

## TEMA 1

Software = Instrucciones + Estructuras de datos + Información descriptiva (documentación)

**Ingeniería del Software** enfoque Sistemático, Disciplinado y Cuantificable en el desarrollo, mantenimiento y uso del software. (Principios de la ingeniería)

El software es de calidad si se ha seguido un proceso eficaz para crear un producto útil, que satisfaga los requerimientos y propone un valor medible.

## **FACTORES DE CALIDAD MC CALL**

**Características Operativas** utilización

**Confiabilidad** Probabilidad de operar sin fallos durante un tiempo

**Corrección** Satisface las especificaciones y objetivos

**Utilizable** Interfaz y documentación

**Integridad** Control de acceso

**Eficiencia** Recursos moderados

**Características de Modificación** Revisión

**Facilidad de mantenimiento** Esfuerzo para corregir errores

**Flexibilidad** Capacidad para adaptarse a nuevos requisitos

**Facilidad de prueba** Esfuerzo requerido para probar que funciona

**Características de Adaptabilidad** Transición

**Portabilidad** Esfuerzo para cambiar el ambiente hardware o software

**Reutilizable** Grado en el que las partes son reutilizables

**Interoperatividad** Esfuerzo para acoplar un sistema a otro

## **Suficientemente bueno**

Mala calidad = barato → Nadie lo compra

Buena calidad = caro → No sale rentable

El termino medio para no ser rechazado ni demasiado caro que no pueda ser terminado

- A veces es aceptable porque el uso ayuda a depurar errores
- En otros casos es un delito negligente entregar software con fallos
  - Errores conocidos (Como en aviones y eso)

Todo esto comenzó en 1968 Alemania, organizado por la OTAN

Errar es humano pero eso no es excusa

## Factores de éxito

El software es difícil de diseñar porque los requisitos cambian

- Respaldo financiero y emocional de la dirección
- Comunicación y buenas relaciones
- Implicación del cliente en la toma de decisiones y requisitos
- Cualificación del personal
- Optimización del negocio

Tendencia a que el modelo Ágil de mejores resultados que Cascada

## **CAPAS DE LA INGENIERÍA DEL SOFTWARE**

**Objetivo** Compromiso con la calidad

**Proceso** La metodología: el qué cuándo cómo y quién

Para lograr el objetivo

**Método** Técnicas: las tareas ejecutar

Para seguir el proceso

**Herramientas** Apoyo

Par realizar el método

## **ACTIVIDADES DEL PROCESO**

**Estructurales** Sin excepción requeridas en todo proyecto

**Comunicación** Entender los objetivos, obtener requisitos

**Planeamiento** Crear un plan que guíe al equipo

→ Administración del riesgo

→ Determinar cuáles son los entregables (artefacto artefacto)

**Modelado** Bocetar el objetivo, luego se refinará

**Construcción** Generación de código y pruebas

**Despliegue** Entregas del software al consumidor

**Sombrilla** Dependen del proyecto, no son siempre necesarias

**Seguimiento y control** De la hoja de ruta

**Administración del riesgo**

**Asegurar la calidad** Define las actividades necesarias

**Revisiones técnicas** Define las actividades necesarias

**Medición** De factores para el control de calidad

**Administración de la configuración**

Reutilización Obtener componentes reutilizables

Preparación y producción Creación de modelos y documentos

## **TIPOS DE FLUJO DE PROCESO**

**Lineal:** Ejecuta las actividades estructurales en secuencia

Es útil en pequeños proyectos que no suelen partir de 0

Son modificaciones con los requisitos bien definidos

**Iterativo:** Lineal donde las actividades se repiten varias veces antes de pasar al siguiente

**Paralelo:** Ejecutan varias actividades a la vez

**Evolutivo:** Realiza las actividades en secuencia de forma cíclica, incremental, iterativa.

Porque obtener todos los requisitos al principio es imposible. El producto mejora hasta obtener una versión completa funcional.

Es el más apropiado para grandes proyectos. Se realizan análisis de riesgo al final de cada etapa. El coste y esfuerzo se ajustan con la retroalimentación

En las primeras versiones se entrega un prototipo.

→ Evolutivo: Si se va a convertir en el producto final hay que aplicar criterios de calidad desde el principio

→ Desechable: Lo más recomendado, no hay que obtener calidad

## **Responsabilidad**

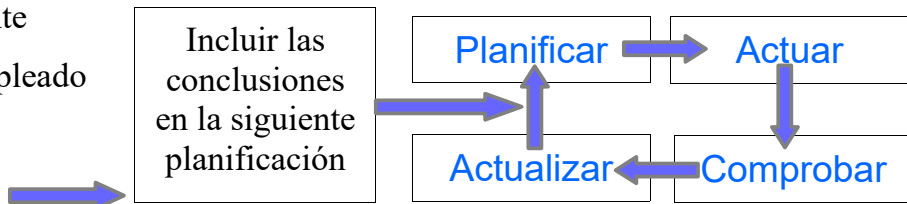
La ingeniería del software tiene responsabilidad

## TEMA 2

Gestión del proceso: organizar el trabajo en equipo

**TQM:** La calidad del producto viene dada por la calidad del proceso

- Atención al cliente
- Involucrar al empleado
- Medir la calidad
- Mejora continua



Hay empresas que crean estándares de calidad (certificados):

Demostrar que eres capaz, comparado con un modelo de buenas prácticas. Porque las empresas buscan garantías.

## **CERTIFICADOS**

Son mucho papeleo

**ISO 9000:** Normas muy generales que garantizan la calidad del proceso. Aplicables a cualquier organización. Mejora la imagen de una compañía, su productividad, organización y gestión de la calidad.

