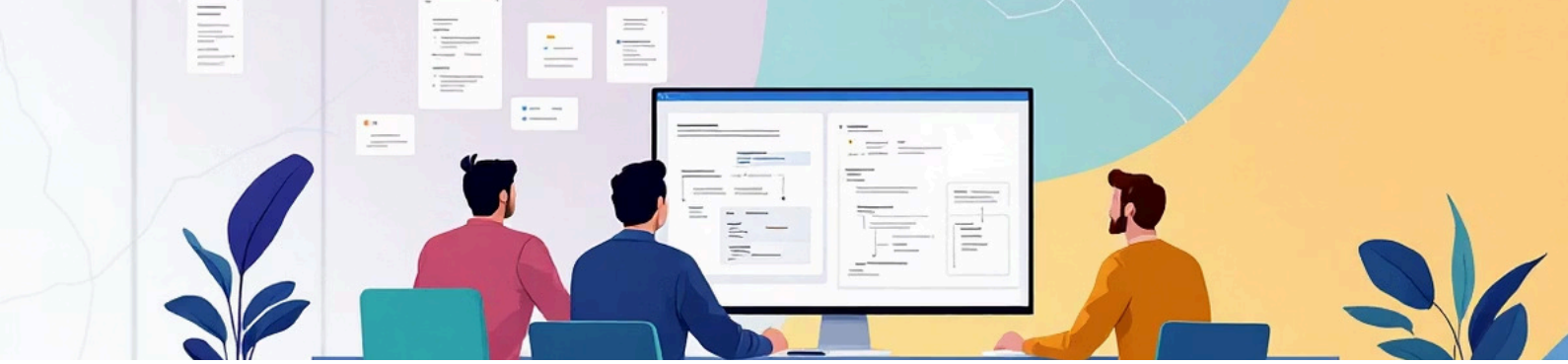
PROMETEO

Unidad 7:

Modelado UML – Diagramas de comportamiento

Entornos de desarrollo

Técnico Superior de DAM / DAW



Sesión 24: Casos de uso, secuencias, actividad y estados

Cómo entender el comportamiento del sistema antes de programar

En diseño de software no basta con saber qué clases tendrá un sistema; también debes comprender cómo se comporta, cómo responden sus objetos ante distintos estímulos, cómo interactúan los usuarios, y qué caminos lógicos sigue cada proceso. Aquí es donde entran los diagramas de comportamiento UML.

Mientras que el diagrama de clases te muestra la estructura estática del sistema, los diagramas de comportamiento explican su funcionamiento dinámico: cómo fluye la información, cómo se comunican los elementos y cómo evoluciona el estado de un objeto a lo largo del tiempo. Esta visión es imprescindible si quieres reducir errores lógicos y crear software con un comportamiento coherente y predecible.

Casos de uso

Son el punto de partida para entender qué espera el usuario del sistema. Representan interacciones entre:

- Actores (personas, organizaciones o sistemas externos).
- El sistema, que responde a esas acciones.

Los casos de uso te permiten tener una visión funcional sin entrar en detalles técnicos. Son esenciales en reuniones con clientes, product owners y perfiles no técnicos.

Diagramas de secuencia

Explican cómo se lleva a cabo un caso de uso a nivel de comunicación entre objetos. Representan:

- el orden temporal de los mensajes,
- las "lifelines" de los objetos,
- las llamadas a métodos y sus respuestas.

Son especialmente valiosos para diseñar APIs, microservicios o procesos que implican comunicación entre varios módulos.

Diagramas de actividad

Visualizan el flujo de tareas de un proceso: decisiones, bifurcaciones, paralelismos y posibles caminos del sistema. Funcionan como un diagrama de flujo centrado en las acciones. Te ayudan a detectar condiciones mal definidas, duplicidades de tareas y puntos de decisión críticos.

Diagramas de estado

Modelan cómo un objeto concreto cambia a lo largo del tiempo. Representan:

