# Prácticas de Programación







# Tema 1. Introducción al análisis y diseño de programas





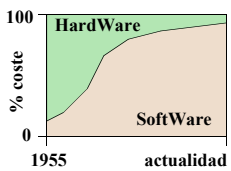


## 1.1 Ingeniería del software

**Meta: desarrollo costeable de sistemas software**

**Surge en 1968 debido a la "crisis del software"**

- **el aumento de potencia de los computadores hacía pensar que sería posible abordar problemas cada vez más complejos**
- **pero las aplicaciones se finalizaban con retraso, costando mucho más de lo presupuestado y con muchos errores**




**Principal aportación de la Ingeniería del Software:**

- **Definición de *modelos de proceso de desarrollo del software*: conjunto de actividades y resultados asociados utilizados para concebir, desarrollar, instalar y mantener un producto software**

**La Ingeniería del Software todavía es una disciplina en evolución**

# 1.2 Actividades en el desarrollo del software

**No existe un proceso de desarrollo universal**

- **depende de la naturaleza del producto y de la empresa**

**Todos los procesos incluyen las siguientes actividades:**

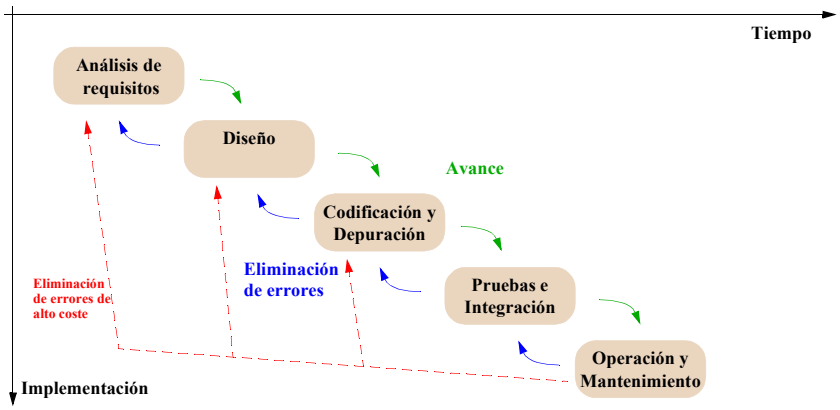
- ***Análisis de requisitos*: estudio de la naturaleza del problema y definición de *qué* debe hacer el sistema software**
- ***Diseño e implementación*: comienza con el diseño de la estructura general del software para, tras sucesivas etapas de refinamiento, alcanzar la fase de escritura del código**
- ***Pruebas e integración*: suelen ser fases cíclicas, en las que de forma incremental se van probando e integrando las diversas partes del sistema**
- ***Mantenimiento*: reparación de problemas, adaptación a nuevas condiciones, mejoras, etc.**