No especifica como se debe aplicar

**ISO 9000-3 2005:** Guía aplicable al software

**ISO/IEC 12207:** Aplicable a empresas productoras y consumidoras. Especifica todo el proceso

**CMMI:** Estudia la madurez de la empresa con formularios

Se estructura en 5 niveles, te dice cómo puedes subir de nivel.

→ **Nivel 1:** No hay buena planificación, los éxitos se deben a una única persona

→ **Nivel 2:** Cohesión entre grupos y mejor gestión del proyecto

→ **Nivel 3:** Buena gestión del proceso y trabajo en grupo. Tiene revisiones entre pares y de una persona ajena al desarrollo (walkthrough)

→ **Nivel 4 y 5:** Depuran el proceso

Se valoran las buenas practicas definidas por otras empresas que vienen a evaluarte. El mínimo es el nivel 3. El 5 es la mejora continua

## **LA CORRIENTE ÁGIL**

**Estandarizar el proceso resta libertad** Las metodologías ágiles optan por la flexibilidad sin renunciar a la calidad.

Tienen una relación más cercana con el cliente.

Se concretan fechas y funcionalidad des para la entrega de software incremental.

**Individuos e interacciones** antes que el proceso

**Software funcional** antes que una documentación extensa

**Responsabilidad** antes que seguir un plan estricto

Colaborar con el cliente en lugar de negociar con él

**PSP** Características de cada trabajador, tiempo necesario y la envergadura del trabajo. Se necesita más de un trabajo para rellenar las plantillas  
Una plantilla por usuario y por área.

**TSP** Agrupa a los trabajadores por su PSP para la formación de equipos de trabajo

## **PROCESO UNIFICADO**

Dividido en 4 fases que se repiten antes de pasar a la siguiente

**Fase de inicio:** Se realiza el modelado de negocio y se comienza a recoger requisitos

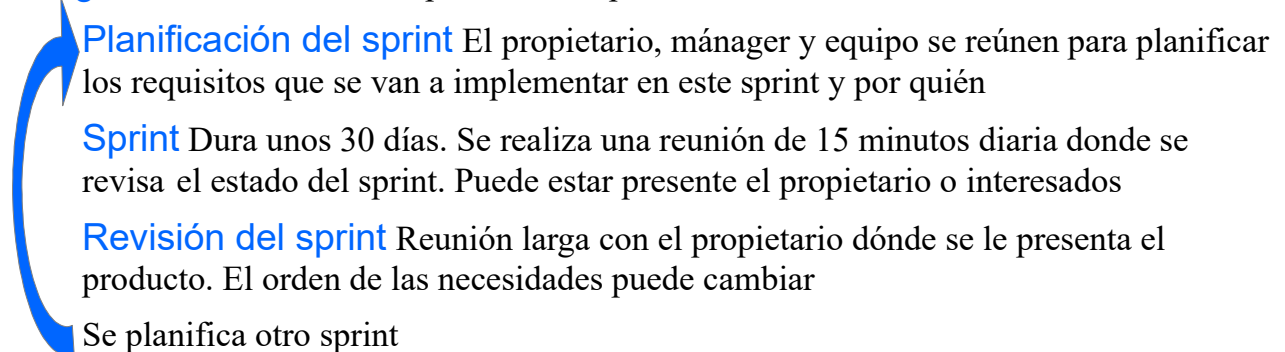
**Fase de elaboración:** Acaba el modelo de negocio, se realiza la recogida de requisitos, análisis y diseño es muy importante.

**Fase de construcción:** Con las sucesivas repeticiones se para recogida de requisitos, cero análisis y diseño. Los esfuerzos se centran en la implementación y un poco de pruebas

**Fase de transición:** Se realizan las pruebas y acaba la implementación. Termina con el despliegue

## **SCRUM**

**Recogida de necesidades** por orden de prioridad



## TEMA 3

### Gestión de un proyecto software

Todas las fases. Planificación, supervisión, control del proceso y del personal a lo largo de la evolución


### **4P GESTIÓN EFECTIVA DE:**

**El Personal** Dependencia del factor humano

Criterios de selección: Experiencia, habilidad, actitud

Maximizar: Coordinación y motivación

- Gerentes:
  - Ejecutivos: Manejan la empresa
  - Proyecto: Líderes del equipo
- Profesionales: El equipo de desarrollo
- Clientes != Usuario final
- Usuario final



StakeHolders:  
los participantes

**El Producto** Antes de comenzar el proyecto hay que definir

- Ámbito: Contexto dónde debe funcionar, objetivos, datos de entrada y de salida, restricciones de rendimiento
- Objetivos
- Restricciones técnicas

**El Proceso** Marco de trabajo para establecer un plan de desarrollo

- Seleccionar el proceso adecuado a las características del producto y el equipo de trabajo

**El Proyecto** Etapas, actividades y tareas para alcanzar un objetivo

El plan de proyecto es correcto si se logra un producto de calidad a tiempo y dentro de los recursos

- El gerente debe entender el problema, construir un equipo eficiente, asignar el tiempo correctamente y analizar el resultado una vez terminado para mejorar en el futuro

### **LA PLANIFICACIÓN: ESTIMACIÓN**

Determinar recursos, dinero, esfuerzo y tiempo necesario

Le corresponde al gerente. Lo toma en base a

#### Criterios de estimación

- Información de los participantes
- Métrica de propuestos anteriores

- Experiencia personal

## Precisión

- Complejidad y tamaño
- Incertidumbre en los requisitos
- Disponibilidad de históricos

