UNER Facultad de Ingeniería

Carrera: Licenciatura en Ciencia de Datos

Cátedra: Ingeniería de Software II

Título: Trabajo Práctico 1 - Parte 1: Arquitectura

Docentes: Valotto, Victor; Godoy, Cielo

Alumnos: Carrozzo, Felipe; Ré, Lautaro Caupolicán

Año: 2025

Desarrollo

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

- Arquitectura de Software vs. Diseño de Software: el concepto de *diseñar software* es mucho más abarcativo, porque engloba diferentes niveles de detalle, entre los que se encuentra el *diseño de la arquitectura de software*. En este se identifican los componentes principales que constituyen la solución desde una visión de alto nivel.
  Cuando hablamos de *arquitectura de software*, el punto principal se encuentra en cómo se va a manejar el sistema a nivel organizativo (lógica de negocios, estructura, stack de tecnología a utilizar, etc.). Es donde se definen los escenarios que buscan cubrir los requerimientos del proyecto.

- Estilos arquitectónicos: es uno de los primeros puntos a cubrir cuando se comienza a planificar un sistema. En esta etapa, se definirá el diseño, junto con los componentes y conectores que conformarán al mismo.
  Tener el problema y los requerimientos bien definidos, son un comienzo fuerte para dar inicio al debate de cuál arquitectura es la que mejor se adecúa a nuestra problemática.

- Patrones arquitectónicos: son una solución general y probada a un problema recurrente en el diseño de software a nivel de arquitectura. Se presenta como una guía o plantilla que describe cómo estructurar un sistema, teniendo en cuenta el contexto y las restricciones del proyecto.

- Vistas de arquitectura (Modelos: 4+1, Siemens, SEI, C4): son representaciones de la solución que ayudan a comprender cómo se estructura el sistema. Estas vistas pueden modelar diferentes partes de la solución. Cada una de ellas tienen enfoques distintos: desde la visión lógica y de escenarios, pasando por módulos y componentes, hasta el despliegue en infraestructura y el nivel de código.

- Aspectos transversales (seguridad, gestión operativa, comunicación): los aspectos transversales son fundamentales para el funcionamiento de un sistema robusto y seguro. Estos atraviesan todo el sistema, sin limitarse a un único módulo o componente. Son transversales porque atraviesan varias capas y partes de la solución, influyendo en cómo se diseñan, implementan y mantienen los distintos elementos.
  Por ejemplo, si queremos controlar a qué áreas pueden acceder ciertos usuarios, lo hacemos mediante la *autorización*, una vez que los usuarios están *autenticados*, en base al rol que su perfil tenga, se definen qué acciones puede realizar.

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?

   Los atributos de calidad definen la arquitectura de software porque condicionan y guían su desarrollo, asegurando que el sistema no solo haga qué debe hacer, sino que lo haga cómo se necesita. Por ejemplo, un software que cumple las funciones pero no satisface atributos como rendimiento o confiabilidad probablemente no será útil en la práctica.

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

   Los *escenarios de calidad* describen situaciones concretas, donde se ponen a prueba los atributos de calidad. Por ejemplo, ¿qué pasa si aumenta la carga de usuarios?

   Las *tácticas de diseño* ofrecen cómo lograr una solución a un nivel más concreto. Son decisiones específicas que buscan cumplir los escenarios. Por ejemplo, usar caché para mejorar el rendimiento.

   Los *patrones de arquitectura* son soluciones generales que integran múltiples tácticas en estructuras probadas (ej: patrón en capas, cliente-servidor). En conjunto, forman un flujo: escenario → táctica → patrón.

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

   Los *estilos arquitectónicos*, son modelos abstractos que definen el vocabulario de los componentes y conectores del sistema, estableciendo reglas de relación. Algunos ejemplos pueden ser arquitectura en capas, cliente-servidor. Mientras que los *patrones arquitectónicos* son soluciones concretas que se reutilizan, para dar solución a problemas específicos de arquitectura. Estos indican el camino para cumplir requisitos y atributos de calidad.