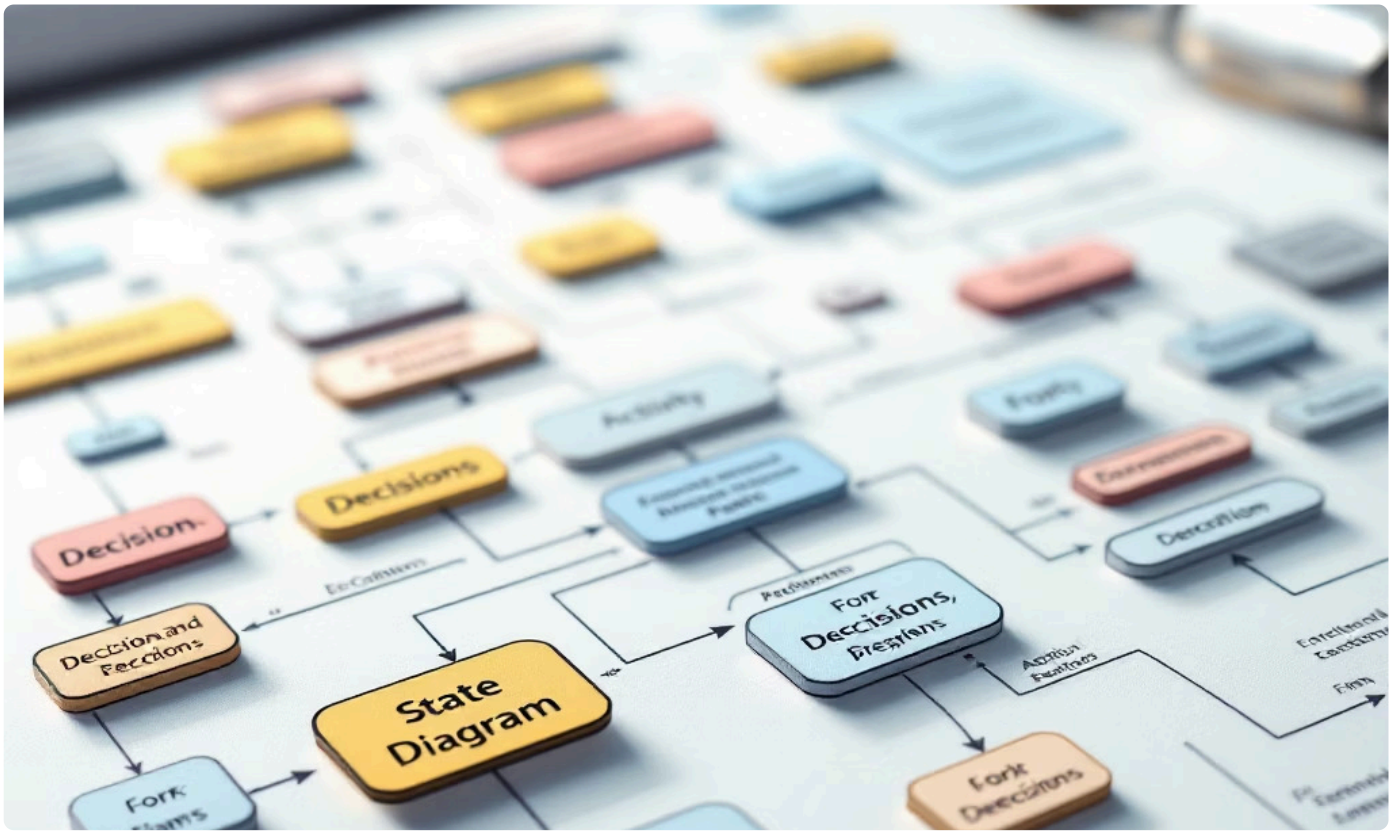
- estados posibles,
- transiciones,
- eventos que provocan el cambio.

Son extremadamente útiles para objetos que siguen un ciclo de vida, como pedidos, reservas o tickets de soporte.

¿Por qué son tan importantes estos diagramas?

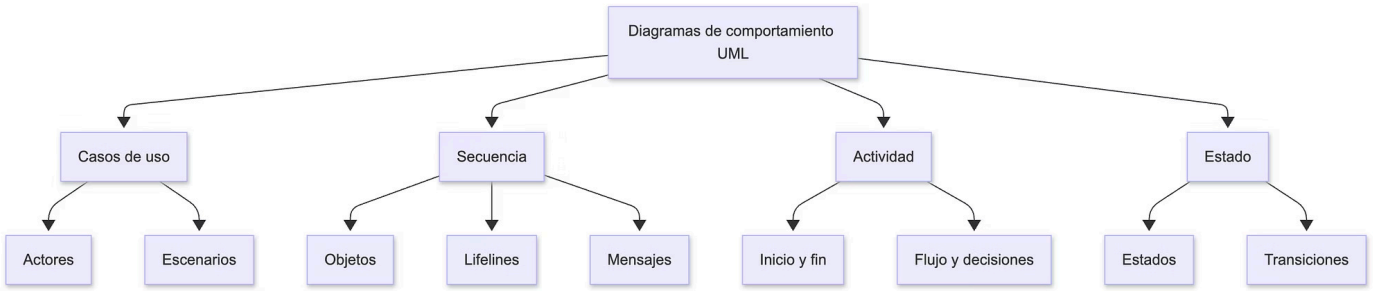
- Detectan errores antes de programar.
- Permiten validar la lógica con el cliente.
- Alinean la visión entre analistas y desarrolladores.
- Facilitan la creación de pruebas.
- Reducen ambigüedades y malentendidos en la implementación.