## Recursos Disponibilidad, momento y tiempo requeridos

**Humanos:** El gerente determina número y esfuerzo tras la estimación  
Selección en base a habilidades

**Recursos software** (reutilizable) componentes

- **COST:** Se paga por él
- **Experiencia completa:** su proyecto es muy similar
- **Experiencia parcial:** hay que adaptarlos al proyecto
- **Nuevos:** Aún no se han desarrollado

**Entorno:** El hardware es la plataforma para el desarrollo del software  
Debe gestionarse por que se comparte entre proyectos

## Técnicas confiables de estimación

**Base histórica:** comparando y extrayendo (analogía) con proyectos similares

**Descomposición:** El coste total es la suma del coste de las partes  
EJ: Método Albretch (Puntos función)

## Técnicas empíricas

EJ: Cocomo

## Método Albretch

Descomposición Puntos Función

Conteo de funcionalidades entregadas al usuario

Los PF no se concentran en los aspectos técnicos de la codificación

→ Funciones de datos (grupos lógicos)

Necesidad de almacenamiento del usuario

- **ILF** (Internal Logic File): El programa los crea, lee y modifica
- **EIF** (External Interface File): Otra aplicación los mantiene
  - **DET** (Data Element Type): campos de información que reconoce el usuario
  - **RET** (Record Element Type): conjuntos de DETs que tienen sentido juntos
  - **FTR** (File Type Referenced): grupo de ILFs o EIFs (o uno solo cualquiera)

→ Funciones transaccionales

Procesos necesarios para el usuario

- **EI** (External Input): datos externos para mantener los ILFs
- **EO** (External Output): sacar cálculos, mantiene los ILFs
- **EQ** (External Query): no realiza cálculos ni mantiene ILFs, solo saca datos

→ PFNA: Puntos función no ajustados, suma de los puntos función de cada parte

→ PFA: Puntos función ajustados ( $PFA = PFNA * FA$ ) el Factor de Ajuste se adapta a las características propias del sistema. Se determina puntuando de 0 a 5 las 14 características por grado de influencia

El esfuerzo se mide en número de horas por hombre, se obtiene multiplicando los PFA por el número de horas por hombre por punto función que se tiene de media

## Método Cocomo

Es una técnica empírica: usa información de proyectos ya finalizados para generar una estimación. Hace una regresión con los datos para estimar las miles de líneas de código y el esfuerzo en horas de trabajador

Realiza 3 aproximaciones en 3 momentos clave

→ Modelos de estimación:

**Modelo de composición** Justo antes de empezar

**Modelo de etapa temprana** Requisitos definidos

**Modelo de etapa post-arquitectura**



Mayor  
precisión

## **CALENDARIZACIÓN**

La realiza el gerente, es de las más complicadas.

Identifica actividades y las interdependencias (orden)

Estima los recursos y asignan las responsabilidades

**EDT**: Estructura de Desglose del Trabajo

Las etapas generales se dividen en unidades de trabajo. Las unidades de trabajo tienen un código único y un responsable.

- **Hitos**: logros que se consiguen al terminar tareas
  - Entregas al cliente: un tipo de hito

Los hitos se usan para comprobar que se sigue el calendario o si hay que realizar alguna modificación

El gerente estima tiempo y esfuerzo

EJ: Asumir que todo va a salir bien y +30% de margen +20% de imprevisto

- **Gráficas de barras** (de Gantt): cuando comienzan, cuando acaban y quien es el responsable de las actividades
- **Red de actividades**: Relaciones de dependencia entre tareas



**Holgura:** cuanto se puede retrasar una actividad sin que eso afecte a la fecha de finalización

0 : actividad crítica |  $>0$  : no es actividad crítica

En el grafo el camino más largo desde el inicio hasta el final es el camino crítico, formado por actividades críticas

## **LA GESTIÓN DEL RIESGO**

**Riesgo** Probabilidad de que ocurra una situación adversa

- Afectan al calendario o a un recurso
- Afectan a la calidad del software
- Afectan a la empresa

El gerente los puede gestionar de forma:

**Reactiva** Preocuparse cuando tengan el problema y no antes

**Proactiva**

- Identificar los riesgos antes de comenzar el proyecto
- Ordenarlos por importancia, probabilidad o gravedad
- El equipo de trabajo intenta que no ocurran y deben elaborar un plan de contingencia para mitigar su efecto

## TEMA 4

# INGENIERÍA DE REQUISITOS

**Requisito:** una capacidad que debe tener el programa para que el usuario resuelva  
(según IEEE) un problema + su documentación

Comunicación → Objetivos → Requisitos (Deducción)

## TIPOS DE REQUISITOS

### Ámbito

- **Sistema:** Software + Hardware  
Si no hay Hardware → Sistema = Software
- **Software:** Desarrollo lógico
- **Hardware:** Entregar piezas de hardware específico

### Características Los requerimientos

- **Funcionales:** Cómo debe funcionar, lo que debe hacer
- **No funcionales:** consideraciones a tener en cuenta en los requisitos funcionales  
EJ: eficiencia, fiabilidad, ética, seguridad (cuestiones de negocio)
- **Información:** los datos con los que trabaja la aplicación

### Audiencia Las personas que a las que va dirigido (documentos)

- **Requisitos C** (cliente): Escritos en lenguaje natural plano.  
Se usan en las reuniones con el cliente, son la base para:
- **Requisitos D** (desarrollador): representación técnica de la aplicación

