Buenos días Don Daniel. ¡Vargas Cipollo!

Tema 2: Arquitecturas Software

1. Conceptos básicos

- **Arquitectura de un sistema software**
    - Estructura de las estructuras: elementos, relaciones y principios de diseño.
    - Incluye:
        1. **Estructuras estáticas**: clases, paquetes, módulos.
        2. **Estructuras dinámicas**: flujos de información en tiempo de ejecución.
- **Elemento arquitectónico**
    - Pieza fundamental (módulo/componente) con:
        - Conjunto de funciones.
        - Frontera (límite).
        - Interfaces visibles.
- **Propiedades visibles**
    1. **Comportamiento funcional**: interacciones con el entorno.
    2. **Propiedades de calidad** (no funcionales): rendimiento, seguridad, mantenibilidad.
- **Principios de arquitectura**
    - Guían diseño y evolución (ej. modularidad, cohesión, acoplamiento).
    - Deben satisfacer intereses de los *stakeholders*.
- **Stakeholders**
    - Individuos o equipos con interés en el sistema (desarrollo, prueba, mantenimiento).
    - Cada uno tiene **intereses** (requisitos, restricciones, objetivos).
    - Importante equilibrar intereses que pueden chocar.

---

2. Vistas y puntos de vista

2.1 ¿Por qué usar vistas?

- Sistemas complejos no se capturan en un único modelo.
- **Vista**: representación parcial que aborda intereses de un grupo de stakeholders.
- Ventajas:
    - Separación de intereses.
    - Facilita comunicación.

    - Manejo de complejidad.

- Inconvenientes:

    - Posible inconsistencia entre vistas.
    - Selección de vistas erróneas provoca fragmentación.

2.2 Modelo de vistas 4+1 (R. Kruchten)

1. **Vista lógica**

    - Representa estructura funcional.
    - Diagramas: clases, estados.
    - Stakeholders: diseñadores, analistas.

2. **Vista de desarrollo**

    - Estructura de módulos/componentes del software.
    - Diagramas: paquetes, componentes, capas.
    - Stakeholders: responsables de mantenimiento y evolución.

3. **Vista de procesos**

    - Concurrencia y sincronización en tiempo de ejecución.
    - Diagramas: secuencia, comunicación, actividad.
    - Stakeholders: ingenieros de rendimiento, operaciones.

4. **Vista física (despliegue)**

    - Infraestructura: nodos donde se ejecuta.
    - Diagramas: despliegue (nodos y conexiones).
    - Stakeholders: administradores de sistemas.

5. **Escenarios (casos de uso)**

    - Ilustran cómo las 4 vistas trabajan juntas.
    - Diagramas: casos de uso.
    - Permiten validar decisiones de arquitectura.

2.3 Modelo de puntos de vista alternativo (Rozanski & Woods)

- **Puntos de vista**: colecciones de convenciones para construir un tipo de vista.

- Cada punto de vista define:

    1. Stakeholders atendidos.
    2. Directrices y plantillas.

- Principales puntos de vista:

    1. **Contexto**

        - Relación sistema–entorno (usuarios, sistemas externos).

        - Representación: diagrama de contexto.

    2. **Funcional**

        - Elementos funcionales y sus interacciones en ejecución.
        - Representación: diagrama de secuencia, colaboración.

    3. **Información**

        - Cómo se gestiona, almacena y distribuye la información.
        - Representación: diagramas de entidades-relaciones, flujo de datos.

    4. **Concurrencia**

        - Partes que pueden ejecutarse simultáneamente; coordinación.
        - Representación: diagramas de estado, actividad con hilos/procesos.

    5. **Desarrollo**

        - Estructura que soporta el proceso de desarrollo (módulos, repositorios).
        - Representación: diagrama de componentes, paquetes.

    6. **Despliegue**

        - Entorno de ejecución (servidores, redes, dependencias).
        - Representación: diagrama de despliegue detallado.

    7. **Operacional**

        - Cómo se opera, administra y soporta el sistema en uso.
        - Aspectos: instalación, monitorización, logs, backups.

- **Perspectivas** (transversales)

    - Aseguran propiedades de calidad (seguridad, confiabilidad) en todas o algunas vistas.

---

3. Patrones arquitectónicos

Patrones de alto nivel que organizan estructura fundamental de un sistema. Cada patrón: esquema, responsabilidades, reglas de organización.

3.1 Patrones básicos tratados

1. **Capas (Layered Architecture)**
2. **Tuberías y filtros (Pipes & Filters)**
3. **Pizarra (Blackboard Architecture)**

3.2 Patrón de Capas

- **Idea**: descomponer sistema en niveles de abstracción jerárquicos.
- Cada capa usa servicios de la capa inferior y ofrece servicios a la capa superior.

A. Descomposición

1. Identificar la capa base (servicios de bajo nivel: acceso a disco, dispositivos).
2. Añadir capa tras capa, cada una abstrae servicios de la anterior.
3. Interfaces bien definidas entre capas.

