían adaptar la arquitectura
- justificación e información de diseño

¿Qué contiene la información que aplica a todas las vistas?

- una introducción al paquete entero, incluyendo una guía de lector que ayuda a los interesados a encontrar rápidamente la información que necesitan
- información describiendo cómo las vistas se relacionan entre sí y con el sistema como un todo
- restricciones y justificaciones para la arquitectura global

28. Explicar para qué sirve una vista de módulos (a partir de un estilo)

Se usa para saber cómo está estructurado como un **conjunto de unidades de código**

Los módulos reflejan la forma en que un sistema de software es descompuesto en unidades administrables de responsabilidad, que es uno de los aspectos más importantes de la estructura de un sistema

Los módulos se relacionan entre sí por las siguientes relaciones: es-parte-de , depende-de, es-un.

- provee un plano para la construcción del código
- explica la funcionalidad del sistema y la estructura del código base
- facilita el análisis de impacto
- da soporte al análisis de trazabilidad de requerimientos
- ayuda en la planificación del desarrollo incremental
- da soporte a la definición de asignaciones de trabajo, planificación de implementaciones e información de presupuesto
- muestra la estructura de la información a ser persistida

NOTA: como brinda una representación del sistema estática más que en ejecución, NO es útil para analizar performance, confiabilidad, etc.

29. Explicar para qué sirve una vista de componentes y conectores

Cómo está estructurado como un **conjunto de elementos que tienen comportamiento e interacciones en tiempo de ejecución**

USOS:

- Mostrar a los desarrolladores y otros interesados cómo funciona el sistema:

  • ¿Cuáles son los principales componentes del sistema y cómo interactúan?
  • ¿Cuáles son los principales almacenamientos compartidos?
  • ¿Qué partes del sistema están replicadas? ¿Cuántas veces?
  • ¿Cómo fluyen los datos en el sistema?
  • ¿Qué partes del sistema corren en paralelo?

- Guiar el desarrollo especificando la estructura y comportamiento de los elementos en ejecución
- Razonar acerca de los atributos de calidad en tiempo de ejecución:

  • performance, confiabilidad, disponibilidad

<u>COMPONENTES:</u> Un componente representa a los principales elementos computacionales o de almacenamiento presentes en ejecución. Puede representar a un subsistema complejo, cuya sub-arquitectura puede ser mostrada en el mismo

diagrama (anidada al componente) o en otro. Los componente tienen interfaces llamadas puertos:

procesos, objetos, clientes, servidores, almacenamiento de datos

CONECTORES: Es el camino en ejecución para la interacción entre dos o más componentes:

enlaces de comunicación, protocolos de comunicación, flujos de información, accesos a almacenamientos compartido

Los conectores tienen roles. El rol define la forma en la cual un conector puede ser utilizado por los componentes para llevar adelante la interacción.

Vistas de componentes y conectores bien documentadas permiten predecir las propiedades del sistema total, a partir de las estimaciones o mediciones de las propiedades de elementos e interacciones individuales.

Ejemplos:

o Para determinar el cumplimiento de los requerimientos de planificación en tiempo real, se necesita conocer el tiempo de ejecución de cada proceso componente.

o La confiabilidad de los elementos individuales y de los canales de comunicación ayudan al arquitecto para calcular la confiabilidad total del sistema.

RESTRICCIONES

Los componentes pueden ser asociados solamente a conectores, no a otros componentes
Los conectores pueden ser asociados solamente a componentes, no a otros conectores.
Attachments sólo pueden ser hechos solamente entre puertos y roles compatibles
Los conectores no pueden aparecer en forma aislada. Un conector debe estar asociado a un componente.

30. Explicar para qué sirve una vista de asignación (a partir de un estilo)

Cómo se relaciona con otros elementos no de software – tales como hardware y equipos de desarrollo?

Las vistas de asignación presentan un mapeo entre los elementos de software (ya sea de una vista de módulo o de componentes y conectores) y los elementos no-software:

• hardware de computación y comunicación
• sistemas de manejo de archivos
• equipos de desarrollo

Qué permite el mapeo de la arquitectura de software con…

• el hardware? Que la performance del sistema pueda ser analizada
• la estructura de archivos? Que el sistema en producción se pueda gestionar
• la estructura del equipo? Que puedan proceder las actividades de administración de proyecto

TIENE 3 ESTILOS:

Estilo de Despliegue: describe el mapeo entre los componentes y conectores de software y el hardware de la plataforma sobre el cual ejecuta el software

