INTRODUCCIÓN AL ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

Índice

1. INTRODUCCIÓN

Desde los años 80 el desarrollo de software orientado a objetos ha ido madurando como un enfoque alternativo a la programación estructurada o modular.

A mediados de los años 90 **convivían multitud de métodos de análisis y diseño orientados a objetos**, lo cual generó una situación de cierto caos y desconcierto.

Con el paso del tiempo se fue haciendo patente que las diferencias entre los distintos métodos eran cada vez menores y que las luchas entre sus partidarios y detractores sólo redundaban en la generación de más caos.

En esta situación, la Rational Software Corporation concluye que todo este ruido está perjudicando al empleo de la Orientación a Objetos y decide encargar a Rumbaugh, Booch y Jacobson la creación de un lenguaje unificado de modelado para el desarrollo orientado a objetos, lo que termina con el nacimiento del UML en 1997, cuando el OMG (Object Management Group) aprueba el borrador de la especificación del UML 1.0 que les es presentado.

Desde entonces el UML ha sufrido diversas modificaciones, y en la actualidad, la versión vigente es la 2.0. En los siguientes puntos se analizan las técnicas de Análisis y Diseño OO que ofrece UML.

2. PROGRAMACIÓN ORIENTADA A OBJETOS

Surge como un intento de resolver los problemas complejos del mundo real de una forma más simple y natural. Ya no se centra en las tareas a realizar, sino que intenta simular el escenario real dentro del programa, a través de un conjunto de **objetos** que interactúan entre sí para resolver el problema.

Los conceptos clave de la POO son los siguientes:

2.1. Objetos

Son la unidad básica de organización de la información en POO y se extraen del análisis de la descripción del problema que se pretende modelar. Suelen corresponder con nombres o sustantivos.

Se puede definir un objeto como una **entidad que combina estado** (atributos), **comportamiento** (métodos) **e identidad** (propiedad que los hace únicos).

Los objetos no son entes aislados, sino que se comunican entre sí a través de mensajes para realizar acciones.

2.2. Clases

Una **clase** es un concepto abstracto empleado en POO para definir tipos de datos. En comparación con la programación estructurada, se puede decir que las clases son los tipos de datos y los objetos las variables. Así pues, a partir de la definición de una clase (**tipo abstracto de datos**), se pueden crear (**instanciar**) un número indefinido de objetos, que compartirán las características comunes (comportamiento) definidas en la clase pero tendrán estado e identidad propios.

Las clases no existen de modo aislado; sino que se relacionan entre sí mediante relaciones de asociación, agregación, composición o herencia.

2.3. Encapsulación

Es la capacidad de reunir en un mismo objeto un conjunto de atributos y los métodos que los manipulan, permitiendo la **ocultación** de los detalles de implementación. De esta manera el objeto se comporta como una "caja negra", de la cual se puede saber qué es lo que hace, pero no cómo lo hace.

2.4. Herencia

Se puede definir como la propiedad que permite a los objetos ser construidos a partir de otros objetos. O dicho de otra manera, la capacidad de un objeto perteneciente a una clase determinada para utilizar las estructuras de datos y los métodos que contemplan sus ascendientes. Por tanto el objetivo final es la reutilización de código.

2.5. Polimorfismo

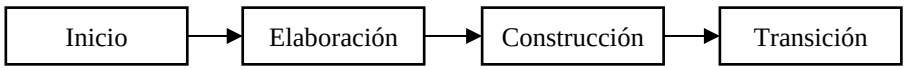
Consiste en la posibilidad de tener varios métodos con el mismo nombre en distintas clases, y por tanto, con comportamientos distintos.

Permite escribir **código genérico**, donde un mismo mensaje puede producir acciones totalmente diferentes cuando es recibido por objetos distintos.

3. METODOLOGÍA RUP (RATIONAL UNIFIED PROCESS)

Metodología creada por la Rational Software Corporation, se emplea conjuntamente con UML para el análisis, diseño, implementación y documentación de sistemas orientados a objetos.