# 1.3 Modelo clásico: desarrollo en cascada

**Se basa en una secuencia de pasos que son optimizados uno a uno a nivel individual**

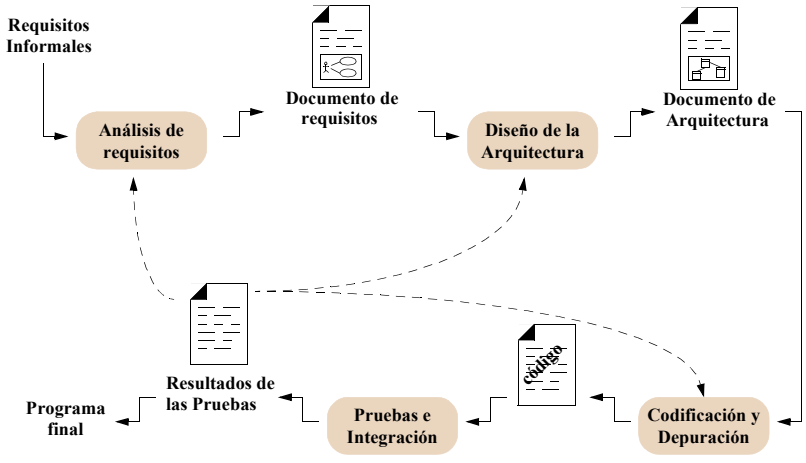







# Resultados de las etapas

**Cada etapa se basa en los resultados de la anterior**



# Limitaciones del modelo de desarrollo en cascada

**La calidad del proceso se ve limitada ya que este modelo:**

- **Exige la toma de decisiones en fases iniciales, sin conocer su efecto en fases sucesivas**
- **Considera que los requisitos del programa están definidos antes de que éste haya sido diseñado**
- **Se basa en criterios tales como la experiencia del programador, y es difícil de automatizar**
- **Hace muy difícil y costoso el proceso de mantenimiento del software**

**Sin embargo, su uso reporta muchos beneficios con respecto a un proceso desordenado, en el que no se siga una metodología clara**





# 1.4 Objetivos de los nuevos modelos

**Los nuevos modelos de desarrollo tratan de:**

- **Proporcionar a los usuarios un objeto ejecutable en las etapas iniciales del proceso para clarificar los requisitos**
- **Posibilitar que el usuario actúe como analista**
- **Minimizar los errores cometidos en las etapas iniciales del desarrollo**
- **Facilitar el mantenimiento**
- **Mantener actualizada la documentación**
- **Analizar y reducir el riesgo**

**Se pueden utilizar en combinación con el modelo clásico, o una combinación de varios de ellos**





# 1.5 Modelo de desarrollo evolutivo

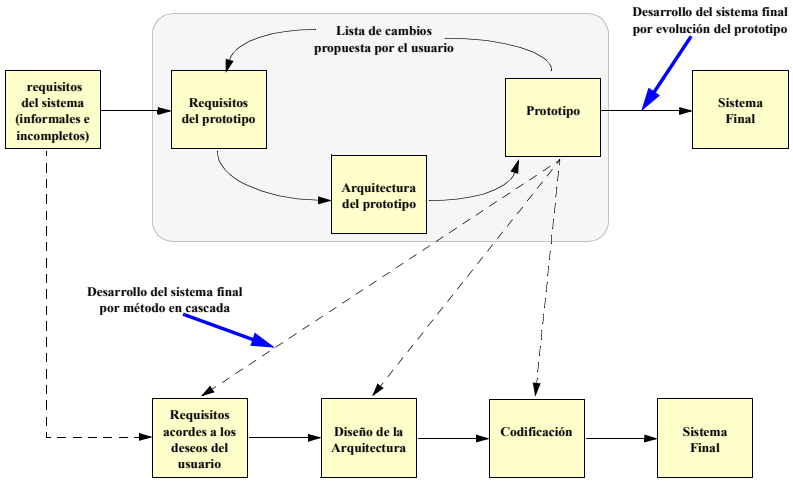
**Se basa en evolucionar un prototipo hasta lograr el sistema final**

- **se crea un prototipo inicial al comienzo del proyecto (posiblemente con funcionalidad reducida)**
- **se entrega al usuario**
  - **lo que le permite definir más claramente sus requisitos: "sabré lo que quiero cuando lo vea"**
- **el prototipo se va refinando en sucesivas versiones**

**Cuando el prototipo alcanza un nivel adecuado se puede:**

- **Transformar en el sistema completo**
  - **Se corre el peligro de que tenga una estructura deficiente**
- **Iniciar un nuevo proceso de desarrollo en cascada basándose en la experiencia adquirida**
  - **Más costoso, pero normalmente más seguro**

# Esquema del modelo de desarrollo evolutivo








# 1.6 Modelo de desarrollo en espiral

**Trata de aunar las ventajas de los modelos clásico y evolutivo**

- **realiza el desarrollo del producto a través de varias etapas**
- **añade un elemento nuevo: el análisis de riesgos**