Usos: analizar performance, disponibilidad, confiabilidad y seguridad - entender las dependencias en ejecución (uso por los testeadores) - facilitar la estimación de costos de hardware

Estilo de Instalación: describe el mapeo entre los componentes de software y las estructuras en el sistema de archivos del ambiente de producción

Usos: crear procedimientos de build-and-deploy - navegar a través de una gran cantidad de archivos y carpetas que constituyen el sistema instalado, y localizar archivos específicos (logs o configuración) - seleccionar y configurar archivos para una versión específica - identificar el propósito de un archivo faltante o dañado que trae problemas. - diseñar e implementar características de actualización automática

Estilo de Asignación de Trabajo: describe el mapeo entre los módulos de software y las personas, equipos o unidades de trabajo que tienen que desarrollar dichos módulos

Usos: mostrar quién producirá cada unidad de software - ayudar a planificar y gestionar la asignación de equipos

31. Explicar un patrón de aplicaciones empresariales de la capa de dominio

TRANSACTION SCRIPT:

DESCRIPCIÓN - organiza la lógica de negocio por procedimiento, donde cada procedimiento maneja un único pedido desde la capa de presentación.

un Transaction Script organiza cada transacción de negocio (funcionalidad) en un único procedimiento, haciendo las llamadas pertinentes a la capa de acceso a datos.

cada transacción de negocio tendrá su propio Transaction Script, aunque sub-tareas comunes pueden ser aisladas en sub-procedimientos.

Un Transaction Script se podría resumir en validar los datos de entrada, consultar la base de datos, realizar cálculos y guardar los resultados en la base de datos

¿Dónde poner los Transaction Scripts? o una server page o un método o una clase o un conjunto de clases

RECOMENDACIONES

separar los Transaction Script tanto como sea posible - al menos, ponerlos en distintos métodos; mejor aún si están en clases distintas a las usadas en Presentación y Acceso a Datos.

no realizar ninguna llamada desde un Transaction Script a lógica de presentación – para facilitar el testeo y mantenimiento

poner los Transaction Scripts en clases de una de la siguiente forma:

- tener varios TS en una única clase, donde cada clase define un área sujeto de TS relacionados. Esto es directo y la más apropiada en muchos casos. Ejemplo: NotasService, ClasesService, etc.
- tener cada TS en su propia clase utilizando el patrón Command1 (GoF)

Ventajas

Dada su simplicidad, es natural utilizarlo en aplicaciones que tengan poca lógica de negocios, ya que involucra poco overhead tanto en performance como en curva de aprendizaje.

Limitaciones

A medida que la lógica de negocio se torna más compleja, es difícil mantenerlo con un buen diseño.

Generalmente se tiene problemas con la duplicación de código. Puesto que el foco está en resolver una transacción, muchas veces se tiende a duplicar código común.

32. Explicar un patrón de aplicaciones empresariales de la capa de datos que sirva para modelar el comportamiento objeto-relación

MOTIVACIÓN

Como hacer para cargar los objetos a memoria y salvarlos en la base de datos?

Si se cargan varios objetos en memoria y se actualizan, se debe llevar registro de cuales se actualizaron para asegurarse que son grabados en la base de datos.

NOS VAN A PERMITIR ASEGURAR LA CONSISTENCIA EN LA BD.

Si se leen objetos, es necesario saber si pueden o no actualizar simultáneamente – problema de concurrencia.

Hay 3: Unit of Work - Identity Map - Lazy Load

IDENTITY MAP:

- Asegura que cada objeto sea cargado en memoria solamente una vez, manteniendo cada objeto cargado en un map. Cuando se referencia a un objeto, en vez de ir a buscar directamente en la bd, primero se lo busca a través del map.

Si no se tiene cuidado, podríamos cargar los datos del mismo registro de base de datos en dos objetos diferentes y modificarlos. En ese caso podríamos tener problemas de inconsistencia de información y de performance.

Como trabaja?

Mantiene registro de todos los objetos que han sido leídos desde la base de datos en una única transacción de negocio.

Cuando se quiere obtener un objeto, primero se busca en el map por si ya está en memoria. Si no está, se busca en la base de datos y se lo incorpora al map.

Si se utiliza el concepto de Unit of Work, entonces el Unit of Work es el mejor lugar para el Identity Map.

También sirve como caché de los objetos obtenidos en una transacción de negocio.