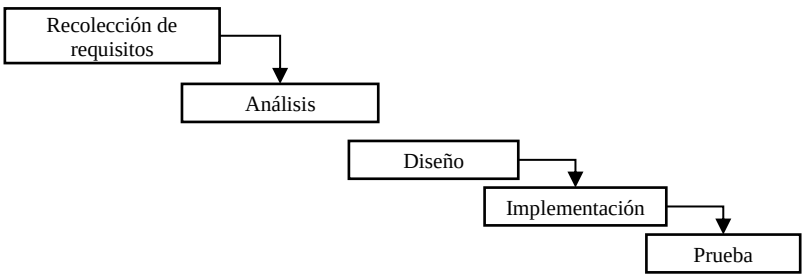
Emplea un **ciclo de vida iterativo e incremental** que mezcla el modelo en cascada y el modelo en espiral basado en cuatro **fases**:



En cada fase se refinan los objetivos de las fases anteriores y se llevan a cabo los objetivos propios de la misma:

- **Inicio**: se centra en la obtención de requisitos e identificación de casos de uso.
- **Elaboración**: se centra en el análisis y el diseño del sistema.
- **Construcción**: se centra en la implementación del producto.
- **Transición**: se centra en eliminar errores y garantizar la entrega al usuario.

En cada fase se realizan una o más **iteraciones**. En cada iteración se realiza un proceso en cascada, con las siguientes etapas:



4. LENGUAJE DE MODELADO UNIFICADO (UML)

UML comprende un conjunto de diagramas empleados para modelar, construir y documentar software orientado a objetos. Fue creado por los padres de la orientación a objetos (Rumbaugh, Booch y Jacobson) y es el estándar utilizado por la industria.

Ventajas:

- Unifica distintas notaciones previas en un solo estándar.
- Independiente de la metodología que se aplique en el desarrollo del software.

Inconvenientes:

- Falta de integración con otras técnicas (diseño de interfaces de usuario…).
- Complejo (el 80% de los problemas se resuelven con el 20% de UML).

A continuación se analizan los diagramas más relevantes.

4.1. Diagramas de casos de uso

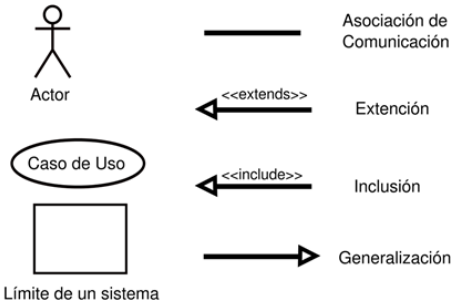
Muestran la interacción entre los usuarios (actores) y el sistema.

Su objetivo es identificar los requisitos funcionales y son la base para guiar todo el proceso de análisis y diseño.

Se modela un **caso de uso** por cada proceso que deba llevar a cabo el sistema y se documenta durante la fase de análisis indicando: el actor que inicia el caso de uso, las precondiciones y postcondiciones que debe cumplir, las acciones que conlleva, etc.

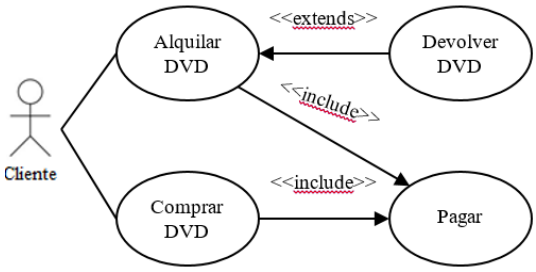
Los principales elementos que forman los diagramas de casos de uso son los siguientes:

- **El sistema**: todo lo que engloba el rectángulo.
- **Caso de uso**: cada una de las funcionalidades del sistema, se representan mediante un óvalo con un nombre identificativo. Especifican qué se hace, pero no cómo se hace y se pueden descomponer en caso de uso más detallados.
- **Actores**: usuarios del sistema, interactúan con este mediante casos de uso.