En un proyecto real, estos cuatro diagramas trabajan juntos: el caso de uso define el objetivo, la secuencia muestra cómo se ejecuta, la actividad detalla el flujo lógico y el estado describe la evolución del objeto implicado.



Esquema Visual: Mapa conceptual de los diagramas de comportamiento

Aquí tienes un diagrama en Mermaid que resume los cuatro tipos de diagramas:



Explicación para recrearlo correctamente:

Nodo central

El nodo central es **Diagramas de comportamiento UML**.

Cuatro ramas principales

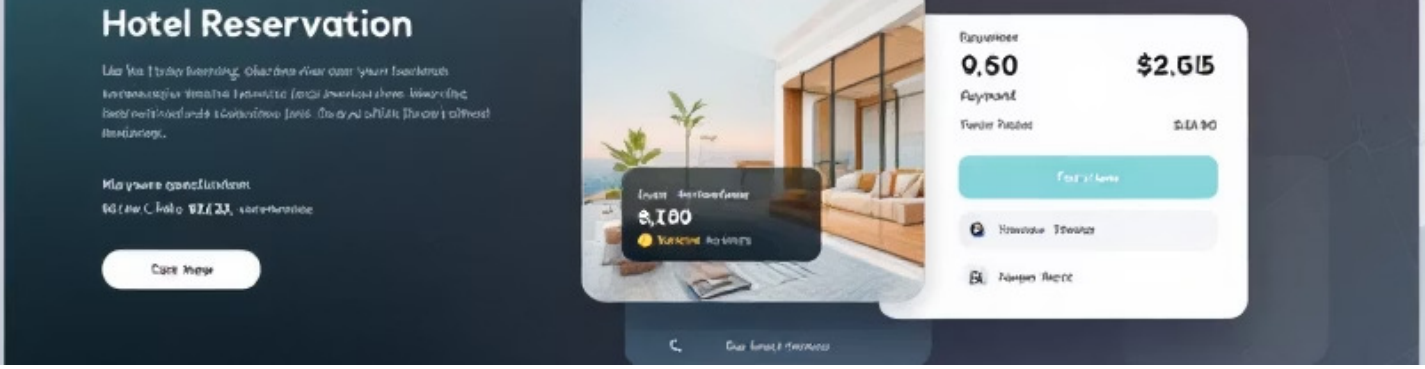
De él salen cuatro ramas: **Casos de uso**, **Secuencia**, **Actividad**, **Estado**.

Elementos clave

Cada rama se divide en los elementos clave necesarios para construir cada tipo de diagrama:

- Actores y escenarios (casos de uso)
- Objetos, lifelines y mensajes (secuencia)
- Punto de inicio, punto de fin, decisiones (actividad)
- Estados y transiciones (estado)

Con esta estructura puedes reconstruir un mapa conceptual completo del funcionamiento dinámico del sistema.



Caso de Estudio: Plataforma de reservas de hotel

Contexto

Una empresa tecnológica desarrolla una plataforma web para reservar habitaciones de hotel. El equipo quiere validar toda la lógica del sistema antes de comenzar con la programación del backend, así que utiliza los diagramas de comportamiento para analizar cada funcionalidad clave.

Estrategia

Se crean cuatro diagramas complementarios para cubrir todas las perspectivas.

Diagrama de caso de uso

Actor principal: Cliente.

Casos de uso:

- Buscar habitación
- Reservar habitación
- Realizar pago
- Cancelar reserva

Este diagrama permitió detectar que se necesitaba un actor adicional: **Pasarela de pago**, indispensable para procesar transacciones.

Diagrama de secuencia

Se modeló la interacción entre:

- Cliente
- Sistema de reservas
- API de disponibilidad
- Pasarela de pago

Flujo representado:

1. El cliente solicita disponibilidad.
2. El sistema consulta la API del hotel.
3. La API devuelve resultados.
4. El cliente selecciona una habitación.
5. El sistema envía datos a la pasarela de pago.
6. La pasarela confirma o rechaza el pago.
7. El sistema crea la reserva o muestra un error.

