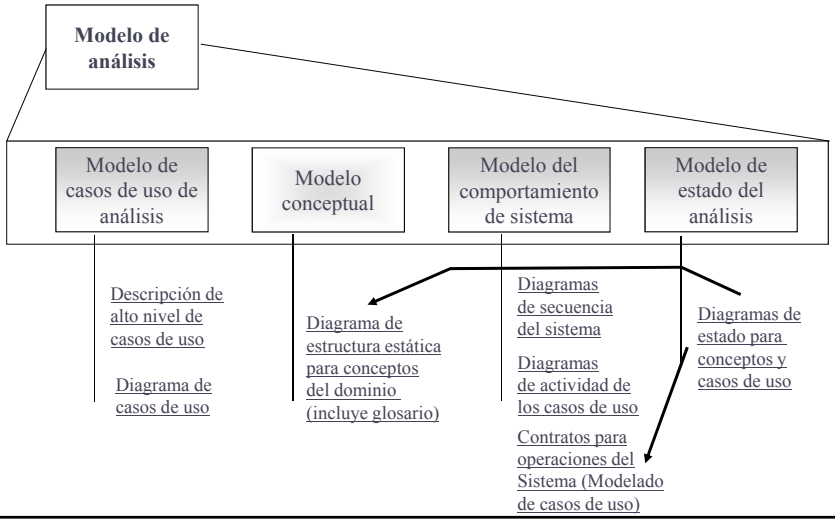
4.1 UML: transición al diseño y diagramas de interacción

Ingeniería del Software Avanzada
Técnicas de análisis y diseño

Objetivos

- Conocer la notación y el concepto de los diagramas de interacción
  - Ser capaces de generar un diagrama de secuencia del sistema a partir de una descripción de caso de uso
- Identificar el concepto de contrato y su utilización
  - Repasar el concepto de modelado de casos de uso e identificarlo con el concepto de contrato, en particular ser capaces de generar las pre y poscondiciones de un contrato a partir de los casos de uso y el modelo E/R o conceptual de clases
- Conocer y comprender el proceso de paso del análisis al diseño

Modelo de análisis para diseño



Fase de análisis: conclusión

Artefactos	**Objetivos**
• Casos de uso	• Procesos del dominio
• Modelo conceptual	• Conceptos y términos del dominio
• Diagrama de secuencia del sistema	• Eventos y operaciones del sistema
• Diagramas de actividad de los casos de uso	• ¿Qué hacen las operaciones del sistema?
• Contratos (Modelado de casos de uso)	

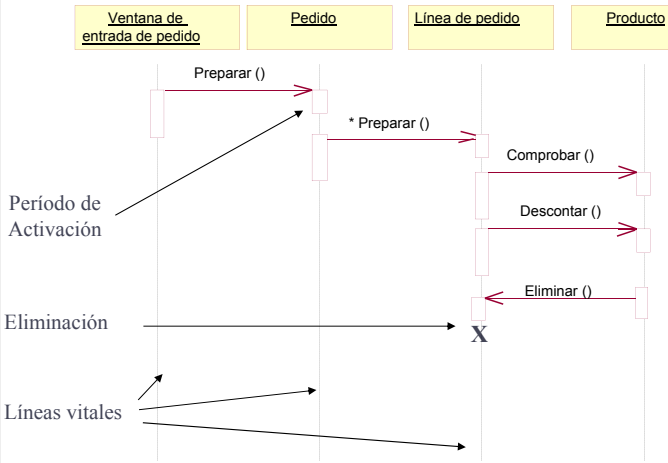
Diagramas de interacción

- Parte del modelo dinámico:
  - Ejecución: los objetos se envían mensajes entre sí
- Modelos que describen cómo colaboran grupos de objetos para un cierto propósito
- Típicamente representan comportamiento de un solo escenario
- Dos tipos para una misma información (cambio automático en herramienta):
  - Diagrama de secuencia
  - Diagrama de colaboración

Diagrama de secuencia

- Se centra en las secuencias de mensajes
  - Cómo se envían y reciben mensajes en grupo de objetos
- Tiene dos dimensiones:
  - Vertical: tiempo, línea vital del objeto (en la interacción)
  - Horizontal: conjunto de objetos (clases del modelo)
- Cada mensaje es un flecha entre dos objetos
  - Horizontal (inmediatez) o ligeramente descendente: existe un retraso de interacción
- Al pasar mensaje, hay expectativa de acción resultante:
  - Retorno de datos, creación de objetos, etc.

Ejemplo diagrama de secuencia

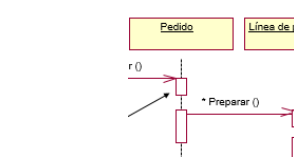


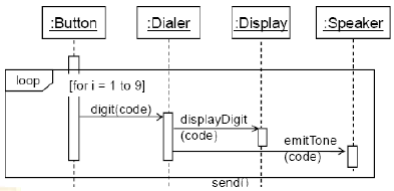
Notación de diagrama de secuencia convencional: Ejemplo de diagrama de secuencia en fase de diseño

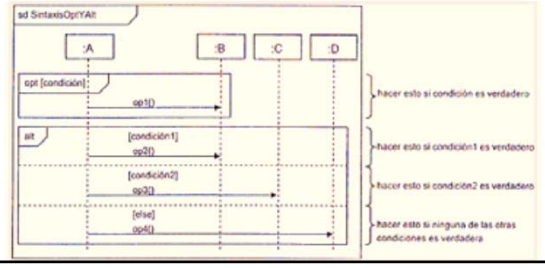
En fase de análisis: diagrama de secuencia del sistema (se verá a continuación)

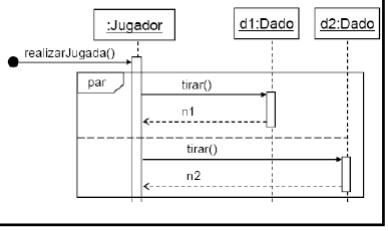
Diagrama de secuencia avanzado

- Posibilidad de repetición de un mensaje, bucle con varios mensajes, mensajes en paralelo, mensajes con condición









Operaciones del sistema

- Diagrama de secuencia de sistema:
  - Muestra gráfica de eventos de actores al sistema
  - Parte de la fase de análisis (modelado de casos de uso): flujo entre actores y caso de uso
  - Como caja negra, no hay que decidir responsabilidades de clase

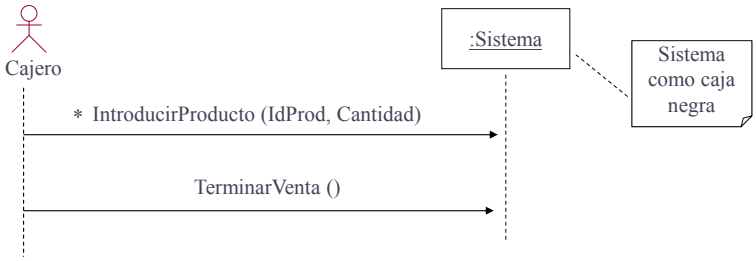
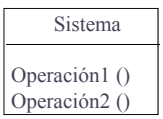


Diagrama de secuencia del sistema

- Representación de **eventos generados por actores** para un escenario de un caso de uso
- Evento de sistema:
  - Hecho externo de entrada que produce un actor en el sistema
- Operación de sistema:
  - Acción de respuesta a un evento



Crear diag. de secuencia del sistema

- Cómo elaborar un diagrama de secuencia de sistema
  - Trazar línea vital de sistema como caja negra
  - Identificar actores que operan sobre sistema
    - Una línea vital para cada uno
  - Siguiendo flujo normal del caso, identificar los eventos externos generados por los actores
- Consejos:
  - Nombre de eventos: comenzar con verbo (añadir, acabar,...)
  - Sintaxis como mensajes entre clases:
    - Retorno := mensaje (par: tipo par; par: tipo par...) : tipo retorno

Ejemplo diagr. de secuencia del sistema

Diagrama de secuencia del sistema a partir de esta descripción de caso de uso

1. El bibliotecario identifica un título	2. El sistema muestra la lista de ejemplares disponibles del título
3. El bibliotecario identifica un ejemplar	4. El sistema pide confirmar la baja
5. El bibliotecario confirma la baja	6. El sistema aprueba la baja mostrando los datos del ejemplar eliminado. Solicita si elimina los datos de préstamos relacionados con él
7. El bibliotecario confirma la eliminación de préstamos	8. El sistema confirma la eliminación mostrando el número de préstamos eliminados

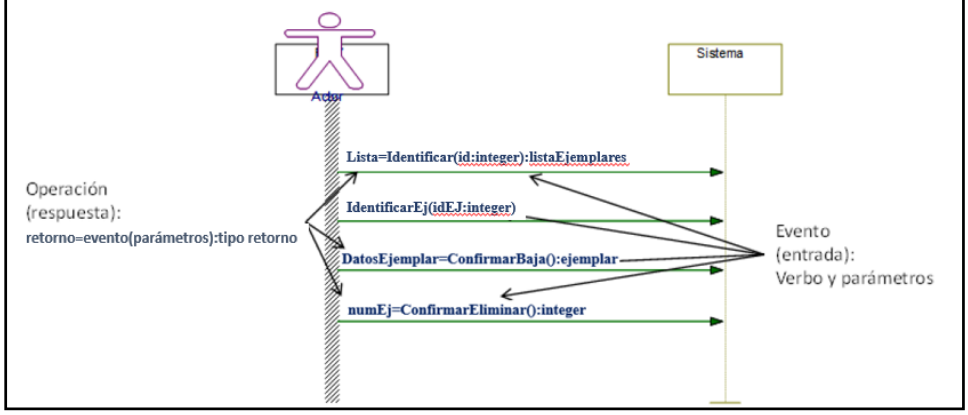
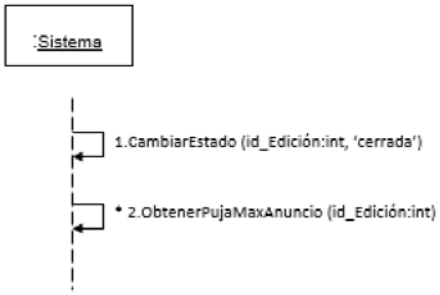