33. Explicar un patrón de aplicaciones empresariales de la capa de datos que sirva para modelar el mapeo estructural objeto-relación

MOTIVACION – Cuando se habla de mapeo objeto relacional (ORM) se trata de como mapear objetos en memoria con las tablas de una base de datos relacional. El tema central es el enfoque diferente para mapear links entre la orientación a objetos y las bases de datos relacionales.

– IDENTITY FIELD

DESCRIPCION - Guarda el campo ID de la base de datos en el objeto para mantener la identidad entre un objeto en memoria y la fila de la base de datos. Para que cuando vayamos a la tabla, sepamos qué registro leer.

Para que funcionen, las claves deben ser únicas… pero para que funcionen bien, también deben ser inmutables.

Cómo definimos la clave?

o Claves con o sin sentido

• Meaningful key – Un atributo que la entidad ya tiene. Por ejemplo, el número de documento de una persona
• Meaningless key – Número aleatorio que no sería utilizado por humanos. Ej: LU. Más simple para el sistema. Más difícil para integrar sistemas y ver q es la misma persona(se deben usar tablas de equivalencia)

o Claves simples o compuestas

• Clave simple - solamente usa un campo de la base de datos
• Clave compuesta - usa más de un campo de la base de datos

o Alcance de unicidad de las claves

• Únicas por tabla
• Únicas por base de datos (ideal por cuestiones de integración)

o Tipo y tamaño

• las operaciones más frecuentes son: comparar igualdad y obtener la siguiente siguiente clave
• pueden tener efecto en la performance e índices de las bases de datos, mientras más larga sea la clave, más bytes, más transferencia, impacta en performance
• generalmente se utilizan: long, GUID (Globally Unique Id) e int

USOS:

Representar el Identity Field en un objeto

• la forma más simple de Identity Field es un campo que concuerde con el tipo de la clave en la base de datos
• para las claves compuestas, generalmente se utiliza una clase que las representa
• muchas veces se factoriza la definición de la clave (junto con la operación de igualdad) en una clase de entidad base y ese comportamiento es heredado por todas las entidades del sistema Ej: clase persona y la clave es dni y número

Obtener una nueva clave

• delegar en la base de datos y que la auto-genere
• usar un GUID
• generar propias (max function, tabla de claves separada)

34. Explicar un patrón de aplicaciones empresariales de la capa de presentación

MOTIVACIÓN – Una de los cambios más importantes en las aplicaciones empresariales en los últimos tiempos fue la utilización de las interfaces de usuario (IU) basadas en la Web. Ventajas de IU basadas en la Web: • No se necesita instalar software en el cliente • Se puede adoptar un enfoque común de IU • Se provee un fácil acceso universal

PAGE CONTROLLER:

DESCRIPCIÓN – Un objeto que maneja un request para una página o acción de una aplicación web

Page Controller tiene un controlador para cada página lógica de la aplicación web. El controlador puede ser la página en sí misma (frecuente en ambiente server pages), o puede se un objeto separado que se corresponda con la página. Sin embargo, este patrón de diseño apunta a una alternativa de un path que conduce a un archivo que maneja un requerimiento

IDEA BÁSICA: Un módulo (clase o lo que fuera) actúa como controlador para cada página de la aplicación web. En la realidad, existirá un controlador por cada acción (incluidas los eventos)

RESPONSABILIDADES

• Decodificar la URL y extraer cualquier dato del request que sea necesario para realizar la acción requerida.
• Crear e invocar objetos del modelo para procesar los datos de entrada. Los objetos del modelo no necesitan conocer al request.
• Determinar qué vista se debe mostrar y enviarle el modelo que corresponda.

RECOMENDACIONES

No necesita ser una única clase. Se podrían tener ambas alternativas en una misma aplicación

CUANDO USARLO?

SI - Cuando mucha de la lógica del controlador es muy simple, ya que no agrega demasiado overhead.

NO - Cuando hay mucha complejidad navegacional, sería preferible usar un Front Controller.

28. Explicar para qué sirve una vista de módulos (a partir de un estilo)

Se usa para saber cómo está estructurado como un **conjunto de unidades de código**

Los módulos reflejan la forma en que un sistema de software es descompuesto en unidades administrables de responsabilidad, que es uno de los aspectos más importantes de la estructura de un sistema

Los módulos se relacionan entre sí por las siguientes relaciones: es-parte-de , depende-de, es-un.