## ESPECIFICACIÓN DE REQUISITOS

Los elabora el ingeniero de requisitos C → D

- No tiene datos innecesarios
- No es ambiguo: usa un glosario de términos
- No faltan requisitos (completo) Todas las respuestas de la aplicación están determinadas
- No hay conflictos entre requisitos (consistente)
- Nunca se repiten requisitos, se especifican independientemente aunque pueden referenciarse entre ellos
- Los requisitos están identificados para facilitar su identificación
- Se cuenta con matrices de trazabilidad que muestran las interdependencias, a fin de poder determinar los daños colaterales que causa un cambio

- Se ordenan por importancia y lo susceptibles que sean al cambio
- El documento debe ser fácil de actualizar
- No se habla de diseño ni de implementación

Es importante detectar errores en la definición de requisitos porque cuesta más solucionarlos en la fase de desarrollo y mucho más en la fase de mantenimiento

## **FASES DE LA INGENIERÍA DE REQUISITOS**

### **Educación** (elicitación)

Obtener el dominio del problema, entender el contexto

Los requisitos se negocian y se resuelven los conflictos

Se obtienen los requisitos C

### **Análisis**

Se usan los requisitos C, sin contradicciones, para depurarlos y profundizar en el problema.

Se establecen las bases para el diseño y se obtienen los requisitos D

### **Validación**

Se corrigen los requisitos C y se comprueban si coinciden con las necesidades del cliente

Se forman los requisitos C y D, se obtiene un prototipo

El proceso se repite iterativamente para perfeccionar los requisitos mediante el prototipo.

Durante todo el proceso es necesaria la interacción del cliente

## **TÉCNICAS DE EDUCACIÓN DE REQUISITOS**

### **Entrevistas**

→ Preparación: Se fijan los objetivos y la persona a entrevistar, se planifica y estudia el problema

→ Realización: Se rellenan las plantillas, se usa un lenguaje no técnico y se recapitula al final

→ Análisis: Se contrasta la información y se genera la documentación

### **JAD** Reuniones en grupo

→ Adaptación: Se decide la duración (de 2 a 4 sesiones), el número de participantes, recaban información del contexto.

→ Sesiones: Exponen sus ideas y el analista recoge los requisitos usando plantillas. Se guardan las dudas para la siguiente sesión

→ Conclusión: El jefe de proyecto revisa con los demás, se envía al ejecutivo para que tome decisiones

**Brainstorming** La más conocida y usada

- Preparación: Se seleccionan los participantes y un jefe de sesión
- Generación: Se expone el problema y se generan las ideas, todas deben ser visibles
- Consolidación: Se revisan y priorizan las ideas, pueden descartarse las menos adecuadas
- Documentación: El jefe escribe las ideas priorizadas y los comentarios de la consolidación

**Casos de uso** Documentar los requisitos funcionales desde el punto de vista del usuario

- Diagrama: Actores, interacciones con los casos de uso del sistema
- Descripciones textuales: se usan plantillas

**Prototipado** modelo/maqueta que ayuda a educir y validar requisitos

Los usuarios se hacen una idea de lo que van a recibir

- Prototipado de interfaz de usuario: Pueden ser solo unos dibujos de la pantalla (storyboard) Verifican el diseño de la UI y la accesibilidad e los requisitos
- Prototipado funcional: autentica simulación, ligado la desarrollo iterativo  
No se descarta, es una primera versión que se refinará hasta la versión final

## **DOCUMENTO DE ESPECIFICACIÓN DE REQUISITOS**

D	Documento	(DRS)
RS	de Requisitos del Sistema	=
E	Especificación	(ERS)

Usaremos el modelo MADEJA: el de la Junta de Andalucía

- **Portada:** nombre, versión, fecha, implicados
- **Lista de control de cambios:** parte de la trazabilidad
- **Índices y listado de tablas y figuras:** organiza el documenta, mayor legibilidad
- 1. **Introducción:** alcance y objetivos
- 2. **Dominio del problema** para conocer a la empresa
- 3. **Situación actual** de la empresa (pros y contras)  
Entorno hardware y software, aplicaciones ya existentes
- 4. **Necesidades del negocio** objetivos del negocio
- 5. **Descripción de subsistemas** o del sistema si solo hay uno
- 6. **Catálogo de requisitos** funcionales (información, reglas de negocio, conducta)  
no funcionales (fiabilidad, estabilidad)
- **Matrices de trazabilidad** (en realidad no viene en la plantilla)
- **Actas de las reuniones**
- **Documentación adjunta**
- **Glosario**

# GESTIÓN DE LOS CAMBIOS EN LOS REQUISITOS

Todos los cambios se deben documentar y controlar.

Las modificaciones en la línea base requieren aprobación  
(Línea Base  $\sim$  especificación de los requisitos)

**Análisis y evaluación** estimación de costes, identificación de requisitos afectados o dependientes

**Valoración** estudiar la viabilidad económica

**Análisis de requisitos y modificaciones** toma una decisión

→ Rechazar el cambio: negociar con el cliente

→ Aceptar el cambio:

**Modificar productos afectados**

**Establecer una nueva línea base**

} **Gestión  
del cambio**

## CASOS DE USO

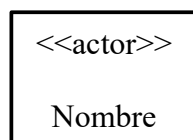
Secuencia de interacciones entre el sistema y los actores

**UML** Lenguaje gráfico unificado (Universal Modeling Language)

**Actor** Persona o Sistema

- **Primario**: activa la funcionalidad
- **Secundario**: intervienen pero no son el que activa la funcionalidad

Se dibuja como un monigote o con una caja con el estereotipo



<<actor>> es el estereotipo

Nombre

**Sistema** Caja negra, es solo lo que percibe el usuario

**Caso de uso** (CU)

Se dibuja como una elipse con un verbo en infinitivo dentro

Verbo en infinitivo

Jerarquías: algunos usuarios están contenidos dentro de otros o son alteraciones

Los casos de uso también pueden ser abstractos y estar relacionados el uno con el otro