Diagrama de secuencia del sistema sin actor

- Casos de uso lanzados por el Sistema

SISTEMA
1.Cambia estado de subasta a cerrada
*2.Obtiene la puja máxima de un anuncio



Contrato de operación de sistema

- Definido por B. Meyer:
  - Documento que describe la operación que se propone
  - Estilo declarativo: qué sucederá (detalles que aseguren que es factible) y no cómo se hará (clases, métodos)
  - Expresar cambio de estado de pre- y poscondiciones
  - No es propio de UML
- Aplicación general:
  - A cualquier nivel: desde caso de alto nivel a método detallado de una clase
  - En la asignatura: limitaremos a contratos para cada caso de uso; definición equivalente a modelado de casos de uso

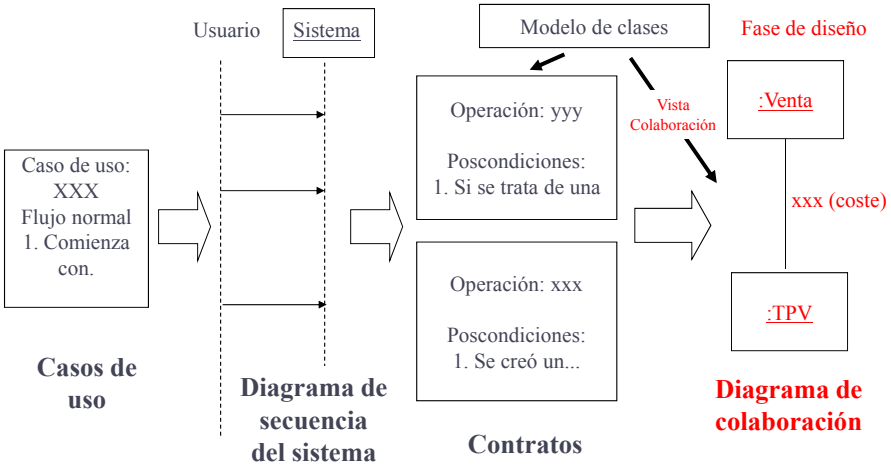
Formato de contrato (B. Meyer)

- Diagrama de secuencia del sistema: nombre de operaciones, parámetros, retorno y tipos de parámetros y retorno
- Responsabilidades, que debe cumplir (informal)
- Referencia cruzada con requisitos/funciones
- Notas de diseño, algoritmos, interfaces, etc.
- Excepciones
- Salida: mensajes o registros al exterior
- Precondiciones: estado del sistema antes de operación
- Poscondiciones: estado posterior

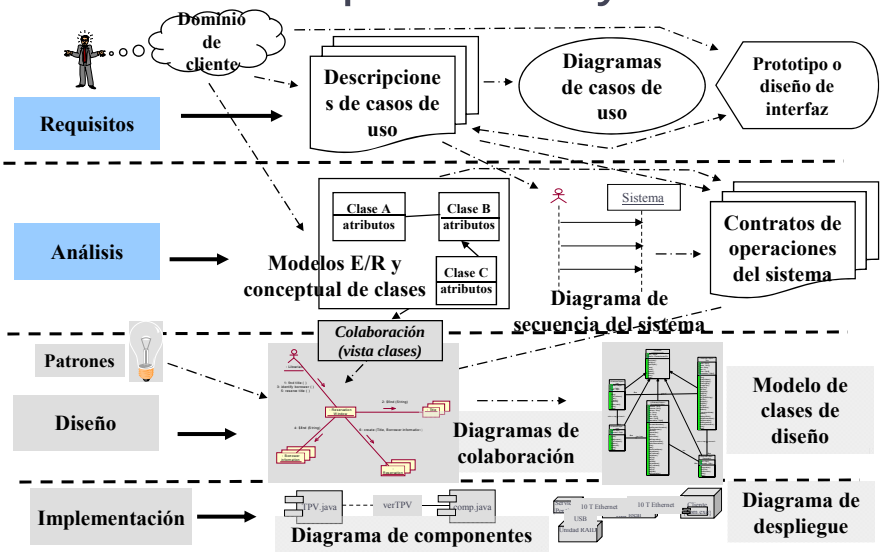
Repaso modelado de casos de uso

- En la asignatura similar a contrato:
  - Elaborar un contrato para cada caso de uso
  - Poscondiciones:
    - Declaración de cambios en modelo conceptual (en pasado: "Se creó un ejemplar de X"):
      - Creación y eliminación de ejemplares
      - Modificación de atributos
      - Asociaciones formadas o canceladas
    - Hechos relevantes: se envió un mensaje, se imprimió un informe,...
  - Precondiciones:
    - Cosas necesarias para el éxito de la ejecución del caso de uso que se han probado en algún momento anterior a la ejecución
  - Recomendación:
    - Incluir información necesaria para detalles de diseño: algoritmos, mensajes, interfaces,...

Proceso de transición al diseño



Conexión rápida fases y UML



4.2 UML: diagramas de diseño

Ingeniería del Software Avanzada
Técnicas de análisis y diseño

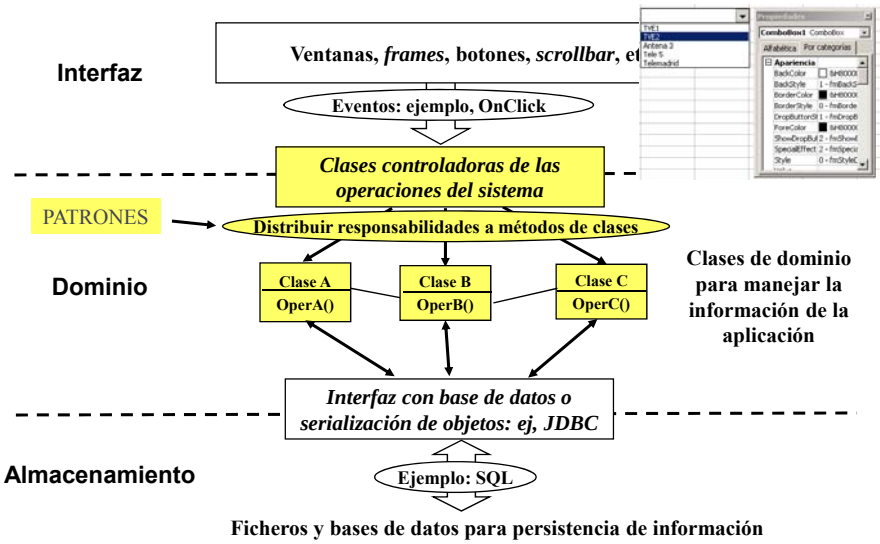
Objetivos

- Comprender la notación de los diagramas de colaboración
- Ser capaces de generar un diagrama de colaboración para cada caso de uso a partir de la información de la fase de análisis
- Conocer los patrones y cómo aplicarlos a los diagramas de colaboración
- Conocer las nociones de visibilidad, dependencias, etc. y aplicar las consecuencias de los diagramas de colaboración a la creación de un diagrama de clases de diseño

Arquitectura

- Asumimos arquitectura de 3 capas:
  - Presentación: ventanas, formularios, etc.
  - Lógica de aplicación o dominio: reglas de negocio
  - Almacenamiento: persistente
- Aislar la lógica en n-capas
  - Objetos de dominio: conceptos del problema
  - Servicios: funciones de interacción de BD, comunicación,...
- Distintas configuraciones Cliente/Servidor
- Ventajas:
  - Aislar la lógica, distribución en varios nodos, asignar tareas de diseño por capas
  - Apoyo en patrón separación modelo-vista

Aplicaciones de tres capas



Patrones

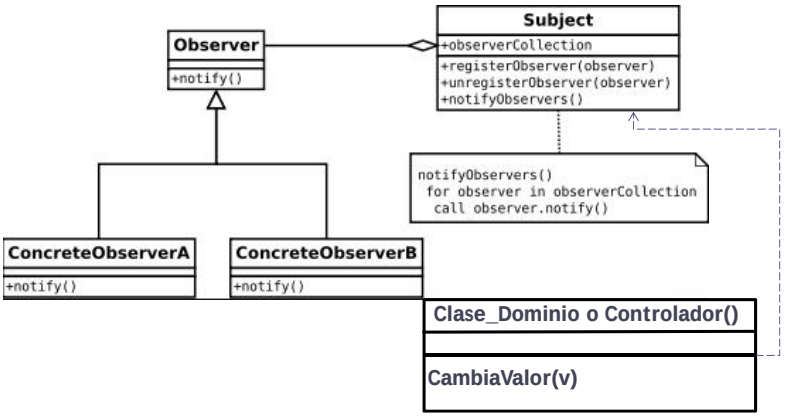
- Principios generales y guías de desarrollo
- Patrón:
  - Descripción de un problema y su solución
  - Nombre propio: facilita la comunicación
  - "Pareja problema/solución con un nombre y aplicable a otros contextos y que sugiere cómo usarlo en situaciones nuevas"
- Los patrones no suelen contener ideas nuevas
  - Solo formalizar la experiencia de desarrollo

Patrón: separación modelo-vista

- Problema:
  - Desacoplar objetos del dominio y de vistas (interfaz GUI), para mayor reutilización y aislar los cambios en interfaz
- Solución:
  - Definir clase de dominio que no tengan acoplamiento ni visibilidad directa con la vista: conservar datos de dominio en clases de aplicación y no en ventanas
    - Clases de dominio: no conocerán existencia de ventanas
    - Clases de presentación (interfaz): tienen acceso a dominio
    - No se enviarán datos ni mensajes a ventanas: sólo dar respuestas
    - En ocasiones (control, simulación), habrá necesidad de comunicación indirecta