Este diagrama permitió identificar un problema: no se contemplaba enviar un email de confirmación. Gracias al diagrama, el equipo añadió un nuevo servicio: **Notificaciones**.

Diagrama de actividad

Se modeló el flujo de pago:

Inicio → Introducir datos

Validar tarjeta

¿Pago aceptado?

Sí → Confirmación

No → Solicitar otro método de pago

Al visualizar este flujo, se detectó un camino no previsto: ¿qué ocurre si se cae la pasarela? El diagrama ayudó a incorporar un estado intermedio "Pago en revisión".

Diagrama de estado

El ciclo de vida del objeto **Reserva** quedó así:

pendiente → confirmada → cancelada

Pero el análisis reveló más estados necesarios:

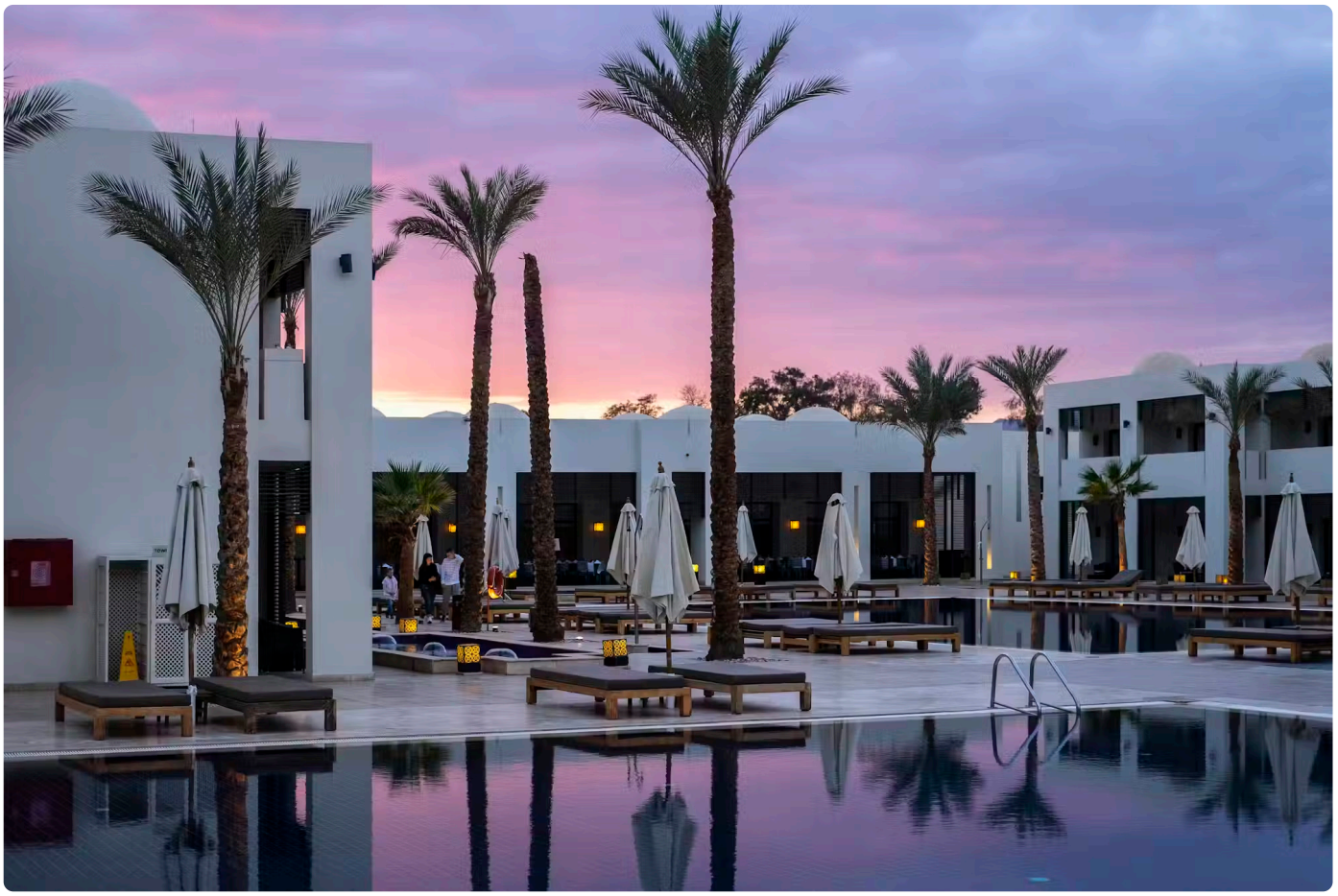
- pago_en_revisión
- caducada si no se paga en un tiempo límite

Resultado

El uso conjunto de los cuatro diagramas permitió:

- Refinar requisitos antes de programar
- Evitar errores de flujo lógico
- Detectar escenarios omitidos
- Alinear a negocio, diseño y desarrollo
- Definir claramente responsabilidades entre sistemas

Este caso demuestra que los diagramas de comportamiento no son solo documentación, sino herramientas de diseño críticas.



