**INGENIERÍA DE SOFTWARE 2 - EXPLICADO PARA ENTENDER**

**1. ¿QUÉ ES UML Y PARA QUÉ SIRVE?**

UML significa "Lenguaje Unificado de Modelado". Imaginate que tenés que explicarle a alguien cómo va a funcionar un sistema, por ejemplo, una aplicación para pedir comida. En vez de escribir mil páginas, usás dibujos (diagramas) que muestran las partes importantes del sistema y cómo se relacionan.

UML NO es una forma de trabajar (una metodología), sino una forma de **representar visualmente** lo que vas a hacer o lo que está hecho.

Sirve para:

* Comunicar ideas entre personas que están diseñando o desarrollando software.
* Mostrar qué hace el sistema y cómo se comporta.
* Poder generar parte del código automáticamente.

Es estandarizado por un grupo llamado OMG (Object Management Group), lo que significa que está bien definido y todos lo entienden igual.

Hay 3 tipos de diagramas:

* **Estructurales**: muestran los "bloques" del sistema, lo que tiene y cómo está armado.
* **De comportamiento**: muestran cómo cambia el sistema con el tiempo.
* **De interacción**: muestran cómo se mandan mensajes entre objetos o partes del sistema.

**2. DIAGRAMA DE CASOS DE USO**

Imaginá que estás diseñando un cajero automático. Lo primero que tenés que pensar es: ¿qué cosas puede hacer un usuario? Eso es lo que muestra un diagrama de casos de uso.

Este diagrama NO muestra cómo está hecho el sistema por dentro, sino **qué puede hacer el usuario** con él. Es una vista general de las funciones importantes.

Componentes:

* **Actor**: es quien usa el sistema (una persona, otro sistema). Se dibuja como un muñeco stickman.
* **Caso de uso**: es algo que el sistema ofrece al actor. Ejemplo: "Retirar dinero". Se dibuja como un óvalo.
* **Sistema**: un rectángulo que encierra los casos de uso.
* **Relaciones**: líneas que conectan actores y casos de uso. Pueden ser especiales:
  + <<include>>: indica que un caso siempre incluye a otro.
  + <<extend>>: indica que un caso puede agregar otro opcionalmente.
  + Herencia entre actores (uno es una versión más específica de otro).

Es muy útil para hablar con clientes o usuarios sin meterse en detalles técnicos.

**3. DIAGRAMA DE CLASES**

Este diagrama muestra **las clases que vas a usar en el sistema y cómo se relacionan entre ellas**. Una clase es como un plano para crear objetos.

Por ejemplo, una clase "Persona" podría tener:

* Atributos: nombre, edad.
* Métodos (funciones): hablar(), caminar().

El diagrama muestra:

* **Clases**: se dibujan como rectángulos divididos en 3 partes: nombre, atributos, operaciones.
* **Visibilidad**:
  + + es público (todos lo pueden usar)
  + - es privado (solo la clase lo usa)
  + # es protegido (solo la clase y sus hijas lo usan)
  + ~ es de paquete (solo para clases dentro del mismo paquete)

Relaciones comunes:

* **Asociación**: una clase "tiene" otra.
* **Agregación**: una clase contiene otra, pero pueden vivir separadas.
* **Composición**: una clase contiene otra, y si la primera se borra, la otra también.
* **Generalización**: una clase hereda de otra.
* **Dependencia**: una clase necesita a otra para hacer algo, pero no está fuertemente conectada.

**4. DIAGRAMA DE ESTADOS**

Este diagrama sirve para mostrar **cómo cambia el estado de un objeto** según lo que le pasa.

Por ejemplo: un pedido puede estar "Pendiente" → "Pagado" → "Enviado" → "Entregado".

Muestra:

* Los **estados posibles**.
* Las **transiciones** (cambios de estado), causadas por eventos.
* El **estado inicial y final**.

Se usa para entender qué puede pasarle a algo con el tiempo.

**5. DIAGRAMA DE SECUENCIA**

Este diagrama muestra **cómo se mandan mensajes los objetos con el tiempo**. Es como un chat organizado.

Muestra:

* Los objetos, cada uno con una "línea de vida" vertical.
* Mensajes entre ellos: llamados, respuestas, mensajes asíncronos (que no esperan respuesta), etc.
* El orden de los mensajes, de arriba hacia abajo (cronológico).

Es útil para ver paso a paso cómo se hace algo, como registrar un usuario.

**6. DIAGRAMA DE ACTIVIDADES**

Muestra **el flujo de trabajo de un proceso**, como si fuera una receta paso a paso.

Incluye:

* Punto de inicio
* Actividades (acciones)
* Decisiones (rombos con condiciones)
* Tareas en paralelo (bifurcaciones)
* Punto final

Por ejemplo: "Crear cuenta" → "Verificar email" → ¿Es válido? → Sí: "Activar cuenta" → Fin

**7. DIAGRAMA DE COMUNICACIÓN**

Es parecido al de secuencia, pero no muestra el tiempo, sino **el orden de los mensajes y las conexiones** entre objetos.

Cada mensaje tiene un número para saber cuál va primero. Es útil para ver cómo se comunican objetos sin importar cuándo lo hacen.