Comunicación indirecta

- Recurrir al patrón observador (publicar-suscribir)
- Problema
  - Un cambio de estado (evento) ocurre dentro de un editor de evento y otros objetos están interesados, pero el editor no debería tener conocimiento de suscriptores
- Solución
  - Definir un sistema de notificación indirecta de eventos
    - Las ventanas se suscriben
  - Crear una clase de gestión de eventos
    - No requiere acoplamiento directo
    - Puede transmitirse a cualquier número de suscriptores



El método CambiaValor pide a Subject que ejecute el método notifyObservers
En todos estos método se pasa el valor v
Subject es visible a la clase Dominio o Controlador (flecha de dependencia)

Fase de diseño

- Tres actividades básicas:
  - 1. Crear diagramas de colaboración o secuencia para casos de uso u operaciones
    - A partir del contrato y de diagrama de secuencia del sistema
    - Aplicar patrones GRASP y otros
    - Principalmente para la capa de dominio
  - 2. Crear el modelo de clases de diseño:
    - Actualizando el modelo conceptual con métodos, nuevas clases, especificando visibilidad, etc.
  - 3. Documentar la arquitectura del sistema
    - Diagramas de despliegue y componentes

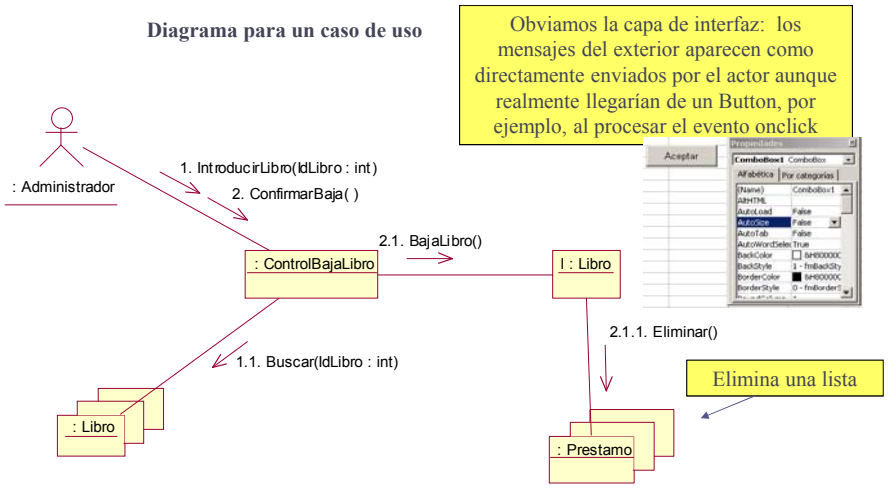
Fases 1 y 2: Concentrarse en capa de dominio

- Capa de interfaz:
  - Viene dada por la interfaz aprobada de cada caso de uso
  - Basada en prototipos enseñados al cliente
  - Usa elementos gráficos (AWT, Swing, etc.): button, fileDialog, textField, etc.
- Capa de almacenamiento:
  - Viene dada por la necesidad de almacenar datos (persistencia) usando base de datos o ficheros (por ejemplo, xml)
  - El modelo conceptual de clases o el modelo E/R deben tener reflejo en esta capa
  - Uso de interfaces: JDBC, etc.

Diagramas de colaboración

- Muestran la interacción entre objetos
  - Incluye las relaciones entre objetos
  - No indica tiempo: sólo números de secuencia
- Como en diagrama de secuencia:
  - Afecta a un subconjunto del modelo de clases:
    - Conjunto de clases y sus relaciones implicados en una acción
    - Marca el contexto de una interacción
- Colaboración:
  - Diagramas de objetos (clases) con mensajes en las asociaciones

Notación (I)



Notación (II)

- Diagrama representa vínculos entre objetos
  - Incluye vínculos temporales
    - Paso de parámetros, variables locales
  - Los vínculos incluyen flecha de navegabilidad
- Mensajes
  - En flecha junto a línea de unión con nombre, parámetros, objeto de retorno y tipos
  - Añaden número de secuencia comenzando en el 1
    - En flujo procedimental, los números se anidan
  - Iteración: * [i:=1..n]; condición: [x > 0] (mismo número de secuencia)
- Aparecen todos los objetos implicados en ejecución
  - Incluso los afectados indirectamente o sólo accedidos

Notación (III)

- Clases y ejemplares:
  - Clase

  - Ejemplar

  - Ejemplar con nombre

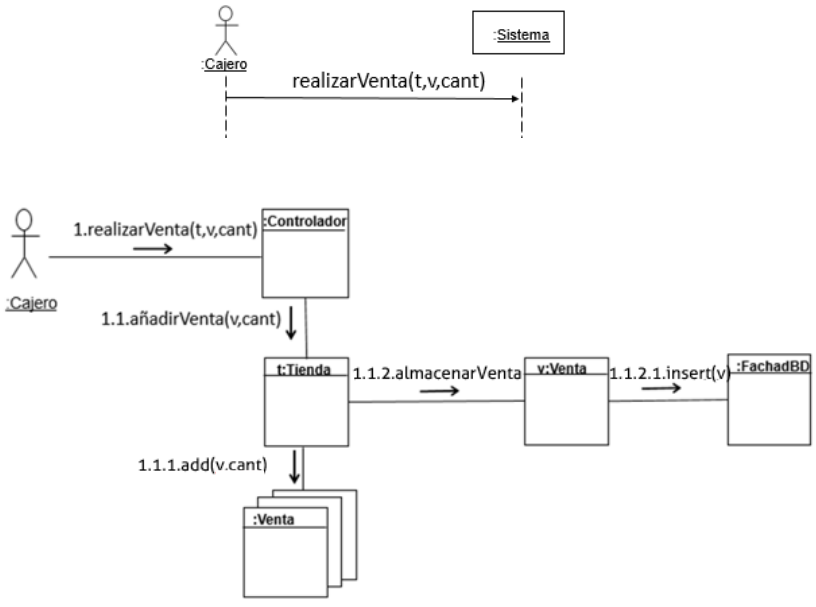
  - Multiobjetos: conjunto (p.ej., *vector*)

- Sintaxis de mensajes:
  - retorno:= mensaje (param: tipo param) : tipo retorno

Crear diag. de colaboración

Información de partida:
•Diagrama de secuencia del sistema; Modelo de clases conceptual; Contrato de operación.

- Pasos:
  - Dibujar el actor (Rational: arrastrar)
  - Incorporar una clase de control específica para el caso
    - Por ejemplo, para "Alta de libro" poner "ControlAltaLibro" o "ControlBibliotecario"
  - Incorporar los mensajes del diagrama de secuencia desde el actor a la clase de control:
  - Asignar respuesta a mensajes a las clases:
    - Analizando en orden de ejecución las acciones a realizar
    - Cuidado con búsquedas y tratamiento de asociaciones
  - Representar acciones como mensajes
  - Revisar diagrama para asegurar cumplimiento de patrones



Patrones GRASP

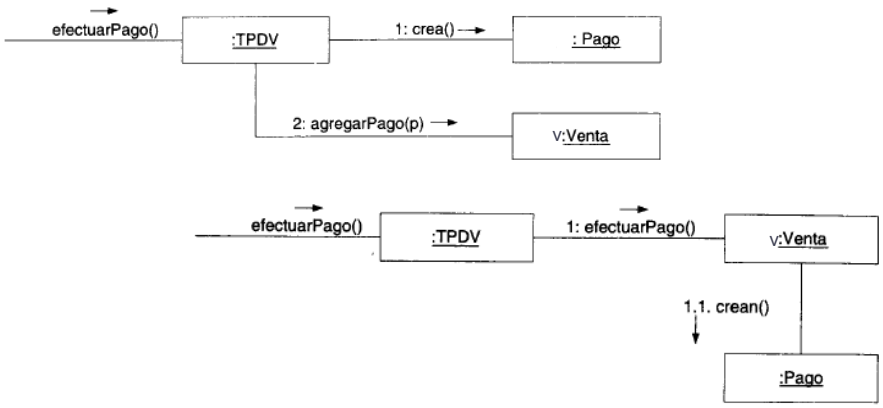
- Muchos patrones
  - Continuamente se proponen más
- Patrones GRASP
  - *General Responsibility Asignment Software Patterns*
  - Propuestos por C.Larman
  - Aplicables a asignación de responsabilidades para clases, al dibujar diagramas de colaboración
- GRASP:
  - Experto, Creador, Gran Cohesión, Bajo Acoplamiento, Controlador
    - Gran cohesión: influye en experto y controlador
    - Bajo acoplamiento: influye en creador y controlador

Metapatrones

- Metapatrones: guías generales de diseño
- Bajo acomplamiento: relación entre clases
  - X tiene atributo de ejemplar de Y o clase Y; X tiene operación que referencia (parámetro) a Y o clase Y; X es subclase directa o indirecta de Y; Y es una interfaz y X lo implementa
- Problemas de acoplamiento:
  - sensible a cambios, menos entendibles, etc.
- Alta cohesión: coherencia de funciones de una clase
  - Ideal: una función por clase
  - Deseable: varias funciones similares (usan mismos datos o hacen cosas similares)

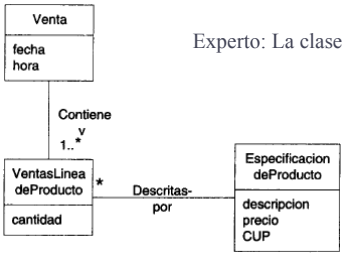
Bajo acoplamiento y Alta cohesión: clases con poca visibilidad y con pocas funciones y relacionadas entre sí

Tenemos que crear una instancia de Pago y asociarla a la Venta. Venta ya tiene visibilidad de Pago por otros casos de uso:
En el primer diagrama TPDV ve dos clases, en el segundo solo una (acoplamiento)
En el primero TPDV realiza dos funciones (crea Pago y lo agrega a Venta), en el segundo solo le pide a Venta que efectué un pago y Venta lo crea agregándoselo (cohesión)