Herramientas y Consejos

Aquí tienes recomendaciones prácticas para aplicar estos diagramas en tu trabajo diario:

Empieza siempre por casos de uso

Son la base del análisis con el cliente. Si el caso de uso está mal definido, todos los demás diagramas también lo estarán.

Usa diagramas de secuencia para definir APIs

Cada mensaje puede representarse como una llamada a un endpoint. Una buena práctica es reflejar:

- peticiones,
- respuestas,
- errores,
- tiempos de espera.

Los diagramas de actividad son perfectos para decisiones complejas

Úsalos cuando el proceso incluya:

- validaciones,
- condiciones,
- bucles,
- caminos alternativos.

Los diagramas de estado ayudan en sistemas basados en eventos

Ideales para objetos como:

- reservas,
- pedidos,
- incidencias,
- tickets de soporte.

Herramientas recomendadas

- **StarUML:** muy usado en entornos profesionales.
- **Visual Paradigm:** excelente para equipos grandes y documentación.
- **Lucidchart:** intuitivo y colaborativo.
- **PlantUML:** combina código + diagramas. Ideal para programadores.

Usa nombres descriptivos

Evita abreviaturas y nombres ambiguos. Por ejemplo, "validarPago()" es mejor que "vp()".

Mantén trazabilidad entre los diagramas

Cada caso de uso debe tener su secuencia, actividad y estados implicados. Esto crea coherencia y facilita las pruebas.

Mitos y Realidades

❌ Mito 1: Los diagramas de comportamiento son solo para analistas.

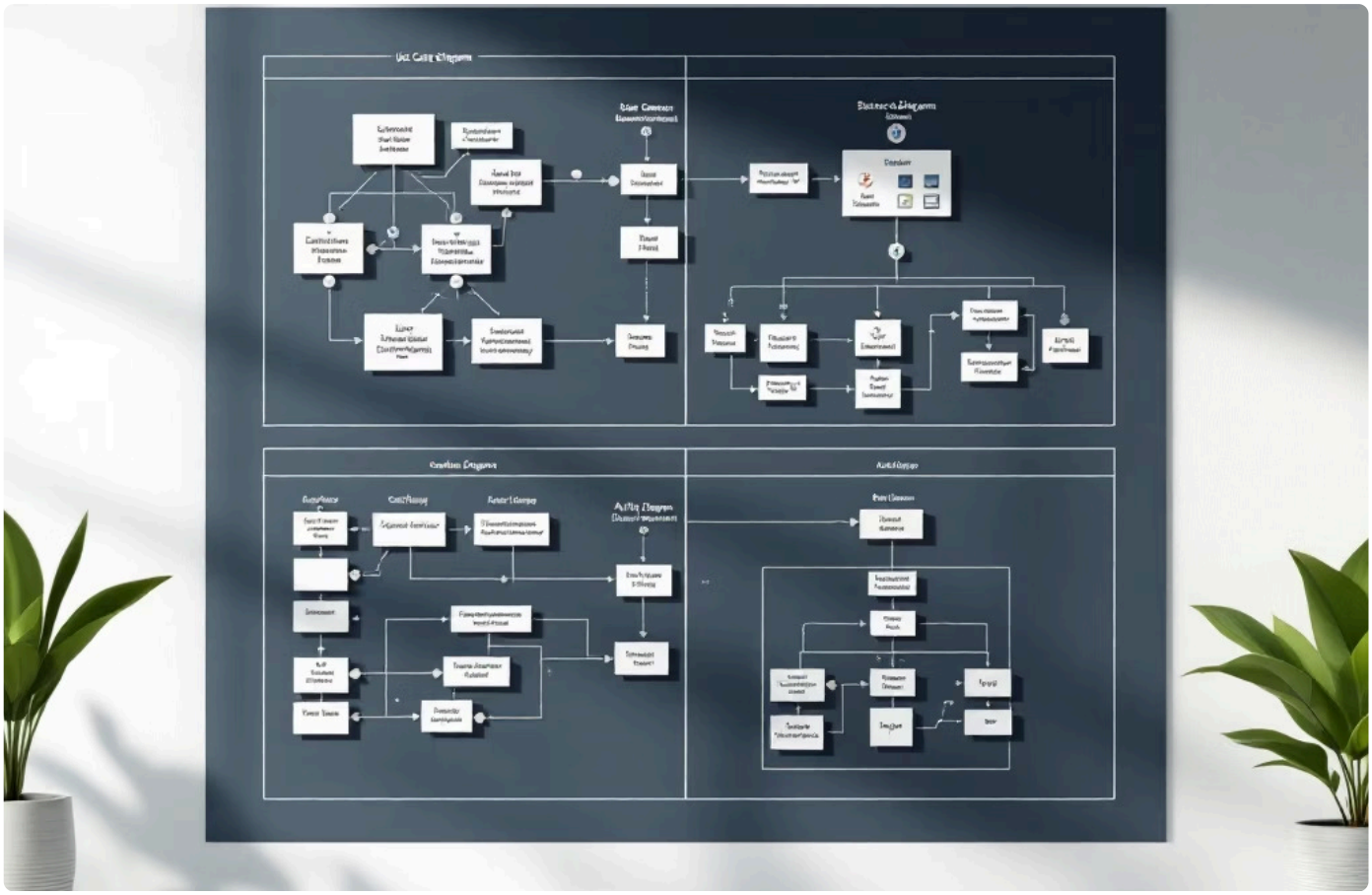
➡️ **FALSO.** Los desarrolladores los usan para diseñar APIs, los testers para crear casos de prueba y los product owners para validar requisitos.