## Inclusión

A incluye a B: Cada vez que se produce A, se produce B  
Que se produzca B no hace que se produzca A  
B suele ser abstracto  
Se dice que B está incluido en A

<<include>>  
A -----> B

## Extensión

A extiende a B: Cada vez que se produce B, podría producirse A  
Que se produzca A no hace que se produzca B  
A puede ser tanto abstracto como concreto  
Se dice que A extiende la funcionalidad de B

<<extend>>  
A -----> B

Inclusión y Extensión no implican ninguna relación de parentesco

## Generalización

B hereda la funcionalidad de A, B es el padre de A  
B es una generalización de A, A es una especialización de B

A —————> B

**Subsistema** Es un conjunto coherente de casos de uso. Se representan con carpetas  
La unión de subsistemas forman el sistema completo

**Especificación textual** Usa lenguaje natural. No hay un solo formato, existen plantillas con diferente grado de detalle (precisión)

- **Nombre y código** Del caso de uso
- **Versión**
- **Evento de activación**, actores que la utilizan
- **Precondición** Pasos a seguir en la interfaz y estado previo del sistema
- **Postcondición** El estado en el que queda el sistema después de la ejecución
- **Cuota de tiempo** Para el rendimiento de un paso
- **Frecuencia de uso**
- **Importancia y estado**
- **Secuencia normal** Los pasos a dar suponiendo que todo va bien  
Condiciones y acciones del actor o del sistema, puede implicar la realización de otro caso de uso por inclusión (incondicional) o por extensión (condicional)
- **Excepciones** Eventos fuera de la secuencia normal  
Con una respuesta asociada, una secuencia alternativa. Abortar, reiniciar, continuar

## TEMA 5

**Modelo** Simplificación de una realidad compleja

Ayudan a visualizar el sistema, documentan las decisiones

Se realizan desde un punto de vista concreto. Un único modelo no es suficiente, se necesitan varios modelos desde diferentes puntos de vista

La calidad del modelo determina la calidad del desarrollo → y esta la calidad del producto

**Modelo estático** (**datos**) Diagrama de clases

**Modelo de dominio:** diagrama de clases sin métodos

**Modelo de diseño:** diagrama de clases con métodos

**Modelo funcional** (**funciones**) Diagramas de secuencia o Diagrama de colaboración

//Preferiblemente el de secuencia

**Modelo dinámico** (**eventos**) Diagrama de estados

## **DIAGRAMA DE CLASES**

NO son clases software, las clases representan:

- Cosas del mundo real ¡Solo las relevantes!
  - Objetos EJ: Un libro
  - Roles EJ: Un socio de biblioteca
- Eventos
- Interacciones

Se usan para construir los requisitos del sistema, derivan de los requisitos de información

### **Clase UML**

Nombre	Formado por un nombre común singular [con un atributo] En cursiva (o negrita) si es abstracto
Atributos ...	Los datos que se necesitan de esa clase
Métodos(parámetros) ...	Servicios que ofrece el objeto Sus responsabilidades

### **Asociaciones UML**

Relaciones entre los objetos del mundo real

Rol A	Nombre de la relación ►	Rol B	La dirección no es necesaria (o con la punta de flecha)
Mul A		Mul B	

La multiplicidad por defecto es 1

0...1	Opcional
1...1 (1)	Obligatorio
0...* (*)	Multiplicidad opcional
1...*	Multiplicidad obligatoria
n...n	Cualquier otra multiplicidad limitada

## Tipos de relaciones (asociaciones)

→ **Reflexiva** Un objeto con otro del mismo tipo

Debe aparecer el rol obligatoriamente

EJ: Jefes o matrimonios

→ **Derivativa** Se infieren de forma automática por otras relaciones

No es necesario escribirlas, si se incluyen deben ir precedidas de /

→ **Agregación** Relación todo-parte poco estricta



La parte NO se guarda en el todo, pueden existir sin un todo o formar parte de más de un todo

→ **Composición** Relación todo-parte muy restrictiva



La parte se guarda en el todo, NO puede existir sin el todo y NO puede pertenecer a más de un todo

→ **Generalización / especificación** Relación es-un

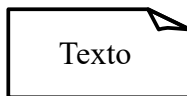
La superclase generaliza, la subclase especifica



Hereda atributos y métodos

- {completa}: La suma de todas las subclases es la superclase
- {incompleta}: Hay casos sin subclase
- {disjunta}: Las subclases no pueden tener instancias en común
- {solapada}: Existen instancias que encajan en varias subclases

## Comentarios



Se puede usar como aclaración o para explicar restricciones sin la notación específica

## Restricciones

{nombre} Dónde el nombre especifica el tipo de restricción, puede aparecer uniendo relaciones o al lado de las clases

- {subset} Una relación se da con un subconjunto de los elementos de otra relación
- {ordered} El orden importa en un conjunto
- {xor} En relaciones, puede darse una u otra pero no las dos a la vez

## MODELADO FUNCIONAL

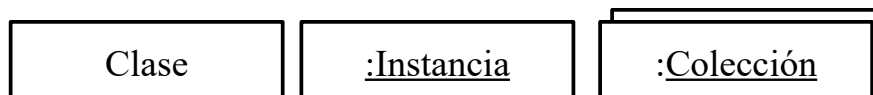
obj1  $\xrightarrow{m1()}$  obj2 El obj1 solicita la ejecución del método m1() del objeto obj2

En el diagrama de colaboración hay que numerar para indicar el orden de las acciones