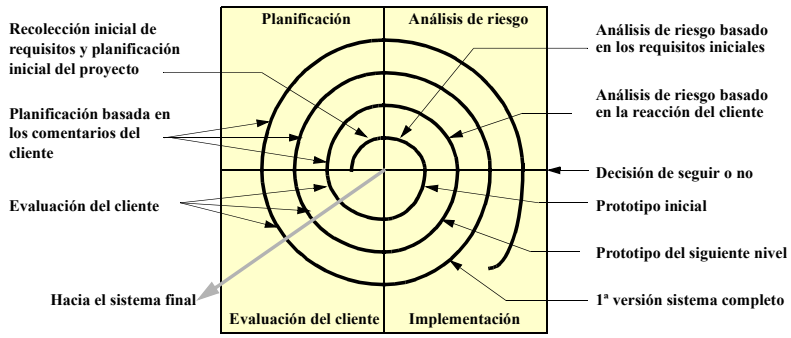
**Cada etapa se compone de las siguientes fases:**

- ***Planificación*: determinación de objetivos, alternativas, y restricciones, de acuerdo con la etapa actual. Salvo en la primera etapa, se tendrán en cuenta las valoraciones del cliente**
- ***Análisis de riesgo*: análisis de las alternativas, e identificación y valoración de riesgos. Como resultado se eligen las alternativas de menor riesgo, o si el riesgo es demasiado alto se abandona el proyecto**
- ***Implementación*: desarrollo del producto del siguiente nivel**
- ***Evaluación del cliente*: valoración de los resultados por parte del cliente**

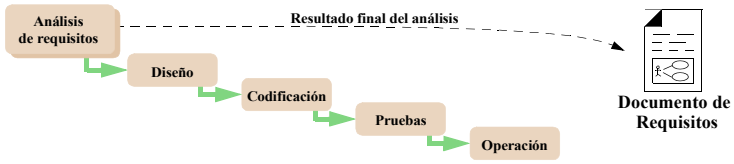




# Esquema del modelo de desarrollo en espiral



# 1.7 Introducción al análisis de requisitos




**Para construir algo lo primero que debemos saber es qué es ese "algo"**

**Análisis de requisitos:**

- **proceso de *entender y documentar* lo que queremos hacer**
- **generalmente se centra en *qué* debe hacer el sistema**
- ***no en cómo* debe hacerlo**
- ***Requisito* (requerimiento o *requirement*): necesidad o restricción que debe satisfacer un producto software para contribuir a la solución de un problema real**

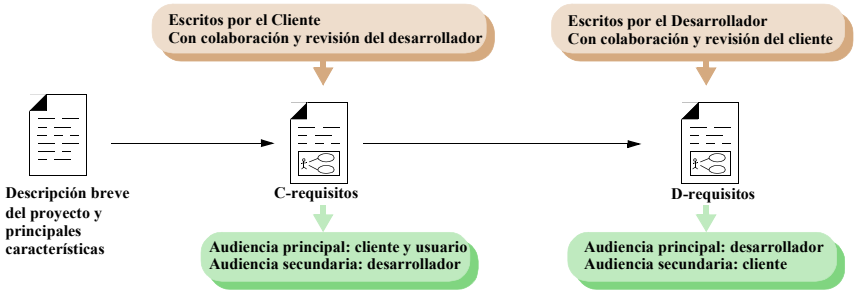




# Requisitos del cliente y del desarrollador

**Existen dos tipos de requisitos:**

- ***Requisitos del cliente* (C-requisitos): lo que el cliente quiere y necesita, expresado en un lenguaje claro para él**
- ***Requisitos del desarrollador* (D-requisitos): más detallados que los C-requisitos y expresados en un lenguaje más formal y estructurado**