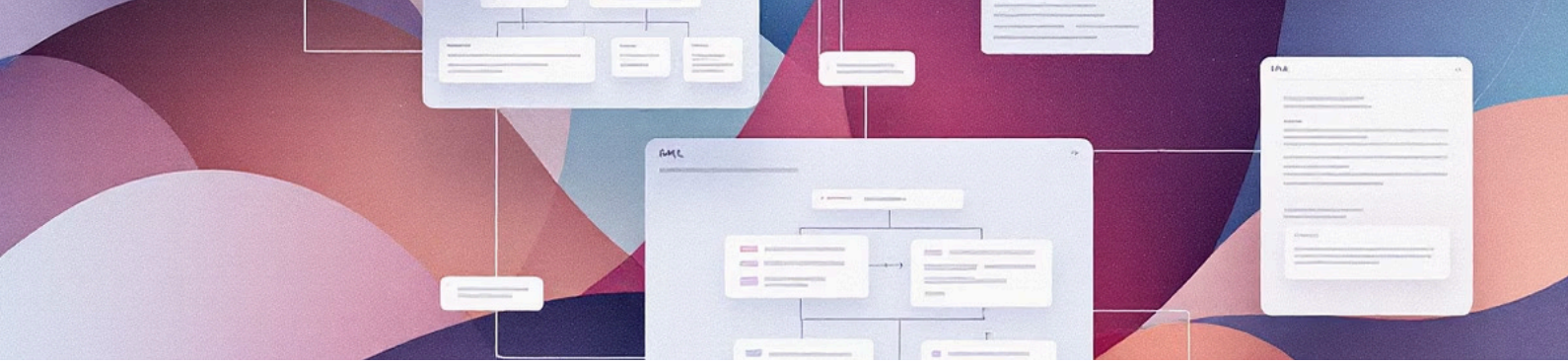
❌ Mito 2: Los diagramas de secuencia son solo para procesos simples.

➡️ **FALSO.** Son críticos en microservicios y sistemas distribuidos, donde necesitas visualizar todas las llamadas entre servicios.

Resumen Final

- Los diagramas de comportamiento muestran cómo actúa el sistema.
- **Casos de uso:** interacción usuario-sistema.
- **Secuencia:** orden temporal de mensajes entre objetos.
- **Actividad:** flujo de tareas y decisiones.
- **Estado:** transición entre estados de un objeto.





Sesión 25: Ejercicios integrales y casos de estudio UML

Integrar todas las vistas UML para diseñar un sistema completo

Hasta ahora has trabajado los diferentes tipos de diagramas UML por separado: clases, casos de uso, secuencia, actividad y estados. Cada uno aporta una perspectiva distinta del sistema, pero el valor real del modelado UML surge cuando se integran todos para construir una visión completa y coherente de un proyecto. Esta sesión te enseña a combinar esas piezas como lo haría un analista o un arquitecto de software profesional.