En el diagrama de secuencia no es necesario, se lee de arriba a abajo (aunque se puede)

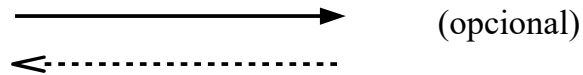
## Mensajes

- Síncronos  $\longrightarrow$
- Asíncronos  $\longrightarrow$

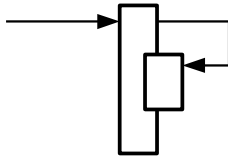




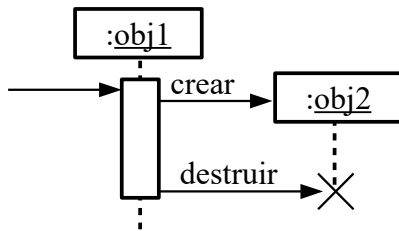
Mensajes con retorno



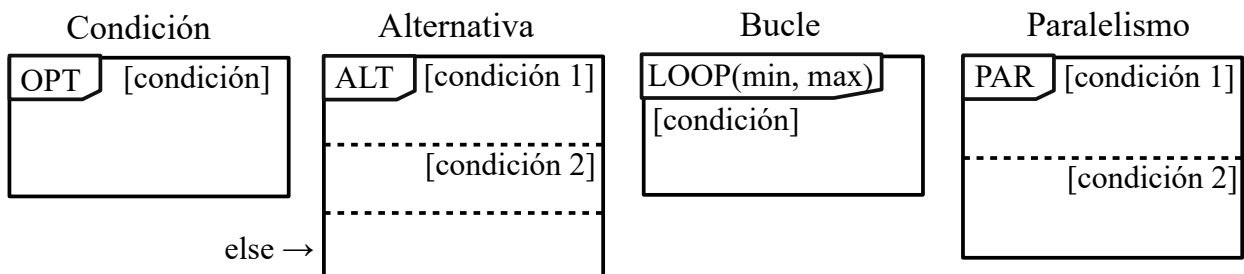
Mensajes a sí mismo



Construcción / Destrucción de una instancia



Fragmentos combinados



Responsabilidades UML (obligaciones)

Se elaboran junto a los diagramas de colaboración/secuencia

- Un método propio
- Sus atributos privados
- Hacer llamadas a otros objetos
- Conocer los datos de otros objetos
- Coordinar objetos
- Cálculos

Los métodos se agregan a las clases para cumplir las responsabilidades → Ellos solos o colaborando con otros métodos de otros objetos

## **MODELADO DINÁMICO**

Diagrama de estados Ciclo de vida (estados) de los objetos

Las clases estáticas no lo necesitan, son meros contenedores de información

Estado: ○ ESTADO

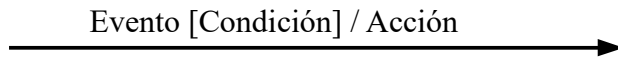
Transición: →

Estado inicial: ●

Estado final: ●

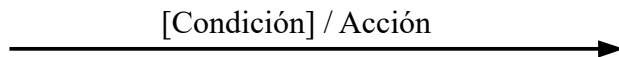
Puede haber varios estados finales. Del estado final no se sale  
No todos los objetos necesitan estado final

## Transición



Las acciones se realizan en el cambio de estado

## Transición automática



No necesita evento, ocurre al entrar en el estado

## Acciones en el estado

entry / Acción → Al entrar en el estado

Evento / Acción → Al ocurrir el evento estado en el estado

do / Acción → Siempre que se esté en el estado

exit / Acción → Al salir del estado

## Señales

Precedidas de ^

^objeto.señal(parametros)

## Pseudoestado inicial

Solo puede haber uno

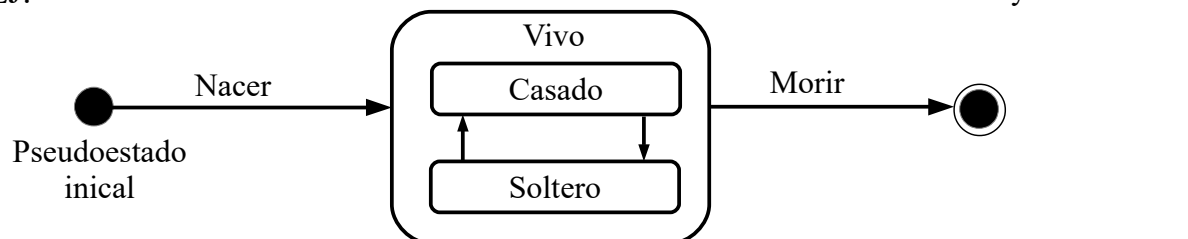
Representa el momento en el que el objeto aún no existe

Es el único estado que acepta eventos de creación

## Estados compuestos (anidados)

Un estado que tiene estados dentro

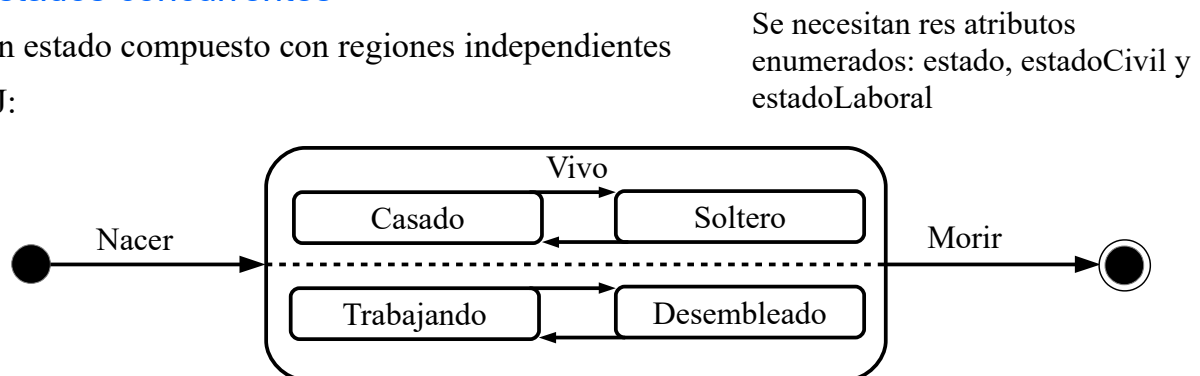
EJ:



