Trabajo Práctico M1 - Parte 1

IS2: Arquitectura

1. Objetivo

Comprender y diferenciar los conceptos fundamentales de arquitectura de software, incluyendo estilos, patrones, vistas y aspectos transversales, y analizar cómo se relacionan con los atributos de calidad.

2. Fecha de entrega:

semana del 08/09/2025 al 15/09/2025

3. Directivas

3.1. Indagar y debatir el significado de los siguientes términos y sus diferencias:

- Arquitectura de Software vs. Diseño de Software.
  La arquitectura de software define la estructura global del sistema, estableciendo los componentes principales, sus relaciones y las decisiones tecnológicas necesarias para asegurar robustez, escalabilidad y mantenibilidad. Su objetivo es *traducir los requerimientos en un marco comprensible para los desarrolladores*.
  El diseño de software, en cambio, se centra en *detallar cómo se implementarán estos componentes a nivel más específico*, definiendo algoritmos, estructuras de datos, interfaces y flujos de interacción para cumplir de manera eficiente con la funcionalidad requerida.

- Estilos arquitectónicos.

Estilo	Descripción	Ventajas	Desventajas	Ejemplos típicos
Monolítico	Toda la aplicación está **empaquetada** y desplegada como un **único bloque**	**Simplicidad** (solo sistemas pequeños)	**Dificultad** para **escalar**, **mantenimiento** complejo, despliegue completo cambios	Aplicaciones **antiguas** de escritorio, sistemas heredados.
Cliente-Servidor	**Divide funciones** entre clientes que consumen y servidores que proveen servicios	**Centralización** de datos, facilidad de **gestión**	Punto único de **falla** en el **servidor**	Aplicaciones **web** básicas, bases de datos **centralizadas**
Tuberías y Filtros	Procesamiento **secuencial** en **etapas** (filtros) conectadas por tuberías	**Modularidad**, **flexibilidad**, buen manejo de grandes **volúmenes** de **datos**	**Latencia** en procesamiento en lotes, **menos adecuado** para **tiempo real**	Procesamiento de datos en lotes, compiladores
Centrada en Datos	La base de datos **central** es el **núcleo**; las **aplicaciones** giran **alrededor** de ella	**Consistencia** de datos, **seguridad centralizada**, **fácil**	Punto **único** de **falla**, **difícil escalado** de la BD	ERP, CRM, sistemas de **gestión empresarial**

		integración		
Peer-to-Peer	Cada **nodo** actúa como **cliente** y **servidor simultáneamente**	**Escalabilidad**, **resiliencia**, **compartición** de recursos	**Complejidad** de **coordinación**, **seguridad** más difícil	Redes de **intercambio** de archivos, blockchain
Microkernel	**Núcleo mínimo** con funcionalidades **básicas**; **extensiones** mediante **módulos** o plugins	**Extensible**, **adaptable**, fácil de mantener	Posible impacto en **rendimiento**, **complejidad** inicial	Sistemas **operativos modernos**, aplicaciones con **plugins**
Orientada a Servicios (SOA)	**Sistema** modular **basado** en **servicios** que se **comunican** a través de un **bus** común (**ESB**)	**Reutilización**, **integración** de sistemas heterogéneos, **interoperabilidad**	**Complejidad** en la **implementación**, sobrecarga de **comunicación**	**Integración** de aplicaciones **empresariales**, sistemas **distribuidos**

- Patrones arquitectónicos.