Las **relaciones** conectan casos de uso con actores o casos de uso entre sí. Existen 3 tipos de relaciones:

- **Include**: Se establece una relación de inclusión cuando un caso de uso necesita la funcionalidad de otro, es decir, lo usa.
- **Extends**: Se establece una relación de extensión cuando un caso de uso tiene un comportamiento opcional.
- **Generalización**: Se usa normalmente entre actores, para hacer que uno herede las funcionalidades del otro. Se indica con una flecha que apunta hacia el elemento padre.



4.2. Diagramas de clases

La finalidad de un **diagrama de clases** es proporcionar una perspectiva estática del sistema, para lo que representa las clases y sus relaciones.

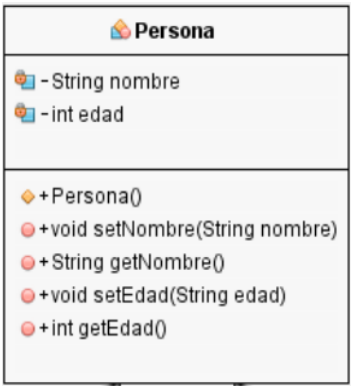
4.2.1. Clases, atributos y métodos

Una **clase** se representa mediante una caja subdividida en tres partes. En la superior se muestra el nombre de la clase, en la media los **atributos** y en la inferior los **métodos**.

El grado de detalle de la representación de una clase es variable.

Se puede representar de forma esquemática simplemente con un rectángulo con su nombre, la representación de los atributos puede hacerse indicando o no su tipo y visibilidad y la representación de los métodos puede indicar o no su visibilidad y los parámetros de entrada y salida con o sin su tipo.

También se pueden añadir notas adjuntas a la definición de una clase, contenidas en un rectángulo conectado a la clase mediante una línea de puntos discontinua.



En cuanto a la **visibilidad de métodos y atributos**, existen 3 niveles diferentes:

- **Público**: Se representa con el símbolo +, indica que puede acceder al miembro de la clase desde cualquier lugar.
- **Protegido**: Se representa con el símbolo #, indica que se puede acceder al miembro de la clase desde la propia clase o desde una clase que herede de ella.
- **Privado**: Se representa con el símbolo -, indica que se puede acceder al miembro de la clase solo desde la propia clase.

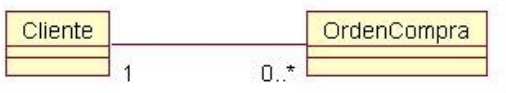
4.2.2. Relaciones: asociación, agregación, composición y herencia

Las relaciones existentes entre las clases, nos indican como se comunican los objetos de esas clases entre sí.

Asociación

Permite asociar objetos que colaboran entre sí y se representa como una línea continua entre las clases asociadas. La multiplicidad nos indica cuantos objetos de una clase se pueden asociar con un objeto de la otra clase de la relación.

Ejemplo:



Un cliente puede tener asociadas muchas Órdenes de Compra, en cambio una orden de compra solo puede tener asociado un cliente.

Agregación y composición

Ambos tipos de relación se emplean para indicar que varios componentes son parte de un compuesto.

Un rombo relleno representa una composición y uno vacío una agregación.

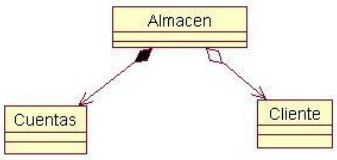
En las relaciones de agregación, los componentes pueden ser compartidos entre varios compuestos y estos no desaparecen al desaparecer el componente.

En las relaciones de composición, los componentes no pueden ser compartidos entre varios compuestos y estos desaparecen al desaparecer el componente.

Ejemplo:

Si el almacén cierra, no tiene sentido mantener las cuentas registradas en él.

En cambio, sí tiene sentido mantener a los clientes, ya que pueden relacionarse con otros almacenes.