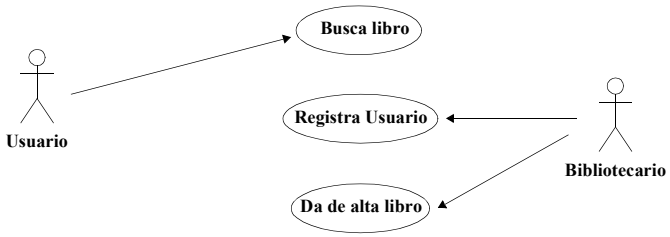

# Requisitos funcionales y no funcionales

**Tanto los C- como los D-requisitos pueden clasificarse en:**

- ***Requisitos Funcionales*: describen la funcionalidad del sistema**
  - **"El sistema de control medirá la temperatura y la humedad de la habitación"**
- ***Requisitos No Funcionales*:**
  - **Prestaciones**
    - **"La captura del vídeo debe realizarse al menos a 12 imágenes por segundo"**
  - **Fiabilidad**
    - **"No se debe experimentar más de un error fatal al mes"**
  - **Restricciones:**
    - **"Debe utilizarse el lenguaje de programación Ada2005"**

# Cómo se expresan los C-requisitos

- En *lenguaje natural*, de la forma más clara y precisa posible
  - "El sistema debe permitir al cliente acceder al saldo de su cuenta"
  - posiblemente apoyado por tablas y diagramas sencillos
- Como *casos de uso*: muestran las interacciones entre los agentes externos (actores) y la aplicación








- Con *casos de uso detallados*: describen una interacción entre un actor y la aplicación

Ejemplo: Caso de uso "Registra Usuario"

1. El bibliotecario selecciona la opción "Registrar Usuario"
2. La aplicación muestra una ventana que permite introducir el nombre y el DNI del usuario
3. El bibliotecario introduce los datos solicitados
4. La aplicación añade el nuevo usuario a la lista de usuarios registrados de la biblioteca
   - En el caso de que ya exista un usuario registrado con el mismo DNI se notifica el error y no se añade el nuevo usuario





# Cómo se expresan los D-requisitos

Pueden usarse los mismos mecanismos utilizados para expresar los C-requisitos:

- *lenguaje natural*, *casos de uso* y *casos de uso detallados*
- aunque quizás alcanzando un mayor nivel de detalle

También pueden utilizarse *diagramas de clases* (si estamos realizando un análisis orientado a objetos)

- muestra las principales clases de objetos existentes en nuestro problema y las relaciones entre ellas (modelo del dominio)

Los límites C-requisitos ⇔ D-requisitos ⇔ Diseño no están claros

- C-requisitos ⇔ D-requisitos: el límite depende del grado de conocimientos técnicos del cliente
- D-requisitos ⇔ Diseño: en los D-requisitos no deben aparecer detalles correspondientes a la implementación

# 1.8 Diagrama de Clases en UML

**El Lenguaje Unificado de Modelado (UML) sirve para:**

- **analizar, construir y documentar un sistema software**

**El UML se está convirtiendo en un estándar de-facto en el desarrollo de software**

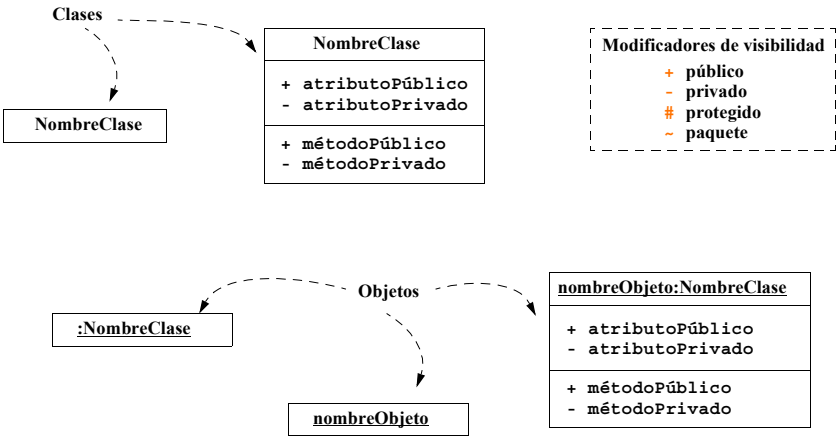
**Define varios tipos de diagramas para modelar el sistema**