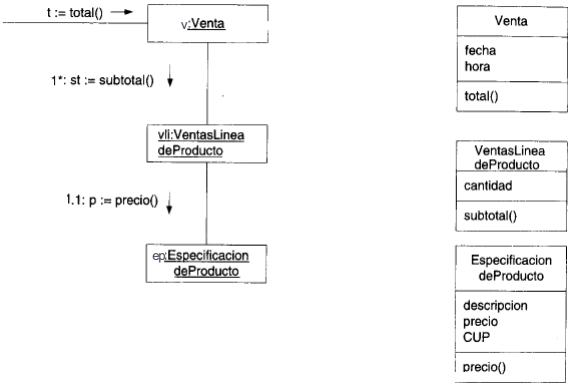
Experto

- Problema:
  - ¿Responsabilidades de manejo de información?
- Solución:
  - Asignar responsabilidad al experto en información
    - La clase que cuenta con información para cumplir responsabilidad
- Explicación:
  - Pueden existir muchos expertos parciales para colaborar
- Beneficios:
  - Bajo acoplamiento y gran cohesión



Experto: La clase que cuenta con información para cumplir la responsabilidad

¿Quién debería calcular el total de una venta? Es necesario conocer las líneas de venta y el precio de cada una. Las líneas las conoce Venta, el precio lo conoce Especificación y el subtotal de una línea, VentasLinea.

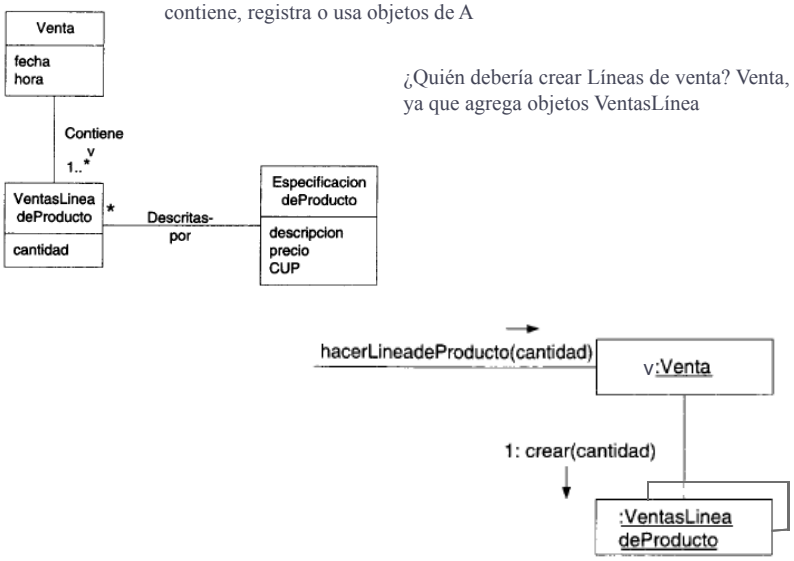


Creador

- Problema:
  - ¿Responsable de crear/borrar un ejemplar de alguna clase?
- Solución:
  - Asignar a B responsabilidad de crear ejemplares de A si:
    - B agrega objetos de A, B contiene objetos de A, B registra ejemplares de A, B usa objetos de A, B tiene datos de inicio de A
- Explicación:
  - Contenedor, Registrador, Agregador...crean contenido, registro, agregado
  - Se aplica también al borrado de objetos
- Beneficios:
  - Bajo acoplamiento: creado es visible ya al creador

Creador: Asignar a B responsabilidad de crear ejemplares de A si B agrega, contiene, registra o usa objetos de A

¿Quién debería crear Líneas de venta? Venta, ya que agrega objetos VentasLínea



Controlador

- Problema:
  - ¿Responsable de manejar eventos del sistema (evento generado por un actor)?
- Solución:
  - Asignar la responsabilidad al "controlador", que será una clase que decide que clases de dominio intervienen para responder al evento
- Explicación:
  - Opciones para determinar el controlador:
    - Que trate todos los eventos del sistema global o de la empresa, solo si hay pocos eventos
    - Que trate todos los eventos generados por un actor
    - Que trate eventos para cada caso (controlador de caso)
- Beneficios:
  - Bajo acoplamiento: independencia interfaz-dominio y Gran cohesión: Los controladores solo se encargan de gestionar eventos y solo lo hacen ellos
  - Problema de controladores saturados: demasiados eventos disminuyen los beneficios anteriores

Controlador: la clase interfaz no ve a las clases de dominio, solo a la clase controlador

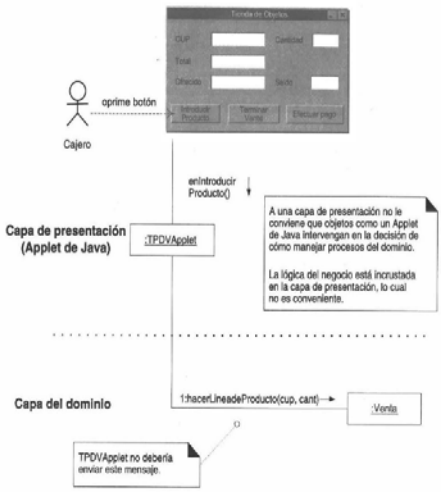


Figura 18.18 Acoplamiento inadecuado de la capa de presentación a la capa del dominio.

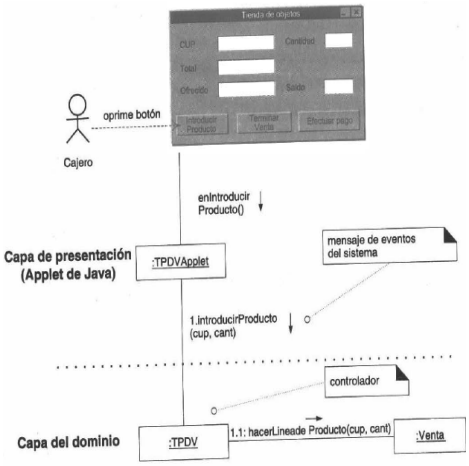


Figura 18.17 Acoplamiento adecuado de la capa de presentación a la capa del dominio.

Diagrama de clases de diseño

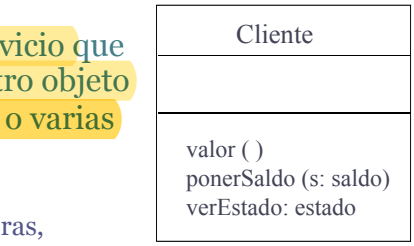
- Exige crear antes:
  - Diagramas de colaboración de diseño: identifican métodos y nuevas clases (por ejemplo: de control, de lista de objetos, etc.)
  - Modelo conceptual: a partir de él, se agregan detalles de definición de clases o incluso nuevas clases
  - Suelen crearse en paralelo a los diagramas de interacción
    - Herramientas como Rational lo hacen
- Deben incluir los detalles para pasar al código:
  - Clases, asociaciones y atributos
  - Interfaces, con operaciones y constantes
  - Métodos
  - Información sobre tipos de atributos
  - Navegabilidad
  - Dependencias

Cómo crear diagrama de diseño

- Identificar nuevas clases a partir de diagramas de interacción
- Dibujar en un diagrama
- Incluir atributos siguiendo el modelo conceptual
- Agregar nombres de métodos del diagrama de colaboración de diseño
- Incorporar tipos a atributos y métodos
- Agregar flechas de navegación (visibilidad de atributo)
- Agregar líneas de dependencias (visibilidad no de atributos: global o declarada localmente)

Métodos en el diagrama de clases de diseño

- Métodos u operaciones
  - Implementación de un servicio que puede ser requerido por otro objeto
  - Una clase puede tener 0, 1 o varias operaciones
  - Especificación:
    - Nombre: si hay varias palabras, iniciales en mayúscula
    - Signatura: nombre, tipo y valores por defecto de todos los parámetros y, en el caso de funciones, un tipo de retorno



Visibilidad

- Capacidad de "ver" de un objeto a otro
  - Alcance o ámbito de objeto
- Cuatro tipos de visibilidad
  - Atributos
  - Parámetros
  - Declarada localmente
  - Global
- Para enviar mensajes a un objeto, debe ser visible
  - Lo visible o accesible para cada clase se debe determinar con los diagramas de colaboración

Visibilidad de atributos

- Visibilidad de atributos de A B:
  - B es atributo de A
  - Relativamente permanente porque persiste mientras existan A y B

```
Public class A
{
...
Private B ObjB;
...
}
```

Visibilidad de parámetros

- Visibilidad de parámetros de A B:
  - B se transmite como parámetro de método de A
  - Relativamente temporal porque persiste sólo dentro del ámbito del método

```
HacerMetodo (B , int c)
{
...
}
```


Visibilidad local

- Visibilidad declarada localmente de A B
  - Se declara que B es objeto local dentro de un método de A
  - Es relativamente temporal porque sólo persiste en el ámbito del método

```
HacerMetodo (int d, int c)
{
B = obtener (d, c)
}
```

Visibilidad global

- Visibilidad global de A B
  - Cuando B es global para A
  - Es relativamente permanente porque persiste mientras existan A y B
  - Medio más obvio:
    - Asignar ejemplar a variable global

Navegabilidad

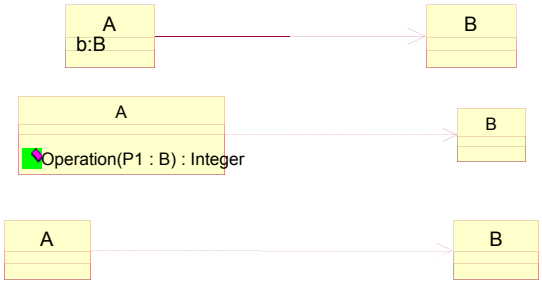
- Navegabilidad: Indica que un elemento usa otro y, por tanto, un cambio en ese elemento puede afectar a otro