B. Escenarios de interacción

1. **Flujo de solicitudes (de arriba abajo)**
    - La capa superior delega en la siguiente si no puede procesar sola.
    - Respuestas viajan de abajo arriba.
2. **Notificaciones (de abajo arriba)**
    - Capa baja detecta evento (ej. entrada sensor).
    - Propaga hacia capa superior como notificación.
3. **Caché intermedia**
    - Capa intermedia procesa y responde si tiene datos, evitando bajar a capa más baja.
4. **Interacción limitada**
    - Capas sólo comunican si la capa inferior está habilitada para procesar.
5. **Pilas de protocolo**
    - Dos pilas idénticas (cliente/servidor) se comunican capa a capa.

C. Ventajas

- Reutilización de capas bien definidas (módulos desacoplados).
- Apoyo a estandarización (interfaces comunes).
- Dependencias locales: cambios afectan solo capas adyacentes.
- Intercambiabilidad: sustituir implementaciones sin cambiar resto.

D. Inconvenientes

- Cambios en cascada: modificar capa inferior puede impactar capas superiores.
- Menor eficiencia: datos pasan por varias capas.
- Trabajo innecesario de capas inferiores que no aportan valor a capas superiores.

3.3 Patrón Tuberías y Filtros

- **Contexto**: sistemas que procesan flujos de datos secuenciales.
- **Componentes**:
    1. **Filtro**: unidad de procesamiento que transforma o refina datos.
        - Activo: extrae y empuja datos.
        - Pasivo: espera ser invocado por el tubo.

2. **Tubo**: conecta filtros, fuente y sumidero; buffer FIFO.
3. **Fuente de datos**: provee secuencia de datos (archivo, sensores).
4. **Sumidero de datos**: recoge resultados (archivo, terminal).

- **Tipos de tubería**
  1. **Push**: fuente empuja datos al siguiente filtro.
  2. **Pull**: filtro solicita datos a la fuente.
  3. **Mixta**: combinación de push y pull.
  4. **Compleja**: múltiples entradas y salidas, filtros interconectados.
- **Flujo de datos**
  - Datos: Fuente → Tubo → Filtro → Tubo → ... → Sumidero.
- **Ventajas**
  - Flexibilidad: reordenar/intercambiar filtros sin cambiar todo el sistema.
  - Reutilización: filtros independientes.
  - Prototipado rápido: ensamblar tuberías existentes y optimizar.
  - Posibilidad de paralelismo (varios filtros procesando simultáneamente).
- **Inconvenientes**
  - Menor eficiencia si comparten muchos datos entre filtros.
  - Overhead en transmisión de datos cada vez que pasa por un tubo.
  - Sincronización frecuente (buffers pequeños pueden detener filtros).
  - Manejo de errores complejo (donde reportar y cómo recuperar).
- **Ejemplo práctico**
  - Sistema de reconocimiento de voz:
    1. **Fuente**: onda de audio.
    2. **Filtros**:
       - Filtrado de ruido.
       - Extracción de características acústicas.
       - Reconocimiento fonético.
       - Modelado de lenguaje.
    3. **Sumidero**: texto transcrito.

3.4 Patrón Pizarra (Blackboard)

- **Contexto**: problemas sin algoritmo determinístico, requieren colaboración de expertos de distintas áreas.
- **Componentes**:

1. **Pizarra**: estructura de datos central que contiene hipótesis (soluciones parciales).
2. **Fuentes de conocimiento (módulos/experts)**:
    - No se comunican directamente; solo leen/escriben en pizarra.
    - Cada módulo conoce cuándo puede contribuir.
3. **Control**:
    - Coordina activación de fuentes según estado de la pizarra.
    - Planea evaluaciones y jerarquiza hipótesis.

- **Funcionamiento**
    1. La pizarra se inicializa con datos de entrada (ej. señales de audio).
    2. Módulos leen pizarra, añaden hipótesis (fragmentos de solución).
    3. Control selecciona qué módulo aplicar a continuación.
    4. Hipótesis se combinan/rechazan hasta obtener solución final.
- **Ventajas**
    - Separación clara: pizarra, módulos y control.
    - Reutilización de módulos especializados.
    - Tolerancia a fallos: hipótesis erróneas se eliminan.
- **Inconvenientes**
    - Dificultad de testeo: ejecución no determinística, resultados variables.
    - Complejidad de control: decide qué módulo ejecutar y cuándo.
    - Eficiencia baja: revisar y descartar muchas hipótesis.
    - Sin paralelismo real sin agregar mecanismos extra; sincronización costosa.