Patrón	**Descripción**	**Ventajas**	**Desventajas**	**Ejemplos típicos**
Capas	**Organización** del software en **capas** (presentación, servicios, negocio, datos)	**Claridad** en **responsabilidades**, facilita **mantenimiento** y **pruebas**	Puede generar **sobrecarga** en **comunicación** entre **capas**	**Aplicaciones empresariales**, frameworks como .NET, Java EE
Dirigida por eventos	Componentes **desacoplados** que reaccionan a **eventos**	**Escalabilidad**, **agilidad**, **bajo acoplamiento**	**Complejidad** en **depuración**, **riesgo** de **eventos** no **gestionados**	**IoT**, sistemas de mensajería, Node.js
Microservicios	**Servicios pequeños**, **independientes**, con su propia lógica y datos	**Escalabilidad**, despliegue **independiente**, **resiliencia**	Mayor **complejidad** de **orquestación**, **monitoreo** y **seguridad**	Netflix, Amazon, Kubernetes + Docker
Serverless	La **nube** gestiona la **infraestructura**; el dev solo escribe **funciones**	**Bajo costo** operativo, **escalado automático**, **rápido** desarrollo	**Dependencia** del **proveedor**, **latencia** por arranque en **frío**	AWS Lambda, Azure Functions, Google Cloud Functions
Broker	**Intermediario** central que **coordina comunicación** entre componentes	**Centralización** de **gestión**, **integración** de **sistemas** heterogéneos	Puede convertirse en **cuello** de **botella** o punto de falla	Apache Kafka, RabbitMQ, ActiveMQ
Pizarra (Blackboard)	**Repositorio común** donde agentes **especializados** aportan **conocimiento**	Adecuado para **problemas complejos** con **múltiples fuentes** de datos	**Complejidad** de **implementación**, puede ser **menos eficiente**	Sistemas de **IA**, **reconocimiento** de **voz**, **diagnósticos complejos**
MVC	**Divide** en **Modelo** (datos), **Vista** (UI) y **Controlador** (lógica)	**Separación** de **responsabilidades**, facilita **reuso** y **pruebas**	Puede ser **excesivo** para apps simples, **curva** de **aprendizaje**	Django, Ruby on Rails, Spring MVC

- Vistas de arquitectura (Modelos: 4+1, Siemens, SEI, C4).

Modelo	Vistas / Niveles	Enfoque principal	Dirigido a	Ventajas
4+1 (Kruchten)	- Lógica - Procesos - Desarrollo/Despliegue - Física - Escenarios (casos uso)	Funcionalidad, organización, topología y comportamiento del sistema	Usuarios finales, programadores, ingenieros de sistemas, integradores	Estructura integral, conecta vistas con escenarios, facilita comunicación
Cuatro Vistas de Siemens	- Conceptual - Módulos - Código - Ejecución	Relación software-hardware, del concepto a la ejecución	Arquitectos, desarrolladores y equipos de hardware	Visión completa de alto a bajo nivel, incluye retroalimentación constante
Viewtypes del SEI	- Módulos - Componentes y conectores - Asignación	Organización del software (estático y dinámico) y su despliegue	Diseñadores de software, equipos de desarrollo, administradores de despliegue	Flexible y detallado, abarca diseño, ejecución y despliegue
C4 (Simon Brown)	- Contexto - Contenedores - Componentes - Código	Trazabilidad clara desde contexto hasta detalle de código	Clientes, desarrolladores, arquitectos	Simple, comprensible, mantiene coherencia entre niveles, buena comunicación con interesados no técnicos

- Aspectos transversales (seguridad, gestión operativa, comunicación).

Área	Aspecto Transversal	Descripción	Objetivo Principal
Seguridad	Autenticación	Acceso al sistema mediante contraseñas, tokens, biometría	Verificar identidad de los usuarios
Seguridad	Autorización	Define qué acciones puede realizar cada usuario según sus permisos	Controlar el acceso a funciones específicas
Seguridad	Auditoría	Registro y trazabilidad de acciones y eventos	Detectar anomalías y actividades sospechosas
Seguridad	Gestión de Roles y Perfiles	Administración centralizada de permisos según roles	Simplificar gestión de usuarios y seguridad
Gestión Operativa	Gestión de Excepciones	Manejo eficiente de errores y fallos	Mantener la integridad del sistema y asegurar recuperación
Gestión Operativa	Instrumentación	Monitoreo y medición de desempeño (tiempos de respuesta, uso de recursos)	Optimizar calidad y anticipar problemas
Comunicación	Interoperabilidad	Integración entre sistemas y componentes usando estándares de comunicación	Garantizar intercambio de datos y servicios
Comunicación	Mailing y Notificaciones	Envío de alertas y notificaciones a los usuarios	Mantener informados a los usuarios sobre eventos relevantes.