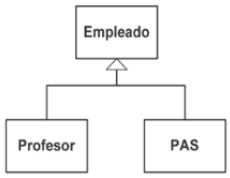


Herencia

Relación entre una clase y sus subclases.

Éstas heredan los atributos y métodos de la clase padre y poseerán además características propias.

Se indica con un triángulo sin rellenar que apunta a la clase padre.



4.3. Diagramas de objetos

Se emplean cuando se quiere mostrar el estado del sistema en un momento determinado. Es una "foto" del estado en un momento dado, de las instancias del **diagrama de clases**.

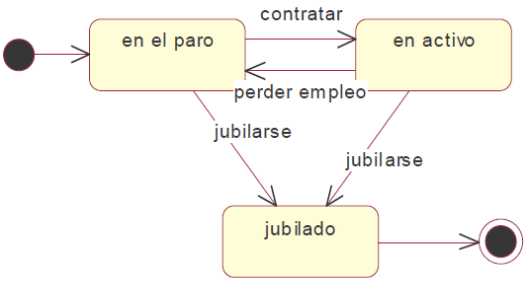
Los **objetos** se representan de la misma forma que una clase, con la diferencia de que en el compartimento superior aparece el nombre del objeto (opcional) junto con el nombre de la clase subrayados y separados de ":".

4.4. Diagramas de estados

Muestra la secuencia de estados por los que pasa un objeto a lo largo de su vida y los eventos (transiciones) que los provocan.

Un estado se representa como una caja redondeada con el nombre del estado en su interior y una transición se representa como una flecha desde el estado origen al estado destino.

El estado inicial se muestra como un círculo sólido y el estado final como un círculo sólido rodeado de otro círculo. Un objeto no puede "estar" en esos estados, pero nos sirven para saber cuáles son las transiciones inicial y final.



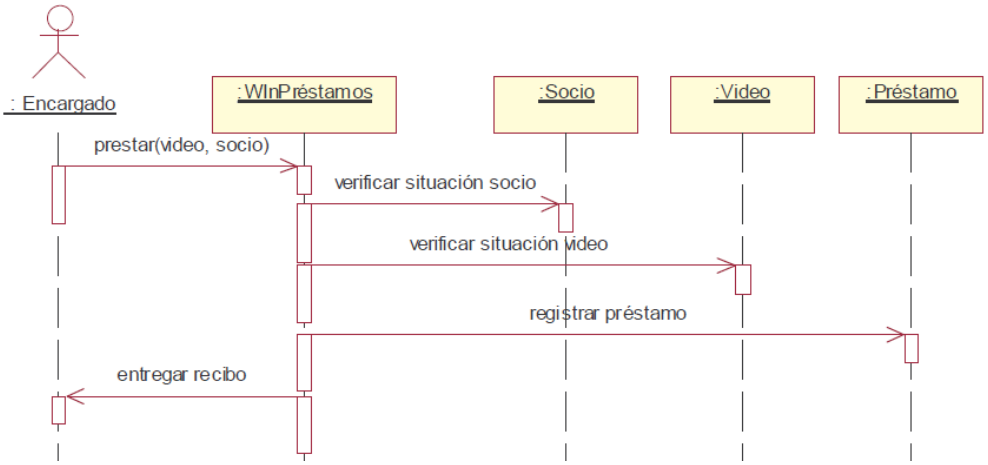
4.5. Diagramas de interacción

Describen el comportamiento dinámico del sistema en un determinado escenario, mostrando la interacción entre el conjunto de objetos que cooperan, mediante el paso de mensajes, en la realización del escenario.

Hay dos tipos de diagramas de interacción: de secuencia y de colaboración.

4.5.1. Diagrama de secuencia

Muestra los objetos participantes en una interacción y los mensajes que intercambian ordenados según su secuencia en el **tiempo**.