   Los *patrones arquitectónicos* actúan a nivel global del sistema, afectando la estructura del sistema e interacción de los módulos. En cambio los *patrones de diseño* se enfocan en resolver problemas de implementación y relacionan las partes internas.

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

   Documentar una arquitectura permite mostrar diferentes perspectivas del sistema, enfocándose en lo que es más relevante para cada tipo de interesado (desarrolladores, arquitectos, clientes, equipo de operaciones). También ayuda a gestionar la complejidad, ya que cada vista aborda un aspecto particular (estructura, comportamiento, despliegue, etc.).

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

   Los aspectos transversales son fundamentales porque impactan a todo el sistema, no solo a un módulo aislado. La *autenticación*, por ejemplo, afecta a la seguridad global, y la *interoperabilidad* determina la capacidad de integración con otros sistemas. Si no se diseñan adecuadamente desde la arquitectura, pueden comprometer atributos esenciales como seguridad, disponibilidad, escalabilidad o mantenibilidad, debilitando la robustez del sistema completo.

3. Estudiar y comprender los siguientes ejemplos.

- **Estilo arquitectónico Cliente-Servidor:** Es un estilo general de organización en el que el sistema se divide en dos roles principales:
  - *Cliente*: solicita servicios (interfaz de usuario, aplicación ligera).
  - *Servidor*: provee servicios (procesamiento, almacenamiento, lógica de negocio).
  Ejemplo: aplicaciones web con navegador (cliente) y servidor web.

- **Patrón arquitectónico Capas:** Tiene una dependencia de las capas más cercanas al usuario a las capas de infraestructura (recursos de hardware). Promueve una organización del software donde la capa de más arriba (dependiente) es la que está más cerca al usuario. La capa de presentación es la responsable de unir al usuario final con la solución.

- **Patrón arquitectónico Microservicios:** Organiza el sistema como un conjunto de servicios independientes, cada uno con una funcionalidad específica, que se comunican entre sí mediante APIs ligeras (generalmente REST o mensajes).

- **Patrón de diseño Modelo–Vista–Controlador (MVC):** Está vinculado a la capa de presentación, que interconecta tres partes: la *vista* es la interfaz entre el usuario y el sistema, presenta información e interpreta acciones del usuario. El *modelo* representa la información de la aplicación y es en donde reside la interpretación del dominio. El *controlador* gestiona los comportamientos o acciones del usuario, manipulando el modelo para actualizar la información.

- **Vista de arquitectura Modelo 4+1:** El *modelo de vistas 4+1 de Kruchten* propone documentar la arquitectura de software desde cinco perspectivas complementarias, con el fin de cubrir las necesidades de todos los interesados en el sistema. La *vista lógica* describe la funcionalidad principal y cómo se organiza mediante clases y objetos. La *vista de procesos* se enfoca en el comportamiento dinámico y la concurrencia. La *vista de desarrollo* refleja la organización interna del software en módulos o paquetes. La *vista física* describe el despliegue en la infraestructura de hardware. Finalmente, la *vista de escenarios* valida la arquitectura a través de casos de uso, integrando todas las demás vistas y asegurando que la solución cumpla con los requisitos funcionales y no funcionales.

Conclusiones

Dentro de los conceptos trabajados, los que nos resultaron más interesantes fueron el patrón de diseño *Modelo–Vista–Controlador* (MVC) y el *modelo C4*. MVC, aunque ampliamente usado en frameworks modernos, exige comprender con claridad la separación de responsabilidades y cómo las interacciones entre vista, modelo y controlador impactan en la experiencia del usuario y en la mantenibilidad del sistema. Por otro lado, el modelo C4 nos presentó mayor complejidad porque requiere visualizar la arquitectura en distintos niveles de abstracción —desde el contexto general hasta el detalle del código—, asegurando coherencia y trazabilidad entre ellos. En conjunto, ambos conceptos muestran la importancia de pensar en distintas escalas del diseño: desde la organización del código hasta la representación global de la arquitectura.