## Estados concurrentes

Un estado compuesto con regiones independientes

EJ:



Los estados se pueden representar con booleanos, uno por cada valor de un atributo enumerado, con una regla XOR.

## TEMA 6

### **PATRONES GRASP**

Guía para asignar responsabilidades

#### Experto en información

Se le asocia la responsabilidad a la clase que tiene la información necesaria para realizarla

EJ: La clase venta tiene información sobre las líneas de pedido (cantidad) y producto (precio), es la escogida para calcular el coste total del pedido

#### Patrón creador

Determina la clase creadora

- Es una composición / agregación
- Registro: relación física del mundo real
- Tiene los datos necesarios para crear la clase

EJ: Venta es la clase que crea la línea de venta

#### Bajo acoplamiento

Una clase tiene alto acoplamiento si depende de otras clases para realizar sus responsabilidades

→ Alta dependencia

Alto acoplamiento = malo → Requiere más esfuerzo para comprender

→ Difícil de reutilizar

Reducir el acoplamiento reduce el coste de los cambios

EJ: El patrón creador diría que el registro crea el pago y solicita a una venta que agregue ese pago

Con bajo acoplamiento sería el registro el que le pide a venta que cree el pago y esta se lo agrega a la vez                      Reduce el acoplamiento

#### Alta cohesión

Las responsabilidades deben estar bien distribuidas y agrupadas

Una clase no puede realizar demasiadas tareas, menos si son diferentes (sin relación)

Las clases deben realizar un trabajo muy concreto y comparten el esfuerzo

#### Controlador

No pertenece a la interfaz de usuario

Recibe los eventos del sistema (un evento que ha sido generado por un actor externo)

Sus métodos son la interfaz pública del sistema

# **DCD: DIAGRAMAS DE CLASES DEL DISEÑO**

Agrega información sobre la implementación

## Visibilidad

+ Público                                      Si no es ninguna de las dos se deja sin especificar  
- Privado

## Tipos de datos

Atributo : [Lista de] Tipo [:= valor inicial]  
{frozen} ← Para una constante

Método : TipoDevuelto                      Los métodos también pueden ser +públicos o -privados  
{abstract} ← Si es abstracto

El número de clases del DCD puede variar con respecto al diagrama de clases

- Los métodos <<create>> se ignoran a menos que realicen alguna tarea específica
- Los getters y setters no se suelen incluir porque resultan sencillos y solo aportan ruido al diagrama
- Tipos de datos
  - Si se usan herramientas de generación automática de código es necesario especificarlos exhaustivamente
  - Si el diagrama es para los desarrolladores, solo genera ruido, no es necesario

## Transformación a código

Las asignación de atributos y métodos es directa pero no trivial, se trata del objetivo final