- **Casos de uso, diagramas de actividad, diagramas de clases, diagramas de objetos, diagramas de secuencia, ...**
- **En la asignatura sólo usaremos dos: casos de uso y diagramas de clases**





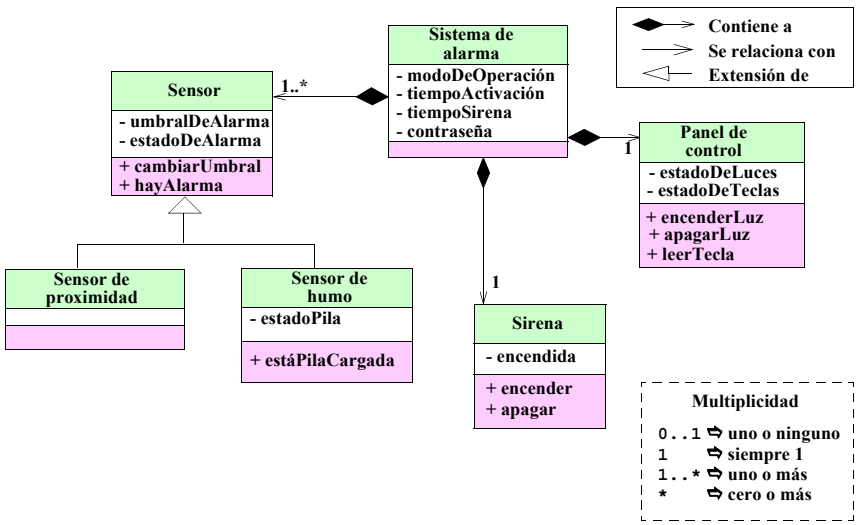
# Representación de clases y objetos








# Relaciones entre clases



# 1.9 Análisis orientado a objetos

**Tradicionalmente el software se desarrollaba desde una perspectiva algorítmica**

- **el principal bloque estructurador es la función**
- **este enfoque conduce a software difícil de adaptar y mantener**

**La tendencia actual es utilizar métodos orientados a objetos**

- **todo el ciclo de desarrollo del software está basado en los conceptos de objeto y clase**

**Las principales ventajas del desarrollo orientado a objetos son:**

- **es fácilmente comprensible puesto que existe un paralelismo entre las entidades del mundo real y las del sistema software**
- **ese paralelismo permite al cliente participar en las primeras etapas del desarrollo**
- **facilita la reutilización de código**





# Objetos y clases

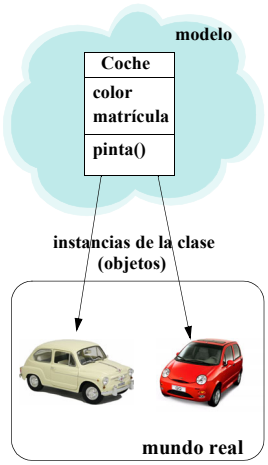
***Objeto*: entidad que existe en el espacio y en el tiempo, posee:**

- **identidad: nombre**
- **estado: atributos**
- **comportamiento: operaciones que otros objetos pueden realizar sobre él**

***Clase*: descripción de un conjunto de objetos con los mismos atributos y comportamiento**

**El análisis orientado a objetos realiza el modelado del problema en términos de:**

- **clases**
- **objetos**
- **comunicación entre objetos**
- **herencia y polimorfismo**