3.2. Analizar, responder y debatir las siguientes preguntas:

1. ¿Por qué se dice que los atributos de calidad son los que definen la arquitectura de software?

   La arquitectura de software no se define solo por lo que hace el sistema, sino por cómo lo hace bajo ciertas restricciones de calidad.
   Los atributos de calidad restringen y orientan las decisiones arquitectónicas: qué componentes usar, cómo se comunican, dónde se despliegan, qué tecnologías adoptar. Dos sistemas con las mismas funcionalidades pueden tener arquitecturas totalmente distintas si sus requisitos de calidad difieren.

2. ¿Cómo se relacionan los escenarios de calidad, las tácticas de diseño y los patrones de arquitectura?

   Podemos decir que conforman la ruta desde el requisito hasta la estructura del sistema.
   Un escenario de calidad (qué necesitamos lograr) exige el empleo de ciertas tácticas (estrategias simples para lograr un atributo), y luego esas tácticas se implementan con más facilidad dentro de un patrón arquitectónico adecuado.

3. ¿Qué diferencias existen entre estilos arquitectónicos y patrones arquitectónicos? ¿Y entre patrones arquitectónicos y patrones de diseño?

   Un estilo arquitectónico tiene un alcance más abstracto y general, define una filosofía o forma de organizar sistemas, mientras que el patrón es más concreto y aplicable, proporcionando una manera de completar la implementación del estilo elegido. Se pueden usar uno o más patrones arquitectónicos en un estilo.

4. ¿Qué ventajas aporta documentar una arquitectura usando un modelo de vistas? ¿Qué diferencias encuentran entre el modelo 4+1 y el modelo C4?

   Las vistas facilitan la comprensión y comunicación de la arquitectura de una solución. Una de las ventajas de una documentación basada en vistas es que posee un enfoque por audiencia donde cada vista se adapta a un stakeholder, además evita saturar con detalles irrelevantes ya que cada vista muestra solo lo necesario, facilitando,también, la evolución y la mantenibilidad de la documentación al cambiar partes del sistema.

   El modelo 4+1, organiza la arquitectura en cuatro vistas técnicas: la vista lógica (qué componentes hay y su estructura), la vista de proceso (cómo se maneja la concurrencia y comunicación entre procesos), la vista de desarrollo (cómo se organizan los módulos en el código y repositorios) y la vista física (cómo se despliega en hardware o servidores). A estas cuatro se suma una quinta, basada en escenarios o casos de uso, que sirve para vincular y validar las demás. Es un modelo formal, riguroso, pensado para entornos académicos o corporativos tradicionales, y suele usar diagramas UML. Sin embargo, puede volverse complejo o rígido en sistemas modernos, distribuidos o ágiles.

El modelo C4, es un enfoque moderno, pragmático y visual, centrado en la comunicación clara y accesible. Se basa en cuatro niveles de abstracción que funcionan como "zooms": el nivel de Contexto (muestra el sistema dentro de su entorno, con usuarios y sistemas externos), el nivel de Contenedores (aplicaciones, servicios, bases de datos, APIs), el nivel de Componentes (módulos o piezas dentro de cada contenedor) y el nivel de Código (detalles de clases o funciones, opcional y rara vez incluido). Usa diagramas simples, tipo "cajas y líneas", fáciles de entender, y permite escalar naturalmente desde lo general hasta lo específico.
La principal diferencia entre ambos es el enfoque, 4+1 separa perspectivas técnicas, mientras que C4 separa niveles de detalle. 4+1 es más formal y exhaustivo; C4 es más intuitivo y comunicativo.