A más completo es el DCD, más sencilla es la implementación

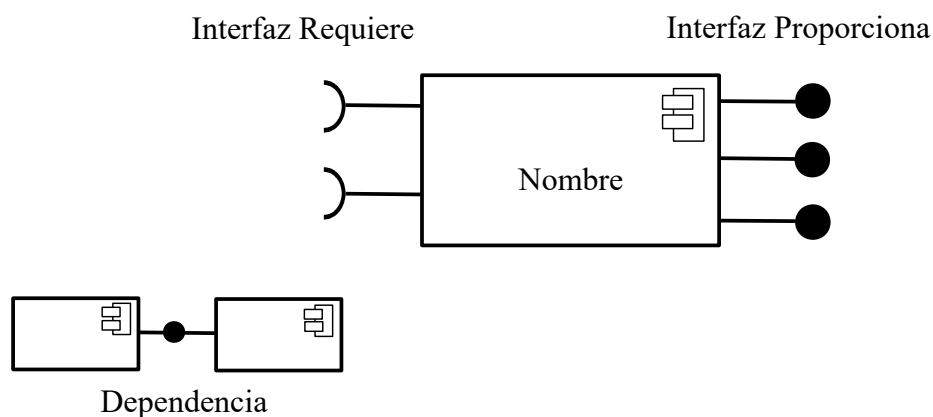
# **DIAGRAMAS DE IMPLEMENTACIÓN**

Las partes del software y dónde se ejecutan

## Diagrama de componentes

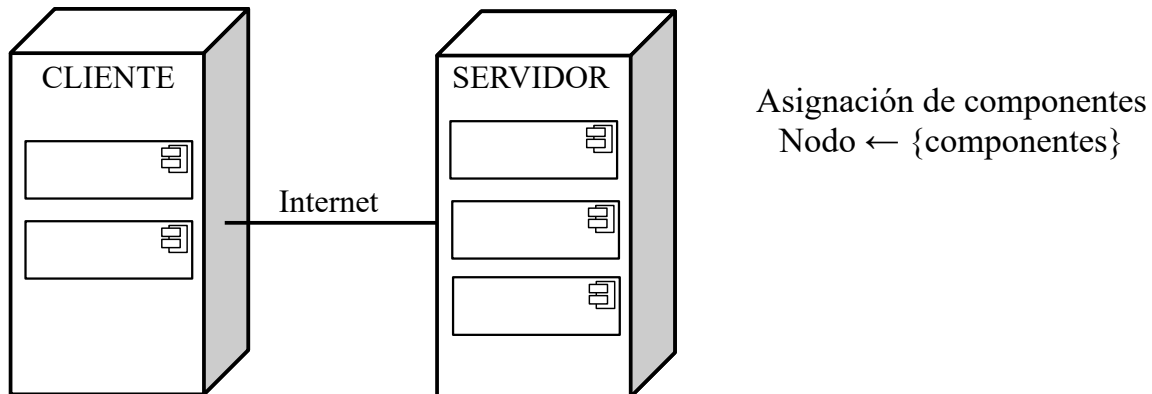
**Componente** Entidad ejecutable independiente y desplegable

Se accede por su interfaz y puede tener requisitos



**Diagrama de despliegue** Qué se ejecuta y dónde

**Nodo** Unidad que ejecuta, normalmente un ordenador  
Los nodos se conectan entre sí



## **MANTENIMIENTO DEL SOFTWARE A MEDIDA**

Modificar un sistema después de haber sido entregado

### **Reparar defectos**

- Errores de código Son fáciles de reparar
- Errores de diseño Son caros, puede ser necesario reescribir componentes
- Errores de requisitos Implica tener que rediseñar el sistema, son los más caros

**Mantenimiento del entorno** Hardware y Software

### **Agregar funcionalidades**

Requiere analizar en profundidad los cambios. Los motivos de su alto coste son:

- Se disuelve el equipo de desarrollo original
- El mantenimiento lo realiza una empresa diferente
- El equipo de mantenimiento puede no conocer el lenguaje/tecnología
- Los programas se degradan con los cambios  
(Si no se siguen herramientas de Ingeniería del Software)


### **Métodos de mantenimiento**

- **Reingeniería** El diseño se examina y modifica. La implementación se reconstruye  
Se usa en proyectos que se han degradado con el tiempo y es difícil agregar funcionalidades
- **Ingeniería inversa** La documentación se crea a partir del producto únicamente  
Se usa con software antiguo o sin documentación
- **Reestructuración del software** (refactorización)  
No se modifica el modelo, solo cambia el código fuente
  - Reestructuración de código: Misma funcionalidad, mejora la calidad y legibilidad
  - Reestructuración de datos: Las estructuras de datos se rediseñan bien definidos

## TEMA 7

### **PRUEBAS DE SOFTWARE**

Obtener software suficiente bueno, demostrar que funciona descubriendo defectos

Las pruebas solo pueden describir fallos, no pueden demostrar que un software esté libre de fallos 

Reflejar en un informe el resultado obtenido al ejecutar el programa con datos de prueba y el resultado esperado.

Lo realiza un equipo de integración y los programadores, que son los responsables

**Pruebas de componentes individuales** (unitarias)

Se prueban todos los métodos de todos los objetos

**Pruebas del sistema** (integración)

Se prueban los componentes juntos

**Pruebas de entregas**

Participan los clientes, como una caja negra

**Pruebas de regresión**

Una prueba que se repite porque se han agregado componentes pero la funcionalidad no debe cambiar

A más componentes → más pruebas

Los fallos podrían camuflarse

Machar el sistema para intentar que falle y ver como reacciona

**JUnit** Para automatizar pruebas unitarias en Java

## TEMA 8

Gestión del código y documentación  
(Configuración del software)

Versiones correctas y reversibles para llevar la cuenta de los cambios introducidos  
Almacenar versiones de los componentes, el sistema se compone de componentes

**Linea base:** un punto que se toma como referencia cuando se introducen cambios  
Contiene todos los requisitos conocidos  
Solo se puede modificar con el sistema de control de cambios

- Equipo de desarrollo      Genera nuevas versiones que incluyen los cambios en los requisitos
- Equipo de garantía de calidad      Si rechaza la versión, los desarrolladores realizan otra versión  
   Si la aceptan, se toma como una nueva linea base

## **PLANIFICACIÓN DE LA CONFIGURACIÓN**

**Elementos a gestionar:** especificaciones, diseños, datos de prueba

- Responsables y gestores                      → Políticas de gestión
- Herramientas                                      → Requerimientos y datos de versión

Formulario para la gestión del cambio

- Evaluar posibilidades y costo
  - Si es aceptado se realizan los cambios hasta que la calidad del software es la esperada
- Se establece la linea base  
Revisión formal y modificación del documento

## **IDENTIFICACIÓN DE VERSIONES**

**Versión** El sistema modificado

**Variante** Otro sistema equivalente para otro entorno (Hardware / Software)

**Entrega** Lo que se distribuye al cliente

→ **Numeración del documento**

Empresa		Proyecto		Tipo de		Revisión		Atributos
desarrolladora				documento				Extra

(lenguaje, fecha, ...)

## → Numeración de versiones

X.Y.[Z]

X: Principal

Y: Menor

Z: Mantenimiento

Principal: cambian las funcionalidades claves

X.Y.Etapa

Menor: se modifican las funcionalidades existentes y se corrigen errores

X: Principal

Y: Menor

Etapa: alpha,beta

Mantenimiento: una por entrega

RC, final

Fecha

YY.MM

## **EL CONTROL DE VERSIONES**

Mucho mejor que hacer copias de seguridad

- Git: Distribuido
- Subversion (SVN): Centralizado, no trabaja bien con ramas