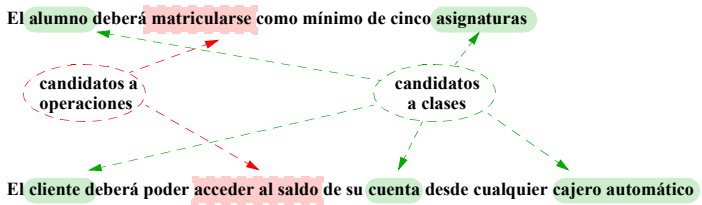





# Identificación de clases

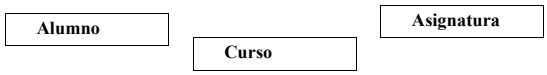
**Durante las primeras fases del análisis se procede a la identificación de las clases que componen nuestro problema:**

- **espacio de nombres del problema ⇨ candidatos a clases**
- **relaciones (verbos) entre los nombres ⇨ candidatos a casos de uso (u operaciones de las clases)**

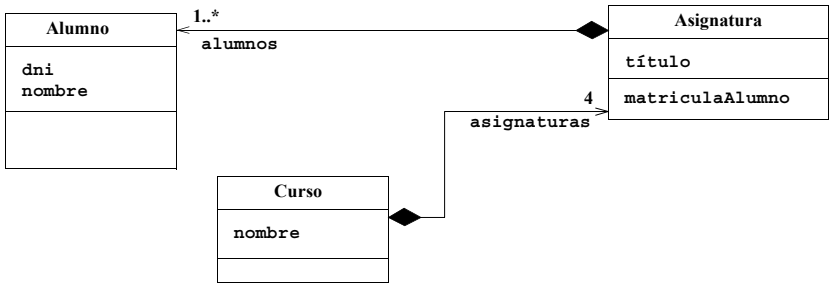


# Modelo de análisis

**Se comienza identificando las clases que forman el dominio de nuestro problema**




**Posteriormente se identifican las relaciones entre clases, sus atributos y sus métodos**

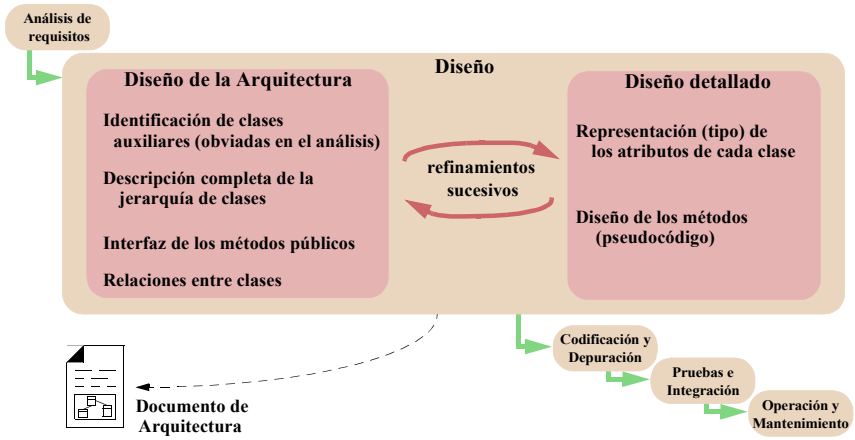







# 1.10 Diseño orientado a objetos

**Continuación del análisis profundizando en la descripción de las clases**



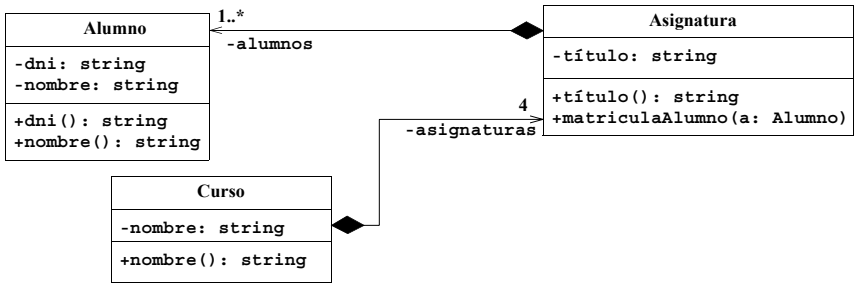





# Modelo de diseño

**Añade detalles al modelo de análisis (tipos, visibilidad, etc.)**

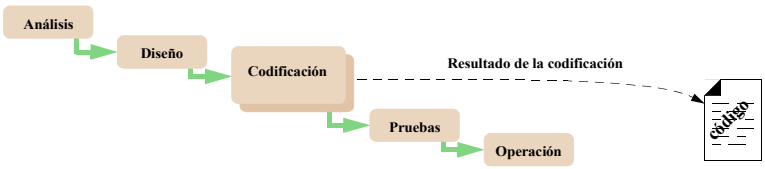
- **también se añaden clases auxiliares (en caso de que las haya)**




**El diagrama final deberá incluir todo lo necesario para *verificar todos los requisitos* de nuestro sistema**

Los problemas a tratar en esta asignatura serán sencillos por lo que, en general, no distinguiremos entre los modelos de análisis y diseño