**8. RUP (PROCESO RACIONAL UNIFICADO)**

El Proceso Racional Unificado, conocido como RUP, es una forma organizada y flexible de **planear, construir y entregar software**. No es una receta fija, sino una guía que se adapta a lo que necesita el equipo o el proyecto. Se usa especialmente en proyectos grandes o en equipos con muchos desarrolladores.

**¿Para qué sirve?**

Sirve para dividir el trabajo en fases, y para asegurarse de que cada parte del sistema esté bien pensada, diseñada, construida, probada y entregada al cliente.

**Fases del RUP:**

1. **Inicio**: se define qué se quiere hacer y si realmente vale la pena hacerlo. Acá se entienden los requisitos básicos y se hace un primer plan.
2. **Elaboración**: se reduce la incertidumbre y se crea la arquitectura principal. Se elige cómo se va a construir el sistema.
3. **Construcción**: se construyen las partes del sistema. No todo de golpe, sino por partes funcionales.
4. **Transición**: se entrega el sistema al cliente, se dan capacitaciones y se hacen los ajustes necesarios.

**Características importantes:**

* **Iterativo**: se repiten ciclos (iteraciones) donde se mejoran las cosas poco a poco.
* **Incremental**: se agregan funciones nuevas paso a paso.
* **Orientado a casos de uso**: se diseña pensando en lo que el usuario quiere hacer.
* **Orientado a la arquitectura**: se construye una base sólida del sistema al inicio.

**Principios de RUP:**

* Adaptarse al contexto del cliente.
* Equilibrar las prioridades de todos los involucrados.
* Trabajar en equipo con buena comunicación.
* Mostrar resultados valiosos en cada iteración.
* Reutilizar conceptos para simplificar.
* Asegurar la calidad desde el inicio.

**9. METODOLOGÍAS ÁGILES**

Las metodologías ágiles son formas modernas y flexibles de trabajar en proyectos de software. Cambian la idea tradicional de seguir un plan fijo. Acá se busca adaptarse al cambio, mejorar continuamente y centrarse en el cliente.

**¿Cuál es la idea central?**

Que los equipos trabajen juntos, entreguen resultados rápido, escuchen al cliente, y mejoren cada vez que se repite el proceso.

**Valores del Manifiesto Ágil:**

* Las personas y su comunicación son más importantes que los procesos y herramientas.
* Un software que funciona es mejor que montones de documentos.
* La colaboración con el cliente es mejor que negociar un contrato.
* Adaptarse al cambio es mejor que seguir un plan cerrado.

**12 Principios del Ágil:**

1. El cliente siempre es lo más importante.
2. Hay que estar abiertos al cambio.
3. Se entregan prototipos funcionales frecuentemente.
4. Todos los miembros del equipo deben colaborar.
5. La motivación del equipo es clave.
6. Se debe comunicar claramente todo el tiempo.
7. La funcionalidad del producto mide el progreso.
8. Se debe dar soporte y mantener lo entregado.
9. Buscar excelencia técnica.
10. Mantener las cosas simples.
11. Coordinar tareas y personas.
12. Aprender en cada ciclo de trabajo.

**Etapas comunes (no son fijas, pero ayudan):**

* Análisis de requisitos
* Planificación
* Reunión del equipo
* Desarrollo del producto
* Análisis, documentación y entrega

**10. METODOLOGÍAS ÁGILES POPULARES**

**SCRUM**

Es como una agenda organizada para trabajar en equipo. Divide el trabajo en **Sprints** (ciclos cortos, como de 2 semanas).

En cada Sprint:

* Se decide qué se va a hacer.
* Se trabaja en eso.
* Se revisa cómo salió.
* Se mejora para el siguiente Sprint.

**Roles en Scrum:**

* **Scrum Master**: ayuda al equipo a trabajar bien, resuelve problemas.
* **Product Owner**: elige qué es lo más importante para hacer primero.
* **Equipo de desarrollo**: hace el trabajo.

**Herramientas:**

* **Backlog**: lista de tareas ordenadas por prioridad.
* **Daily Scrum**: reunión diaria para ver avances.
* **Sprint Review**: al final, se muestra lo hecho.
* **Sprint Retrospective**: se habla de cómo mejorar.

**KANBAN**

Es un método visual. Usa un **tablero con columnas** (como: Por hacer, En proceso, Hecho). Cada tarea es una tarjeta que se mueve de una columna a otra.

Principios:

* Empezar desde lo que ya se hace.
* Mejorar de a poco.
* Respetar cómo trabaja el equipo.
* Fomentar liderazgo en todos.

Prácticas:

* Visualizar el trabajo.
* Limitar cuántas cosas se hacen al mismo tiempo.
* Medir el flujo.
* Tener reglas claras.
* Recibir comentarios frecuentes.
* Mejorar todo el tiempo.

**XP (Extreme Programming)**

Está más enfocado en la **calidad del código y la programación**. Es muy rápido y se adapta fácilmente.

Valores:

* Simplicidad
* Comunicación
* Retroalimentación
* Valentía
* Respeto

Prácticas importantes:

* Trabajar en parejas.
* Hacer pruebas automáticas desde el inicio.
* Integrar todo constantemente.
* Diseñar lo más simple posible.
* Compartir el código (cualquiera puede mejorar cualquier parte).

Todo esto es clave para entender cómo se trabaja en proyectos reales de software hoy en día. No importa cuál se use, lo importante es adaptarse a lo que el equipo y el cliente necesiten.