- provee un plano para la construcción del código
- explica la funcionalidad del sistema y la estructura del código base
- facilita el análisis de impacto
- da soporte al análisis de trazabilidad de requerimientos
- ayuda en la planificación del desarrollo incremental
- da soporte a la definición de asignaciones de trabajo, planificación de implementaciones e información de presupuesto
- muestra la estructura de la información a ser persistida

NOTA: como brinda una representación del sistema estática más que en ejecución, NO es útil para analizar performance, confiabilidad, etc.

ESTILO DE USOS

DESCRIPCIÓN o muestra cómo los módulos dependen unos de otros o un módulo dependen de otro si su correctitud depende de la correctitud del otro

RELACIONES La vista documenta la relación usa, la cual es una forma de la relación depende-de. Un módulo A usa un módulo B si A depende de la presencia de B correctamente funcionando para satisfacer sus propios requerimientos.

RESTRICCIONES No tiene restricciones topológicas. Sin embargo, si las relaciones de uso presentan ciclos, amplios abanicos, o largas cadenas de dependencia, la capacidad de que la arquitectura sea entregada en subconjuntos incrementales se verá dañada.

USOS:
o planificar el desarrollo incremental
o debugging y testing
o medir los efectos de los cambios

Notación Informal

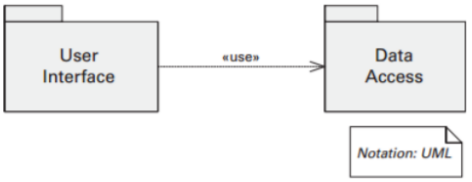
- Tablas de dos columnas - Elementos y Elementos usados

Notación Semiformal – UML

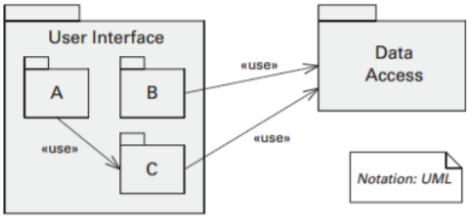
- Módulos → Paquetes

- *usa* → Relación de dependencia con estereotipo <<*use*>>

(a)

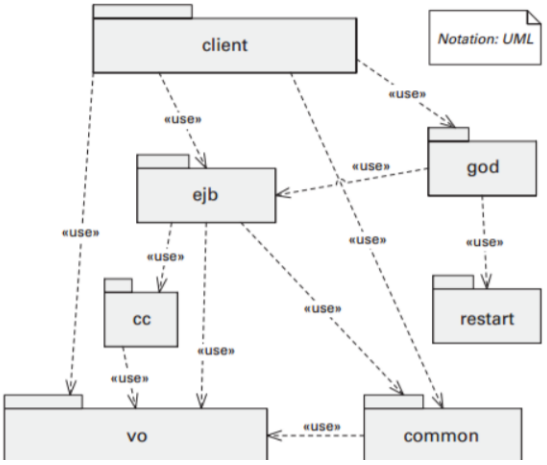


(b)



Notación Semiformal – Matriz de Dependencia

Los siguiente diagramas son intercambiables y expresan lo mismo.



using module / used module	client	ejb	cc	god	restart	common	vo
client	0	0	0	0	0	0	0
ejb	**1**	0	0	**1**	0	0	0
cc	0	**1**	0	0	0	0	0
god	**1**	0	0	0	0	0	0
restart	0	0	0	**1**	0	0	0
common	**1**	**1**	0	0	0	0	0
vo	**1**	**1**	**1**	0	0	**1**	0

Key: "1" means module in column uses module in row

ATRIBUTO DE CALIDAD - USABILIDAD

CALIDADES DEL USUARIO

La usabilidad define cómo de bien la aplicación satisface los requerimientos del usuario y del consumidor de ser intuitiva, fácil de localizar y globalizar, proveyendo buen acceso para personas con discapacidad, y resultando, en general en una buena experiencia del usuario

¿Se relaciona con la arquitectura del software?

- El contraste entre el texto y el fondo para garantizar que sea legible.
- La distribución de los objetos en pantalla puede hacer que "no sea visible" el botón guardar cambios.
- El ATM debe entregar el comprobante de una transacción cuando la transacción ya fue registrada exitosamente y guardó datos de auditoría.
- En el formulario web, si la validación de datos ingresados no es exitosa, la aplicación informa al usuario sin que se pierden los datos cargados

ATRIBUTO DE CALIDAD - MANTENIBILIDAD

CALIDADES DE DISEÑO

La mantenibilidad es la habilidad de un sistema de someterse a cambios con un grado de facilidad. Esos cambios pueden afectar componentes, servicios, características e interfaces cuando se agrega o se cambia la funcionalidad corrigiendo errores y satisfaciendo nuevos requerimientos de negocios.