- Relación de asociación
  - Relación de asociación con flecha si:
    - A envía mensaje a B
  - Visibilidad de atributos: flecha normal de asociación y navegabilidad

- Relación de dependencia
  - Visibilidad de parámetro, global o declarada local

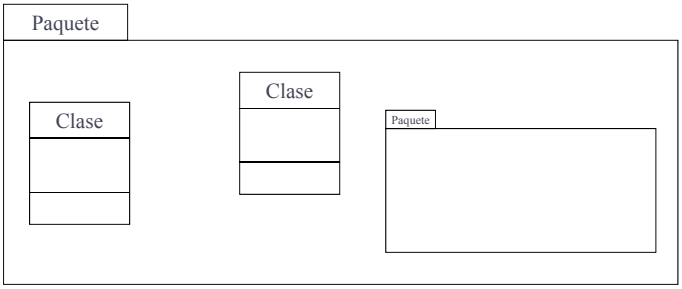
Notación de navegabilidad

- Atributos
- Parámetros
- Local:
  - Metodo 1 (p1: p)
    - b: B



Organizar modelos: paquetes

- Útil para el diagrama de clases de diseño
- Permite grupos de elementos o subsistemas en UML
  - Define en un ámbito de nombres
- Notación

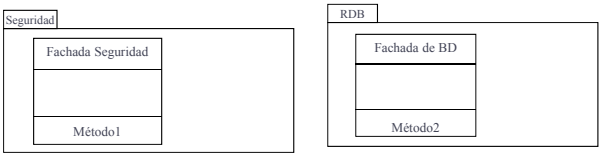


Paquetes y capas

- Formas de utilizar paquetes
  - Aislar cada capa en un paquete: interfaz y dominio
  - También agrupar elementos para un servicio común, por ejemplo clases para cálculos estadísticos y matemáticos
  - Acceso a servicios del paquete: limitar acceso a una o pocas clases (fachada)
- Patrones relacionados:
  - Fachada y Separación modelo-vista

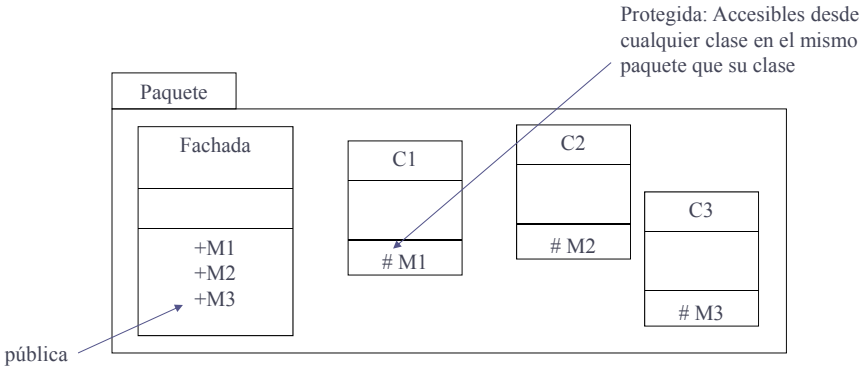
Patrón adicional: fachada

- Problema:
  - Se requiere una interfaz unificada y común para un conjunto heterogéneo de interfaces
- Solución:
  - Definir una clase individual que unifique la interfaz y que se responsabilice de colaborar con clases de otros paquetes y con clases privadas del mismo paquete



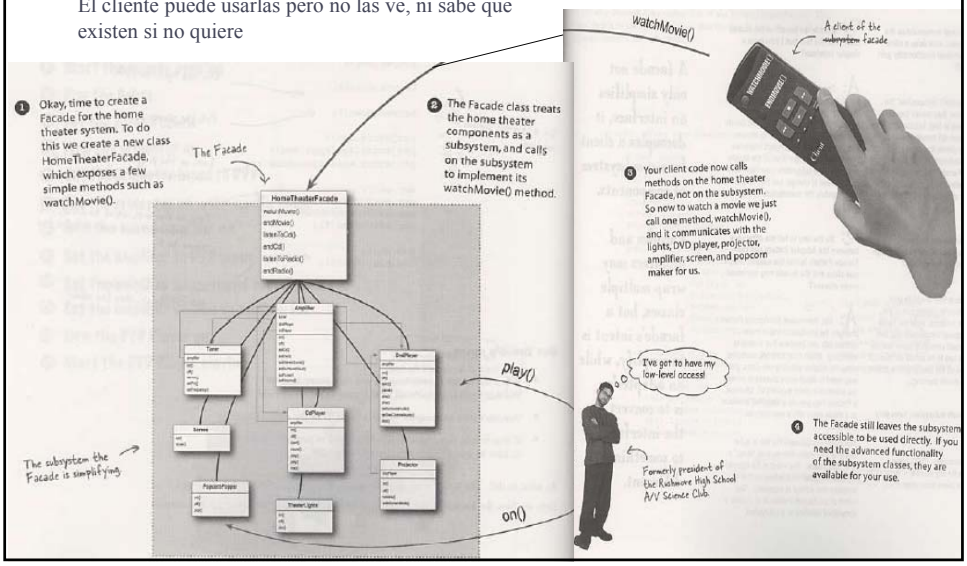
Fachada

- Coordinado con la visibilidad de paquete

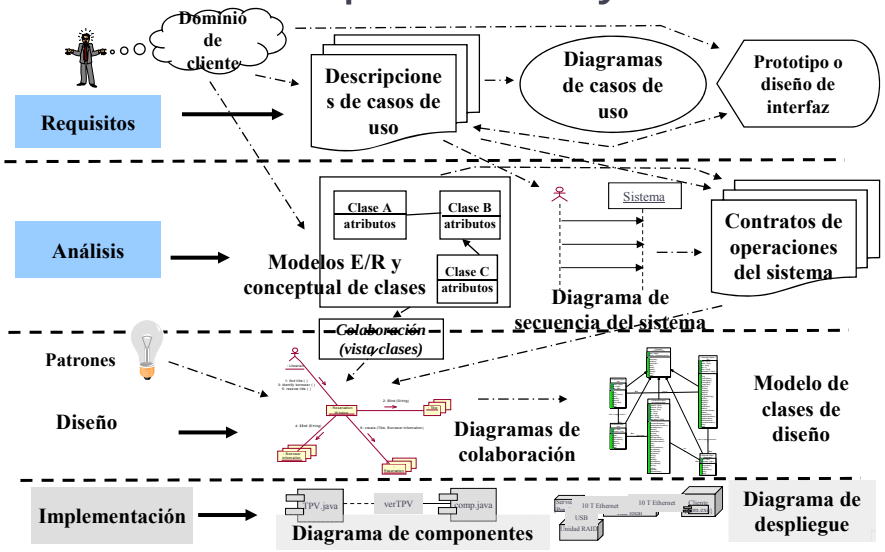


Fachada – Una clase que contiene métodos que combinan o llaman a métodos de otras clases

La clase fachada tiene visibilidad de las otras clases
El cliente puede usarlas pero no las ve, ni sabe que existen si no quiere



Conexión rápida fases y UML



Referencias

- Libros:
  - E. Freeman et al., “Head First Design Patterns”. O'Reilly Media, 2004

4.3 UML: diagramas de implementación

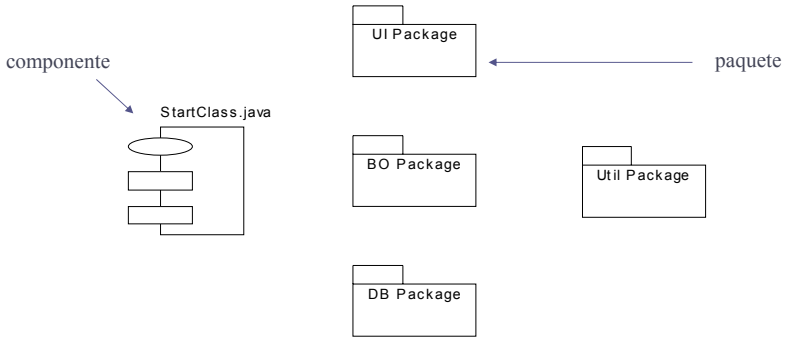
Ingeniería del Software Avanzada
Técnicas de análisis y diseño

Modelos de arquitectura física

- Modelo de componentes:
  - Aspecto de ficheros del sistema
  - Componente:
    - Parte física y reemplazable del sistema que conforma un conjunto de interfaces y proporciona un servicio
- Modelo de despliegue
  - Aspecto físico del sistema (hardware)
  - Nodo:
    - Procesador o dispositivo sobre el que se pueden desplegar los componentes

Modelo de componentes

- Representación final de componentes, ficheros, etc.
- Se pueden organizar en paquetes
- Utilísimo para la gestión de configuración



Gestión de la configuración

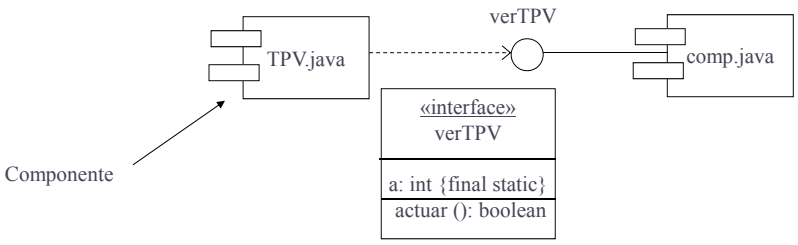
- Arte de identificar, organizar y controlar las modificaciones que sufre el software
- Se basa en los componentes del software
- Gestiona:
  - El modelo de componentes (componentes y sus relaciones)
  - Versiones de componentes (evolución de un componente en el tiempo)
  - Variantes de componentes (Variaciones de una versión de un componente, por ejemplo la misma versión de un componente adaptada a distintos sistemas operativos)