Un modelo UML integral funciona como un mapa multifacético del sistema:

- El diagrama de clases define la estructura y las entidades.
- Los casos de uso explican qué espera el usuario y qué debe hacer el sistema.
- Los diagramas de secuencia detallan cómo se comunican los objetos para ejecutar un caso de uso.
- El diagrama de actividad muestra la lógica del proceso paso a paso.
- El diagrama de estados describe cómo evoluciona un objeto clave en el tiempo.

Este enfoque es habitual en proyectos reales porque permite validar la arquitectura antes de programar. Cuando un equipo trabaja en un sistema complejo —por ejemplo, una plataforma de pedidos, una aplicación bancaria o un servicio de reservas— necesita asegurarse de que los requisitos, la lógica y la implementación están alineados.

Un modelo UML unificado ayuda a:

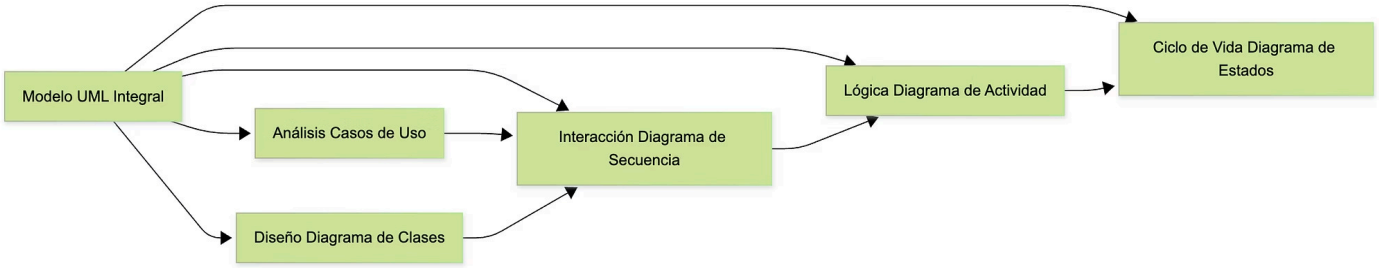
- Detectar inconsistencias entre distintas vistas del sistema.
- Mantener coherencia entre lo que se pide y lo que se desarrolla.
- Facilitar la comunicación entre perfiles técnicos y no técnicos.
- Servir como base para documentar, generar código o planificar pruebas.
- Reducir el riesgo de errores lógicos o estructurales antes de escribir código.

En entornos ágiles, UML no se usa como una documentación rígida, sino como un lenguaje visual para aclarar conceptos y acelerar la toma de decisiones. En proyectos grandes, se convierte en parte esencial del proceso de análisis y diseño.

En esta sesión integras todo lo aprendido y construyes una visión completa de un sistema real. Aprenderás a identificar qué diagrama necesitas en cada fase y cómo conectarlos para que el modelo final sea sólido y útil.

Esquema Visual: Integración de los diagramas UML

El siguiente esquema conceptual en Mermaid resume cómo se relacionan todas las piezas del modelado:



Explicación del esquema:

01

Nodo central

Modelo UML Integral es el nodo central.

02

Fases del desarrollo

Cada tipo de diagrama representa una fase del desarrollo: análisis, diseño, interacción, lógica y ciclo de vida.

03

Flujo de alimentación

Las flechas muestran cómo un diagrama alimenta al siguiente:

- Los casos de uso determinan qué secuencias desarrollar.
- El diagrama de clases proporciona los objetos que intervienen en las secuencias.
- Las secuencias revelan acciones que se convierten en actividades.
- El diagrama de actividad destapa los estados y transiciones del sistema.

Este flujo refleja un proceso de modelado completo y profesional.

Caso de Estudio: Sistema de pedidos online (modelo UML completo)

Contexto

Una empresa de ecommerce necesita diseñar un sistema de gestión de pedidos online que permita al usuario registrarse, añadir productos al carrito, realizar una compra y seguir el estado del pedido. El objetivo es construir un modelo UML integral que sirva como guía de desarrollo y como documentación para los distintos equipos.

Estrategia UML: integración de vistas

- Diagrama de Casos de Uso

  Actores:

  - Cliente
  - Carrito
  - Pasarela de pago

  Casos de uso principales:

  - Registrarse
  - Iniciar sesión
  - Añadir producto al carrito
  - Realizar pedido
  - Consultar pedidos
  - Procesar pago

  El análisis reveló la necesidad de gestionar estados del pedido, por lo que se añadió el caso de uso "Actualizar estado".