¿Se relaciona con la arquitectura del software?

- la distribución del código en módulos
- el modelo de asignación de responsabilidades a los módulos
- la documentación del sistema
- el uso de estándares de programación y documentación de programas
- mantener actualizada la documentación

ATRIBUTO DE CALIDAD - PERFORMANCE

CALIDADES DE TIEMPO DE EJECUCIÓN

La performance es una indicación de la respuesta de un sistema para ejecutar una acción dentro de un intervalo de tiempo dado. Puede ser medido en términos de latencia o rendimiento. Latencia es el tiempo requerido para responder a un evento. Rendimiento es el número de eventos que se procesan dentro de una determinada cantidad de tiempo,

¿Se relaciona con la arquitectura del software?

- las estructuras de programación usadas
- el orden y complejidad de los algoritmos
- la capacidad y organización del hardware
- la distribución física de los componentes en los servidores
- el modelo de comunicación entre componentes

USABILIDAD	
Arquitectura	**No Arquitectura**
○ Permitir cancelación de operaciones ○ Permitir deshacer operaciones ○ Reusar datos previamente ingresados	○ Hacer la interfaz de usuario clara y fácil de usar. ○ ¿Radio button o checkbox? ○ Diseño de pantalla ○ Tipografía

MANTENIBILIDAD	
Arquitectura	**No Arquitectura**
▪ ¿Cómo la funcionalidad es dividida en componentes? ▪ Cohesión, acoplamiento y dependencia entre componentes	▪ Técnicas de codificación dentro de un módulo ▪ Estándares de codificación

PERFORMANCE	
Arquitectura	**No Arquitectura**
▪ Cuánta comunicación es necesaria entre componentes ▪ Qué funcionalidad ha sido asignada a cada componente ▪ Cómo son asignados los recursos compartidos	▪ La elección de los algoritmos para implementar determinadas funcionalidades ▪ Cómo estos algoritmos son codificados.

El **REFACTORING** es el proceso de cambiar un sistema de software de manera que no altere el comportamiento externo del código pero que mejore su estructura interna.
Es una forma disciplinada de limpiar el código que minimiza las posibilidades de introducir errores.
En esencia, cuando se refactoriza, se mejora el diseño del código después de que ha sido escrito.

La refactorización consiste en tomar un diseño malo, incluso caótico, y volver a trabajar en un código bien estructurado.

La **inyección de dependencias** es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree dichos objetos. Esos objetos cumplen contratos que necesitan nuestras clases para poder funcionar. Nuestras clases no crean los objetos que necesitan, sino que se los suministra otra clase 'contenedora' que inyectará la implementación deseada a nuestro contrato.[1]

En otras palabras, se trata de un patrón de diseño que se encarga de extraer la responsabilidad de la creación de instancias de un componente para delegarla en otro.

Favorece la **modularidad**, **reutilización** y **testeo.**

Ej: Si tengo múltiples repos que usan una bd, no tengo que crear múltiples bds.

Puede inyectarse en el constructor o con un setter. Con el constructor nos aseguramos de que al crear el objeto, las dependencias ya existen. Con setter deben ser nullables y es mas quilombo.

Los modulos son los encargados de crear las instancias, y son los inyectores de dependencias.

MVP: En vez de tener un controlador tenemos un presentador. La vista en vez de escuchar al modelo, escucha al presentador y este es quien se encarga de orquestar el estado y las actualizaciones. Si la vista escucha el modelo, este modelo empieza a tener observadores por todos lados. Entonces se agrupan a través de un presentador y queda más balanceada la arquitectura.

MVVM: Muy similar al MVP, en vez de un presentador tenemos un viewmodel, la diferencia es que se piensa al presentador como estados de la vista, no como lógica de negocios. La vista se liga a ciertos fields a través de algún framework y ser actualiza automáticamente. Cuando creo una vista digo “esta vista se actualiza en base a esta propiedad de este viewmodel” la vista conoce al vm pero el vm no conoce a la vista. El vm define lo que serían campos observables, llama al model y actualiza esos observables y automáticamente en la vista se actualizan esos cambios.

MVI: Sigue la misma idea, la vista dispara acciones, el controlador/presentador dispara eventos al modelo, escucha los resultados. La diferencia es que hay un componente reducer entre el controlador y la vista que se encarga de tomar el estado anterior de la vista (se la piensa como estados), tomar algún dato nuevo (algún obj del modelo) y resolver el nuevo estado y se lo expone a la vista.