Estereotipos (tipos de componentes)

- File. Archivo físico. Subclases de File:
  - Document. Fichero que no es código fuente o ejecutable
  - Executable. Artefacto que se puede ejecutar en un nodo
  - Library. Fichero de una biblioteca de objetos estática (enlazada en compilación) o dinámica (llamada en ejecución)
- Source. Fichero de código fuente
- Table. Tabla de BD
- Pueden definirse otros según proyecto

Interfaces

- Colección de operaciones para especificar un servicio de un componente
- Notación: VerTPV es una interfaz de comp.java que es utilizada por TPV.java



Ejemplo interfaz

- Conjunto de métodos y atributos que se hacen visibles a otro componente

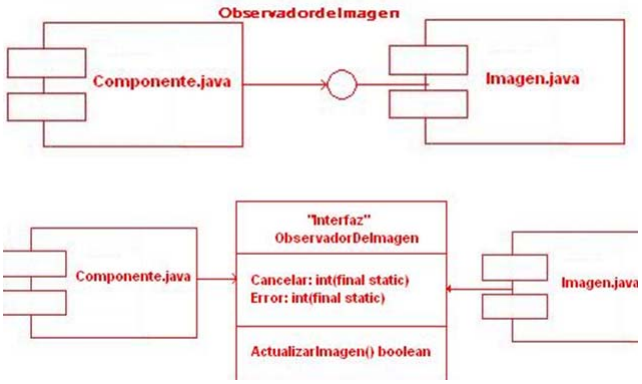
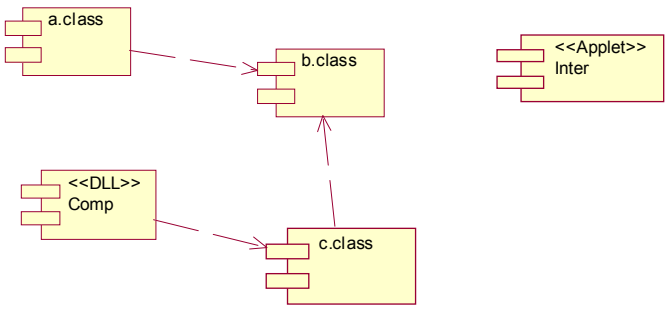


Diagrama de Componentes

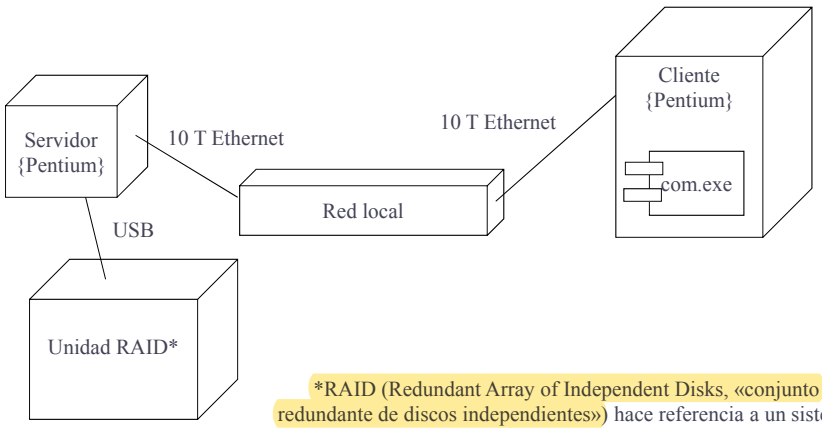


Modelo de despliegue

- Nodo:
  - Elemento físico que representa un recurso de computación
  - Se asignan componentes a nodos
  - Se pueden organizar en paquetes*
  - Incorporar etiquetas de características físicas
- Conexiones:
  - Asociación que representa conexión física entre nodos

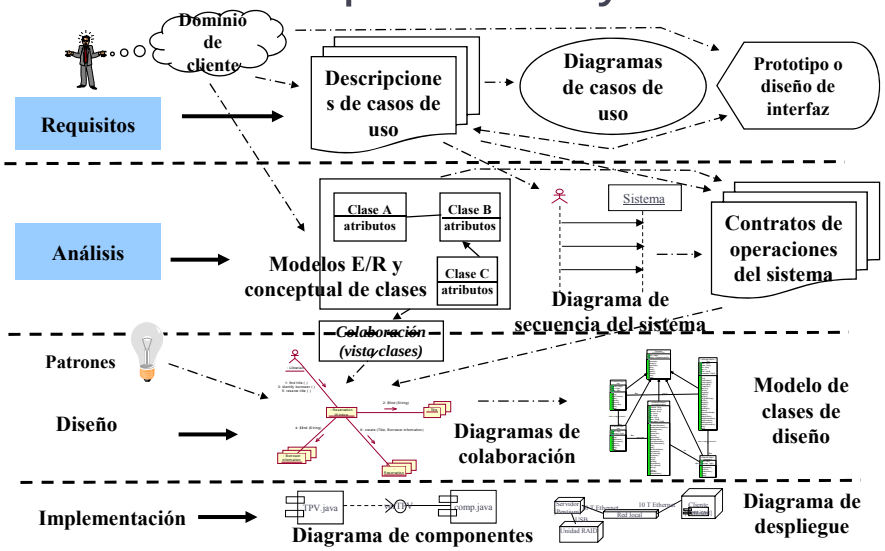
*Igual que las clases (diagrama de clases), los componentes (diagrama de componentes) y los nodos (diagrama de despliegue) se pueden organizar en paquetes (agrupaciones lógicas convenientes a la visión de su utilizador)

Diagrama de despliegue



*RAID (Redundant Array of Independent Disks, «conjunto redundante de discos independientes») hace referencia a un sistema de almacenamiento que usan múltiples discos duros entre los que se distribuyen o replican los datos

Conexión rápida fases y UML



Metodologías

Ingeniería del Software Avanzada
Proceso software

Introducción. Modelos de ciclo de vida

	VENTAJAS	INCONVENIENTES	USO
CASCADA	Gestión simple - Fácil planificación y estimación - Posible revisión al final de cada fase	No hay producto hasta el final - No iteracciones ni paralelismos, poco real - Los errores aparecen tarde y son muy costosos	Sistemas muy claros desde el principio - Requisitos bien definidos - Poca innovación
PROTOTIPADO	Fácil identificación y validación de requisitos - Combina con otros modelos	Tentación de utilizar el prototipo, baja calidad - Prototipo demasiado bueno, mayor coste y esfuerzo - Desilución del usuario	Usuarios poco comunicativos o requisitos poco definidos - Entorno estable
INCREMENTAL	Menos planificación y estimación - Riesgo de implantación menor - Fácil detectar errores - Producto parcial disponible antes	Difícil evaluar plazo y coste final - Necesaria buena gestión de versiones y configuración - Dificil detectar unidades y servicios genéricos de todo el sistema - Peligro de sistema poco consistente	Compromiso del cliente - Sistemas pequeños y sencillos - Sistemas poco claros inicialmente
ITERATIVO	Software modular - Riesgo de implantación menor - Fácil detectar errores - Producto parcial disponible antes	Solo para subsistemas independientes y poco intregrados - Los errores en cada subsistema aparecen tarde	Sistemas fácilmente divisibles - Requisitos bien definidos
ESPIRAL	Permite iteracciones y vuelta atrás - Incorpora objetivos de calidad y análisis de alternativas	Mayor coste y tiempo derivado del análisis de alternativas - Compeljidad de la gestión, necesidad de un buen gestor experto	Sistemas con riesgos, entornos inestables o alta innovación

Concepto

- Conjunto integrado de técnicas y métodos que permiten obtener la forma homogénea y abierta, cada una de las actividades del ciclo de vida del SW
  - Método. Actividades llevadas a cabo para conseguir un objetivo. Ejemplo: método de análisis, de diseño,...
  - Integrado. Significa que las técnicas se utilizan como parte de los métodos.
  - Técnicas. Utilizadas en la aplicación de un método.
  - Homogénea. Sistemática, que se utilice en todos los proyectos de una organización. No debería definirse una nueva metodología para cada proyecto.
  - Abierta. A cambios y adaptación según el proyecto que se va a llevar a cabo.

Que define una metodología

- Una metodología debe definir:
  - Cómo dividir un proyecto en etapas
  - Qué trabajos se llevan a cabo en cada etapa.
  - Que salida se produce en cada etapa, y cuando se debe producir. Las salidas normalmente son documentos y productos SW.
  - Qué técnicas y herramientas se van a utilizar en cada etapa. Normalmente se utilizan técnicas estándar
  - Qué restricciones se aplican (de tiempo, coste, objetivos, etc.)
  - Cómo se gestiona y controla un proyecto

Diferencias con ciclo de vida

- Ciclo de vida es más abstracto:
  - Se refiere sólo al esquema de fases y tareas
  - Es una definición específica de los procesos
  - Suele incluir sólo el Qué hacer
- Una metodología:
  - Puede seguir uno o varios modelos de ciclo de vida
  - Además suele incluir referencias y guías más explícita sobre cómo aplicar métodos, técnicas y herramientas.
  - Incluye el Cómo hacer

Tipos de metodologías

- Clasificación general
  - Por su aplicabilidad:
    - Metodologías oficiales (de iure): Métrica v3 - España, SSADM - Reino Unido
    - Metodologías no oficiales (de facto): Unified process, Yourdon Structured Analysis
  - Por su tecnología:
    - Estructuradas
    - Orientadas a objetos: Orientadas a componentes, Iterativas (RUP)
    - Web
  - Por su gestión del proceso:
    - Pesadas: Métrica V3
    - Ágiles: XP, Scrum
- Metodologías específicas
  - Desarrollo de software seguro

Metodologías estructuradas

- Es la aplicada desde mediados de los años 70.
- Idea:
  - Software se compone de funciones que procesan datos
- Propone crear modelos del sistema que representen de manera descendente los siguientes aspectos:
  - Las funciones (también llamadas "procesos") del sistema
  - Los flujos de datos de entrada, salida e internos de cada función
  - La estructura de los datos procesados por las funciones del sistema.
- Metodologías: Metrica, SSADM, Merise

Técnicas estructuradas

- Análisis:
  - Diagrama de Flujo de Datos
  - Diagrama de Entidad Relación
  - Matriz de funciones-entidades
- Diseño:
  - Diagrama de Estructura Modular (o "de Constantine")
  - Diagrama de Módulos (o "de Jackson")
  - Diagrama de Estructura Lógica de un programa (o "de Warnier"
  - Diagrama de estructura de datos (LDS)
  - Diagrama de tablas (de una Base de Datos)
  - Diagrama de flujo de control (flowchart, organigrama, etc.) y pseudocódigo
- Programación estructurada:
  - Estructuras de control (secuencia, decisión (if), repetición (loop, for, while, repeat,..)
  - Subprogramas: programación modular
  - Lenguajes de programación estructurada: COBOL, FORTRAN, C, PASCAL, NATURAL, PL/SL(Oracle).