---

4. Validación de la arquitectura

- **Objetivo**: asegurar que la arquitectura propuesta cumple requisitos funcionales y no funcionales, y es la más adecuada frente a alternativas.
- **Criterios de validación**
    1. **Adecuación**: ¿satisface las necesidades del sistema?
    2. **Comparación**: ¿es mejor que otras opciones?
    3. **Cumplimiento de propiedades de calidad**: rendimiento, escalabilidad, seguridad.
    4. **Factibilidad**: recursos (personal, tiempo, licencias) disponibles.
- **Técnicas de evaluación**
    1. **SAAM (Software Architecture Analysis Method)**
        - Emplea casos de uso para verificar funcionalidad y capacidad de adaptación a cambios.
        - Pasos básicos: identificación de escenarios, evaluación de impacto en la arquitectura, generación de informe de recomendaciones.

2. **ATAM (Architecture Tradeoff Analysis Method)**
    - Extiende SAAM incluyendo escenarios de atributos de calidad (ej. rendimiento vs mantenibilidad).
    - Pasos: recopilación de objetivos de calidad, generación y clasificación de escenarios, evaluación de riesgos y sensibilidad, análisis de trade-offs.

- **Ejercicios de validación práctica**
    1. **Ejercicio SAAM simple**:
        - Dado un sistema de reservas aéreas (véase ejemplo), definir 3 casos de uso críticos (reservar, cancelar, modificar).
        - Analizar cómo la arquitectura en capas atendería cada caso (¿qué capas intervienen, posibles cuellos de botella?).
        - Identificar al menos 2 escenarios de cambio (ej. incluir pagos por móvil) y evaluar el impacto.
    2. **Ejercicio ATAM enfocado en rendimiento**:
        - Partiendo de una arquitectura de tuberías y filtros para procesamiento de vídeo en tiempo real:
            - Listar 5 atributos de calidad relevantes (latencia, throughput, escalabilidad, etc.).
            - Proponer 3 escenarios para cada atributo (ej. latencia durante picos de carga).
            - Evaluar riesgos: ¿dónde podría fallar la arquitectura actual?
            - Sugerir trade-offs (ej. reducir paralelismo en filtros menos críticos para priorizar latencia).

Posibles ejercicios de examen

1. **Definir conceptos**
    - Explica brevemente qué es un **elemento arquitectónico** y da un ejemplo.
    - ¿Cuál es la diferencia entre **estructura estática** y **estructura dinámica**?
2. **Comparación de vistas**
    - Enumera las 4 vistas del modelo 4+1 y describe en una frase el propósito de cada una.
    - Señala dos ventajas e inconvenientes de usar múltiples vistas.
3. **Aplicación de un patrón**
    - Para un sistema de comercio electrónico, dibuja a alto nivel la arquitectura basada en **capas**: presentación, lógica de negocio, acceso a datos.
    - Indica un escenario en el que la **comunicación de abajo hacia arriba** (notificaciones) sea necesaria en este sistema.
4. **Patrones de tuberías y filtros**

- Dibuja una tubería con al menos 3 filtros para procesar un flujo de texto (normalización de caracteres, tokenización, análisis sintáctico).
- Explica un escenario de **tubería push** y otro de **tubería pull** en ese contexto.

5. **Arquitectura Pizarra**
   - Describe los tres componentes principales del patrón pizarra.
   - Da un ejemplo concreto (no de voz) donde la arquitectura pizarra sea adecuada (p. ej., diagnóstico médico).

6. **Validación de la arquitectura**
   - Diferencia entre **SAAM** y **ATAM**: objetivo principal de cada uno.
   - Dado un escenario de alta concurrencia en una aplicación bancaria, redacta un escenario ATAM que evalúe la escalabilidad.

Ejércicios adicionales para repaso rápido

1. **Relacionar conceptos**
   - Une con flechas:
     - Vista lógica → diagrama de clases
     - Vista física → diagrama de despliegue
     - Vista de procesos → diagrama de secuencia
     - Escenarios → diagrama de casos de uso

2. **Ventajas/Inconvenientes**
   - Escribe dos ventajas e inconvenientes de cada patrón arquitectónico (Capas, Tubos y Filtros, Pizarra).

3. **Breves definiciones**
   1. Stakeholder
   2. Perspectiva
   3. Módulo/filtro/hypótesis (explicar en un contexto concreto)
   4. Control en pizarra

4. **Preguntas de verdadero/falso**
   1. “En un sistema en capas, una capa sólo puede comunicarse con la capa inmediata inferior.” (V/F)
   2. “Un filtro en el patrón Pipes & Filters siempre es pasivo.” (V/F)
   3. “La pizarra central debe contener únicamente la solución final, no soluciones parciales.” (V/F)
   4. “SAAM evalúa principalmente requisitos no funcionales.” (V/F)

Recomendaciones para el estudio rápido

- Repasar **definiciones clave** (stakeholder, vista, patrón, perspectiva).

- Usar **diagramas**: dibujar cada patrón con sus componentes.
- Practicar con los **ejercicios propuestos**, intentando responder en minutos.
- Para cada patrón, tener muy claros:
    1. Componentes y relaciones.
    2. Ventajas principales.
    3. Inconvenientes más relevantes.
- En la parte de **validación**, memorizar secuencia de pasos de SAAM y ATAM.

¡Éxito en tu estudio!