5. ¿Por qué los aspectos transversales (ej. autenticación, interoperabilidad) son críticos para la robustez de un sistema?

   Son críticos para la robustez de un sistema porque afectan a múltiples componentes y capas simultáneamente, y su mal manejo puede comprometer el funcionamiento global, incluso si la lógica de negocio es perfecta.

   No son funcionalidades centrales del dominio, pero sostienen la calidad, estabilidad y confiabilidad del sistema.
   Además, al ser transversales, no pueden encapsularse en un solo módulo: deben aplicarse de forma coherente en toda la arquitectura.

   Por eso, su diseño debe ser intencional y arquitectónico: mediante patrones (como interceptores, middlewares, AOP), políticas centralizadas y estándares.
   Garantizando que el sistema no solo "hace lo que debe", sino que lo hace de forma segura, estable, observable y mantenible , es decir robusta.

3.3. Estudiar y comprender los siguientes ejemplos:

- Estilo arquitectónico Cliente-Servidor
- Patrón arquitectónico Capas.
- Patrón arquitectónico Microservicios.
- Patrón de diseño Modelo–Vista–Controlador (MVC).
- Vista de arquitectura Modelo 4+1.

Concepto	Características	Ventajas y situaciones de uso
Estilo Cliente-Servidor	Es un estilo basado en la separación de la aplicación para el proveedor del servicio (servidor) y el solicitante (cliente). El proveedor es un servidor que	Escalabilidad y Centralización. Se usa típicamente para servicios web,

	brinda una serie de servicios o recursos que el cliente desea consumir. Es considerada una arquitectura distribuida ya que el servidor y el cliente se encuentran (usualmente) distribuidos en diferentes equipos y se comunican por medio de una red.	
Patrón en Capas	Es un patrón basado en la organización de software en capas, donde las superiores, de mayor abstracción y cercanía al usuario, dependen de las de abajo, más cercanas al hardware.	Facilita la modularidad y reutilización del código. Se utiliza en sistemas que necesiten modularidad.
Patrón microservicios	Hace referencia a una arquitectura basada en la división de servicios ofrecidos en secciones pequeñas, cohesivas e independientes.	Se utiliza en plataformas de alto tráfico como Netflix, Amazon, etc
Patrón MVC	El patrón Modelo-Vista-Controlador está basado en la distribución de responsabilidades entre tres partes. La Vista es la interfaz entre usuario y sistema, el Modelo es donde reside la interpretación conceptual del dominio, y el Controlador en el que gestiona las acciones del usuario.	Es útil para favorecer la reutilización de código, así como el trabajo en paralelo. Frameworks como Django y Spring MVC
Vista 4+1	Es un modelo de documentación de arquitectura basado en cinco vistas: ● **Vista lógica**: dirigida a usuarios finales, enfocada a funcionalidad ● **Vista de despliegue**: orientada a programadores, centrada en gestión del software ● **Vista física**: destinada a los ingenieros de sistemas, basada en la topología ● **Vista de procesos**: centrada en el rendimiento y escalabilidad del sistema. ● **Vista de escenarios**: conecta el resto de las vistas, basada en diagramas de casos de uso.	Esto es útil para cuando se necesita documentación compleja para diferentes actores que necesiten entender el sistema desde diferentes perspectivas. Se usa en sistemas grandes y complejos, como sistemas críticos

4. Entregables

Cada grupo deberá entregar un documento en PDF que incluya:

1. Definiciones de los términos solicitados en el punto 3.1, con sus diferencias.
2. Respuestas justificadas a cada una de las preguntas del punto 3.2.

3. Breve descripción de los ejemplos estudiados en el punto 3.3, destacando sus características principales y cuándo son más apropiados.
4. Conclusión grupal: ¿qué concepto les resultó más desafiante y por que?