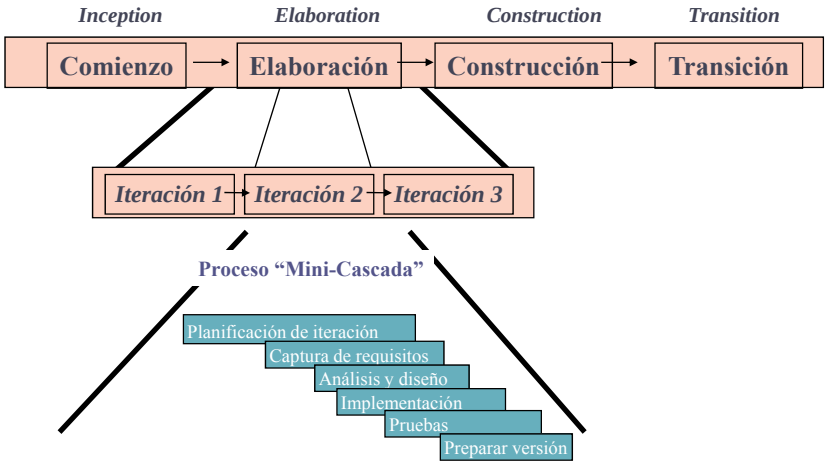
Metodologías Orientadas a Objetos

- Se empieza a aplicar a finales de los años 80.
- Se basa en la idea de que un sistema software se compone de objetos software que interactúan entre sí.
- La funcionalidad de un sistema se reparte entre los objetos, asignando a cada objeto funciones específicas.
- El objetivo final de la ISOO es construir software de la misma forma mediante el ensamblaje de componentes.
- Las ventajas de la orientación a objetos son:
  - Facilidad para la reutilización del software
  - Simplificación del mantenimiento del software
- Metodologías: MÉTRICA 3 (España), UNIFIED PROCESS (Rumbaugh, Booch y Jacobson), FUSION (Hewlett-Packard), OPEN (Henderson-Sellers)

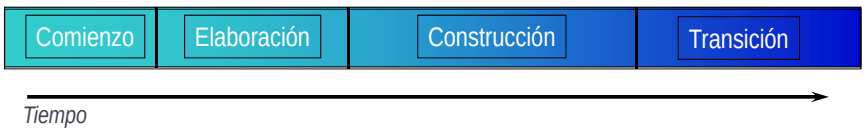
Técnicas OO

- Análisis (www.uml.org) :
  - Diagrama de Casos de Uso
  - Diagrama de Actividades
  - Diagramas de interacción: Secuencia y Colaboración
  - Diagrama de Estados
- Diseño (www.uml.org)
  - Diagrama de Componentes
  - Diagrama de Despliegue
  - Patrones de diseño (independientes de UML)
- Programación OO
  - Herencia: Objetos basados en otros objetos
  - Polimorfismo: Funciones con el mismo nombre
  - Agregación: Objetos que contienen objetos
  - Lenguajes de programación orientada a objetos: JAVA, C++, C#, VISUAL BASIC, SMALLTALK, EIFFEL.

Ejemplo I: Proceso Unificado (RUP)

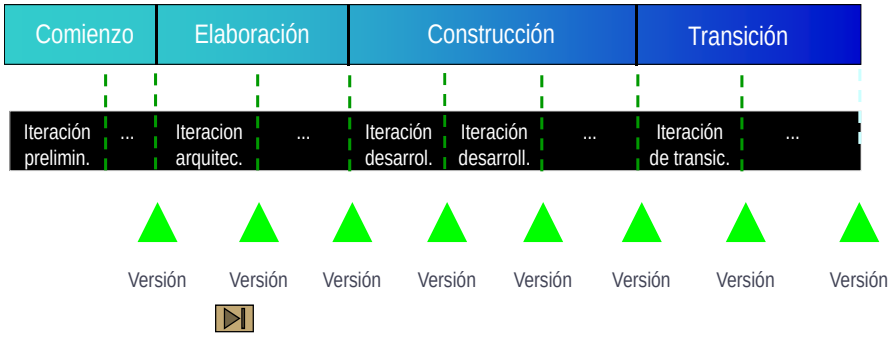


Fases de RUP



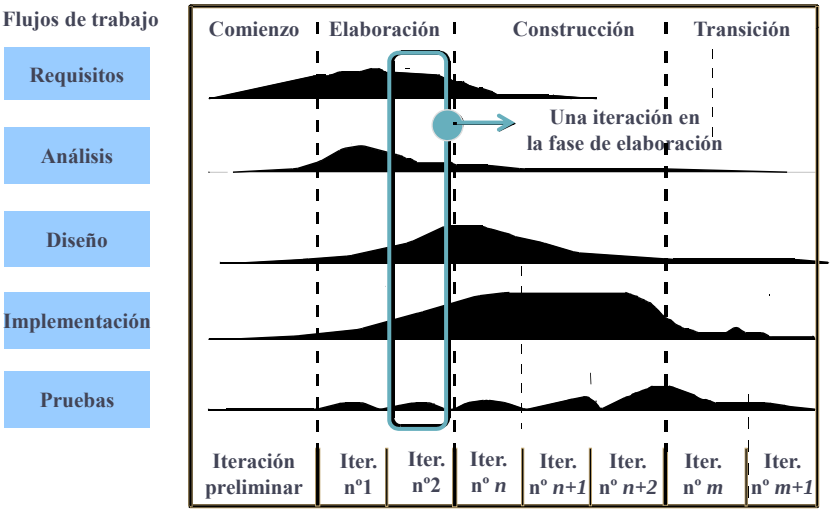
- Comienzo: define el enfoque del proyecto y desarrolla un análisis de negocio (*business case*)
- Elaboración: planear proyecto, especificar características y configuración de arquitectura
- Construcción: programación y prueba
- Transición: traspasar producto a sus usuarios

Fases e iteraciones



Una iteración es una secuencia de actividades con un plan establecido y criterios de evaluación, y que da como resultado una versión ejecutable

Fases



Claves del proceso iterativo

- Integración continua
  - No se hace en un momento cercano a la fecha de entrega
- Versiones ejecutables, frecuentes
  - Algunas internas; algunas para entregar
  - Regularidad: rompe síndrome 90% acabado con 90% por acabar
- Aborda los riesgos a través de una progresión demostrable
  - El progreso se mide en productos, no en documentación o en estimaciones de ingeniería
- Incorpora problemas/elementos/cambios:
  - En futuras iteraciones más que interrupción de producción en marcha

Ejemplo II: Metodologías Orientadas a Componentes

- Proceso para diseño y construcción de sistemas que utilizan componentes de software reutilizables.
- Componente:
  - Una parte reemplazable, casi independiente y no trivial de un sistema que cumple una función clara en el contexto de una arquitectura bien definida
  - Los componentes se identifican por las características de su interfaz (servicios)
- Ing. Soft. Basada en Componentes (ISBC) :
  - ¿Hay componentes comerciales ya desarrollados (COTS) para cada requisito? ¿Se dispone de componentes reutilizables desarrollados internamente? ¿Son compatibles las interfaces de los componentes disponibles dentro de la arquitectura del sistema a construir?

Elementos de apoyo

- Patrones (*patterns*):
  - Existen patrones específicos para mejorar la reutilización y para un diseño y composición apropiada de componentes
- Marcos de trabajo (*frameworks*):
  - Diseño de subsistemas compuesto de colección de clases abstractas o concretas y las interfaces entre ellas.
  - Una aplicación semicompleta con componentes estáticos y dinámicos personalizables para aplicaciones específicas.
  - Un esquema para el desarrollo y/o la implementación de una aplicación

Frameworks

- Diferencias con aplicaciones:
  - Frameworks son soluciones genéricas, incompletos, no cubren toda la funcionalidad sino que abstraen diseño común
- Diferencias con patrones:
  - Frameworks ejecutables e implementados; los patrones son más generales y abstractos, más pequeños y menos especializados
- Diferencias con librerías:
  - Framework no es solo un conjunto de clases (librería), define una estructura de aplicación. "el framework llama a tu código" es distinto a "tu código llama a la librería"

Metodologías Web

- Publicación del estándar IEEE Std 2001-2002:
  - Software Engineering — Recommended Practice for the Internet — Web Site Engineering, Web Site Management, and Web Site Life Cycle
- Se trata de un tipo de ingeniería del software orientada a la naturaleza multidimensional de las aplicaciones web, que implican, además de programación:
  - Definición de estructuras complejas de información (XML, ...)
  - Diseño de estructuras de navegación
  - Gestión de contenidos
  - Diseño gráfico
  - Gestión de seguridad
- Metodologías:
  - OPEN (Henderson-Sellers), HDM (Hypermedia Design Methodology), OOHDM (Object Oriented HDM), RMM (Relationship Management Methodology), EORM (Enhanced Object Relationship Model)