- Diagrama de Clases

  Clases principales:

  - **Cliente:** id, nombre, email, contraseña
  - **Producto:** idProducto, nombre, precio
  - **Pedido:** idPedido, fecha, estado, total
  - **DetallePedido:** cantidad, subtotal
  - **Pago:** idPago, tipo, estadoPago

  Relaciones:

  - Pedido 1...n DetallePedido
  - Cliente 1...n Pedido
  - Pedido 1...1 Pago

  Este diagrama mostró duplicidad de datos en "Pedido" y "Pago", lo cual se corrigió separando la lógica de pago en una clase específica.

- Diagrama de Actividad del proceso de pago

  Acciones clave:

  - Validar usuario
  - Revisar carrito
  - Enviar datos de pago
  - ¿Pago aprobado?
  - Sí → Generar pedido
  - No → Solicitar otro método

  Se detectó un problema lógico: ¿qué ocurre si el pago queda en revisión? Se añadió un flujo adicional.

- Diagrama de Secuencia del proceso de compra

  Objetos: Cliente → Sistema → Inventario → PasarelaPago → Pedido

  Flujo secuencial:

  1. Cliente selecciona productos.
  2. Sistema valida inventario.
  3. Cliente confirma compra.
  4. Sistema envía datos a PasarelaPago.
  5. Pasarela devuelve aprobación o rechazo.
  6. Sistema crea pedido o devuelve error.
  7. Pedido notifica al cliente.

  Este diagrama permitió detectar un caso no contemplado: reintento de pago. Se añadió como camino alternativo.

- Diagrama de Estados del "Pedido"

  Estados identificados:

  - pendiente
  - procesado
  - pagado
  - enviado
  - entregado
  - cancelado

  Transiciones disparadas por eventos como "Pago aprobado", "Pago fallido", "Envío realizado".

Resultados

El modelo integral permitió:

- Detectar escenarios omitidos (reintento de pago, pago en revisión).
- Ajustar clases para evitar redundancias.
- Clarificar responsabilidades entre los módulos del sistema.
- Facilitar las pruebas unitarias gracias al diagrama de estados.
- Crear una documentación sólida para el equipo de desarrollo.

Este caso refleja cómo los diferentes diagramas UML se complementan para crear un sistema entendible, coherente y estructurado antes de escribir el código.

Herramientas y Consejos

Diferencia entre capas visualmente

Usa colores o estereotipos UML (por ejemplo <<Controller>>, <<Service>>) para distinguir:

- clases de dominio,
- controladores,
- servicios,
- repositorios.

Esto facilita revisar el modelo de un vistazo.

Documenta cada diagrama

Incluye una breve descripción del flujo: qué representa, qué actores intervienen y qué decisiones clave contiene.

Herramientas recomendadas

- **Lucidchart:** ideal para trabajar en equipo en tiempo real.
- **Visual Paradigm:** muy completo y profesional; permite ingeniería directa e inversa.
- **PlantUML:** perfecto para desarrolladores porque se escribe como código.
- **StarUML:** muy usado en entornos formativos y profesionales.

Conecta siempre los diagramas

No sirven como piezas aisladas. Asegúrate de que:

- Cada caso de uso tiene su secuencia.
- Cada actividad se refleja en la lógica del sistema.
- Cada estado corresponde a un atributo o transición real.

Usa UML como herramienta de comunicación

En reuniones con clientes, empezar con casos de uso evita confusiones. En reuniones técnicas, los diagramas de secuencia y actividad permiten resolver dudas más rápido.

Actualiza el modelo

Un diagrama que no se mantiene actualizado deja de ser útil. Sincroniza UML ↔️ código periódicamente.

Mitos y Realidades

❌ Mito 1: "UML es demasiado teórico para proyectos reales."

➡️ **FALSO.** Empresas de software, consultoras tecnológicas y equipos de arquitectura lo utilizan para validar la lógica antes de programar y para evitar errores costosos. UML es práctico cuando se usa de forma ligera y orientada a objetivos.

❌ Mito 2: "Crear muchos diagramas ralentiza el desarrollo."

➡️ **FALSO.** Lo que ralentiza un proyecto son los errores de diseño descubiertos tarde. UML acelera el desarrollo porque anticipa inconsistencias y reduce retrabajo.

Resumen Final

- Un modelo UML integral reúne todas las vistas del sistema.
- **Casos de uso** → análisis funcional.
- **Clases** → estructura del sistema.
- **Secuencia** → interacción entre objetos.
- **Actividad** → lógica de procesos.
- **Estado** → ciclo de vida de objetos clave.
- **Herramientas:** Visual Paradigm, StarUML, PlantUML.