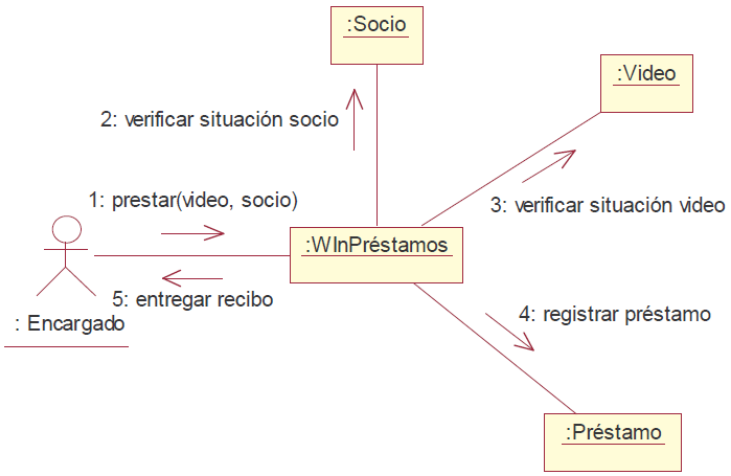


El eje vertical representa el tiempo, y el eje horizontal los objetos y actores participantes en la interacción. El tiempo transcurre hacia abajo. Cada objeto u actor tiene una línea vertical que indica su periodo de vida y los mensajes se representan mediante flechas entre los distintos objetos, fluyendo el tiempo de arriba abajo.

4.5.2. Diagrama de colaboración

Muestra la misma información que el de secuencia pero de forma diferente, ya que no se representa el tiempo, aunque sí se indica el orden de envío de los mensajes entre los objetos.

Además la colocación de los objetos es más flexible permitiendo mostrar con mayor claridad cuáles son las colaboraciones entre ellos.

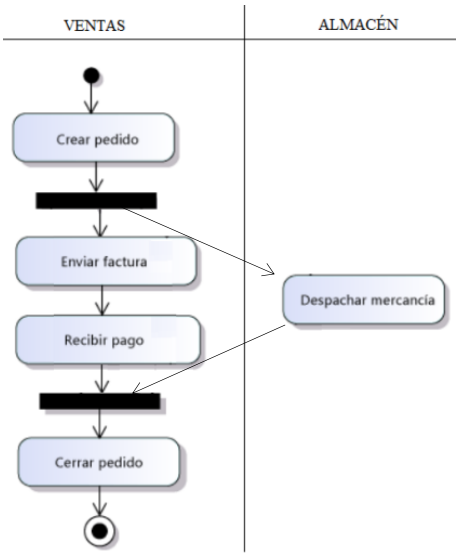


4.6. Diagrama de actividad

Muestra el orden en el que se va realizando una tarea dentro del sistema.

Es una variante de **diagrama de estado**, donde la mayoría de estados son actividades.

- **Actividad**: conjunto de acciones atómicas (ininterrumpidas) que realizan los objetos.
  Se representan mediante rectángulos redondeados.
- **Transición**: flechas que indican el flujo de actividades.
- **Flujos simultáneos**: describen acciones que se ejecutan simultáneamente (concurrentes).
  Se representan mediante barras gruesas perpendiculares a las transiciones.
- **Decisiones**: Representan alternativas en el flujo de la lógica y se representan mediante rombos.
- **Marcos de responsabilidad**: agrupan en columnas actividades relacionadas, indicando en quien recae la responsabilidad de ejecutar dicho conjunto de acciones.



5. CONCLUSIÓN

Con esta introducción se pretende que el alumno desarrolle un sentido crítico del análisis y diseño orientado a objetos, conociendo las ventajas que ofrece su aplicación, de forma que se esté capacitado para dar razones contundentes y certeras de qué métodos son necesarios a la hora de analizar y diseñar un problema mediante orientación a objetos.

6. BIBLIOGRAFÍA

Barranco, J. *Metodología del análisis estructurado de sistemas*. UPCO, 2001.

García, J. B.; Ortín, F. *Programación avanzada orientada a objetos.* Andavira, 2011.

http://www.uml.org/