# 1.11 Programación orientada a objetos




**La implementación de un diseño orientado a objetos es mucho más sencilla si se utiliza un lenguaje orientado a objetos**

**También es posible utilizar un lenguaje no orientado a objetos**

- **pero se repite mucho código y no se aprovechan todas las ventajas del diseño**

**Un lenguaje orientado a objetos (como Java):**

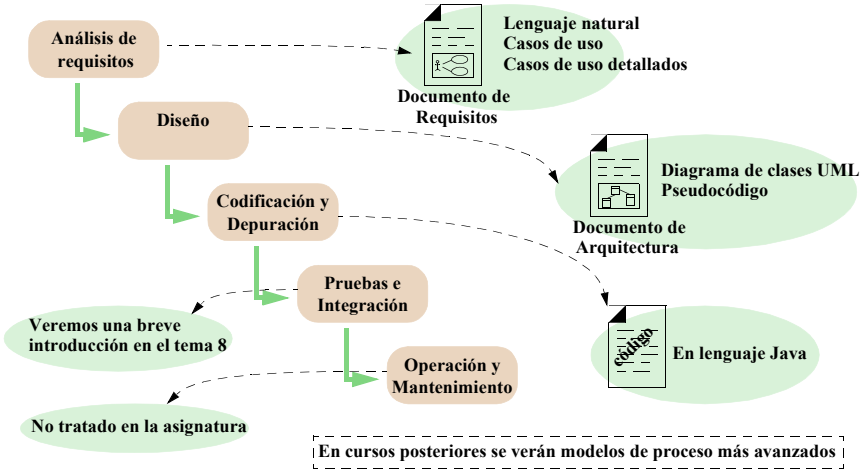
- **proporciona primitivas para definir clases y objetos**
- **soporta de forma directa la herencia y el polimorfismo**





# 1.12 Resumen

**Proceso de desarrollo (sencillo) que seguiremos en la asignatura**








# 1.13 Bibliografía

- **Eric J. Braude, "Ingeniería de Software". Alfaomega, 2003.**
  - **Parte de los capítulos 1, 3 y 4**
- **Ian Sommerville, "Ingeniería de software" (6ª edición). Pearson Educación de México, 2002.**
  - **Parte de los capítulos 3 y 5**
- **Russell Miles, Kim Hamilton. "Learning UML 2.0". O'Reilly Media, Inc. April 25, 2006.**
  **Disponible *on-line* en la biblioteca:**
  `http://proquestcombo.safaribooksonline.com/0596009828`
  - **Parte de los capítulos 2, 4, 5 y 6**

**Este tema sólo pretende ser una breve introducción a algunos aspectos de la Ingeniería del Software. Los contenidos abordados abarcan una pequeña parte de los contenidos tratados en los libros citados**