Indicaciones para web

- Inclusión de aspectos importantes para web:
  - Estudio de la navegación y multimedia (tanto interfaz como contenidos)
  - Análisis de grupos de usuarios (universo de usuarios amplio y variado):
    - Accesibilidad
    - Usabilidad (facilidad de uso)
    - Estandarización
  - Documentar bien la implementación. Mantenimiento complejo y con necesidad de respuesta rápida
- Variaciones en los equipos de trabajo:
  - Diseñador web: combina la parte gráfica con la de código para implementar
  - Editor web: responsable de los resultados de web
  - Diseñador gráfico: control y desarrollo de elementos gráficos y multimedia

Cuadro resumen

	INGENIERIA DEL SOFTWARE ESTRUCTURADA	INGENIERIA DEL SOFTWARE ORIENTADA A OBJETOS	INGENIERIA DEL SOFTWARE WEB
Descripción	También denominada "clásica" ó "Ingeniería del software orientada a funciones". • Es la aplicada desde principios de los años 70. • Se basa en la idea de que un sistema software se compone de funciones que procesan datos. (Existen unas funciones generales y otras funciones específicas en las que se descomponen las primeras). • La Ingeniería del Software estructurada considera que el desarrollo de software ha de hacerse de forma descendente (top-down: desde una visión general cercana al usuario, hasta un nivel de abstracción mas detallado, cercano al programador), proponiendo la creación de modelos del sistema que representen de manera descendiente los siguientes aspectos: ○ Las funciones (también llamadas "procesos") llevadas a cabo por el sistema. ○ Los flujos de datos de entrada, salida e internos de cada función del sistema. ○ La estructura de los datos procesados por las funciones del sistema.	• Se empieza a aplicar a finales de los años 80. • Se basa en la idea de que un sistema software se compone de objetos software que interactúan entre sí. • La funcionalidad de un sistema se reparte entre los objetos, asignando a cada objeto funciones específicas. • El objetivo final de la Ingeniería del Software Orientado a Objetos es construir software de la misma forma como se construye el hardware: mediante el ensamblaje de componentes. • Las ventajas de la orientación a objetos son: ○ Facilidad para la reutilización del software. ○ Simplificación del mantenimiento del software. ○ Mejora de la calidad del software.	También denominada "WEB ENGINEERING". • Se trata de un tipo de ingeniería del software orientada a la naturaleza multidimensional de las aplicaciones web que implican, además de programación: ○ Definición de estructuras complejas de información (XML, DTD, ...). ○ Diseño de estructuras de navegación. ○ Gestión de contenidos. ○ Diseño gráfico. ○ Gestión de seguridad. ○ Gestión de diferentes perfiles de usuario.
Técnicas de Análisis	○ Diagrama de Flujo de Datos. ○ Diagrama de Entidad Relación. ○ Diagrama de historia de la vida de una entidad. ○ Matriz de funciones-entidades.	○ Diagrama de Casos de Uso. ○ Diagrama de Actividades. ○ Diagrama de Secuencia. ○ Diagrama de Colaboración. ○ Diagrama de Estados.	○ Diagrama de Entidad Relación. ○ Diagrama de Slices (Entidad Relación extendido). ○ Diagrama de clases.
Técnicas de Diseño	○ Diagrama de Estructura Modular (o "de Constantine"). ○ Diagrama de Módulos (o "de Jackson"). ○ Diagrama de Estructura Lógica de un programa (o "de Warnier"). ○ Diagrama de Árbol Programático (o "de Bertini"). ○ Diagrama de estructura de datos. ○ Diagrama de tablas (de una Base de Datos). ○ Diagrama de flujo de control (flowchart, organigrama, ordinograma). ○ Pseudocódigo.	○ Patrones de diseño. ○ Diagrama de Componentes. ○ Diagrama de Despliegue. ○	○ Diagrama de navegación.
Técnicas de Programación	○ Estructuras de control (secuencia, decisión (if), repetición (loop, for, while, repeat,..). ○ Subprogramas (Que permiten un tipo de programación estructurada conocida como programación modular ("un tipo de programación estructurada con una estructura conocida como subprograma"). ○ Lenguajes de programación estructurada: COBOL, FORTRAN, C, PASCAL, NATURAL, PL/SL(Oracle).	○ Herencia: Objetos basados en otros objetos. ○ Polimorfismo: Funciones con el mismo nombre. ○ Agregación: Objetos que contienen objetos. ○ Lenguajes de programación orientada a objetos: JAVA, C++, C#, VISUAL BASIC, SMALLTALK, EIFFEL.	○ Lenguajes de marcado (HTML, XML, ...). ○ Lenguajes de script (JavaScript, VBScript). ○ Programación CGI utilizando cualquier lenguaje (C, C++, Java, ..). ○ Lenguajes de programación: estructurados y orientados a objetos (XML, Perl, Java, C, C++, PL/SQL,...
Metodologías	• MERISE (Francia). • SSADM (Reino Unido). • MÉTRICA (España). • Otras...	• MÉTRICA 3 (España). • UNIFIED PROCESS (Rumbaugh, Booch y Jacobson). • FUSION (Hewlett-Packard). • OPEN (Henderson-Sellers). • Otras...	• OPEN (Henderson-Sellers). • HDM (Hypermedia Design Methodology). • OOHDM (Object Oriented HDM). • RMM (Relationship Management Methodology). • EORM (Enhanced Object Relationship Model). • Otras...

Metodologías pesadas

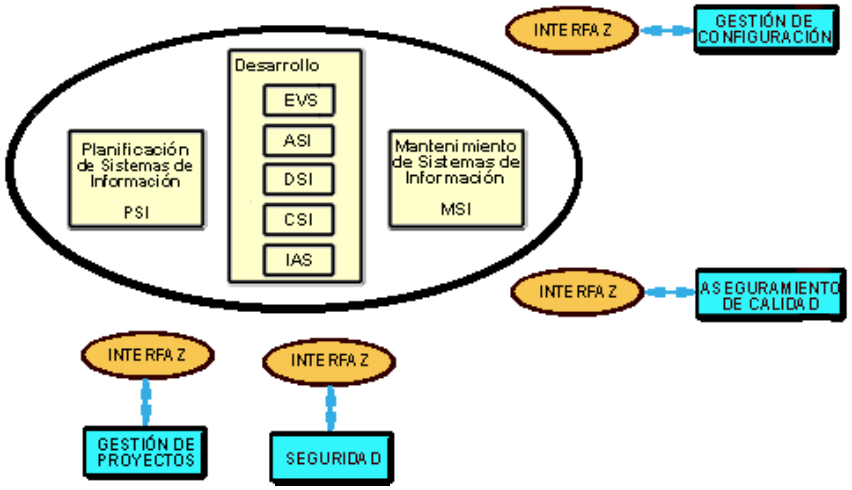
- Se centran en planificar y documentar cada tarea
- Mucho esfuerzo de gestión
- Rígidas respecto a cambios en los requisitos durante el desarrollo
- Cada tarea lleva asociados entregables y responsables de distintas acciones
- Predisposición a contratos, estándares y auditorias
- Facilitan el mantenimiento (intentan minimizarlo definiendo requisitos y arquitecturas con visión de futuro)

Ejemplo: Métrica V3

Es una única estructura la metodología MÉTRICA Versión 3 cubre desarrollo estructurado y orientado a objetos, y facilita a través de interfaces la realización de los procesos de apoyo u organizativos. Los procesos se dividen en Principales e Interfaces.

Cada Proceso detalla las Actividades y Tareas a realizar. Para cada tarea se indican:

- Las técnicas y prácticas a utilizar
- Los responsables de realizarla
- Sus productos de entrada y salida



Metodologías ágiles

- Manifiesto ágil:
  - Varios críticos (K.Beck) de las metodologías tradicionales publican en 2001 este manifiesto de nuevos valores para el desarrollo de software
- Contenido (http://www.agilemanifesto.org/)
  - Descubrimos mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar más el lado izquierdo aunque hay valor en elementos de la derecha

Ágil	Tradicional
Los individuos y su interacción	Los procesos y las herramientas
El software que funciona	La documentación exhaustiva
La colaboración con el cliente	La negociación contractual
La respuesta al cambio	El seguimiento de un plan

Principios

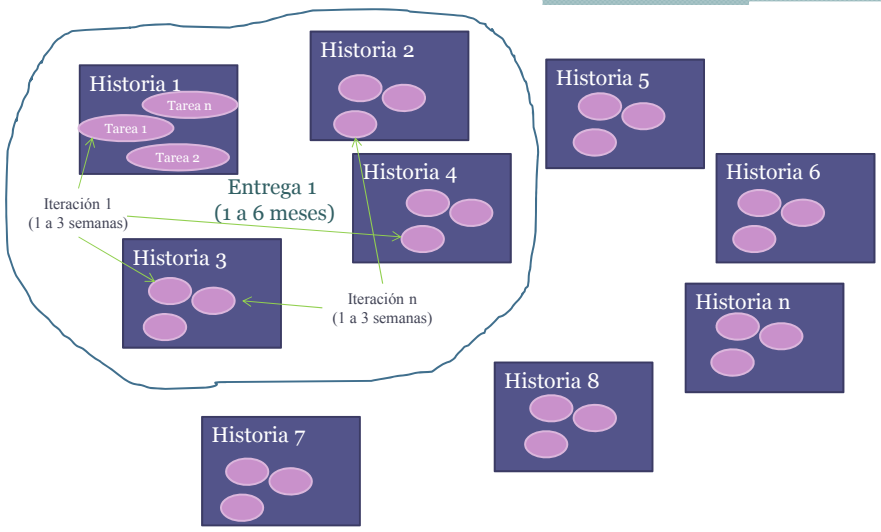
1. La prioridad principal es satisfacer al cliente mediante tempranas y continuas entregas de software que le den valor
2. Dar la bienvenida a los cambios. Los métodos ágiles capturan los cambios para que el cliente tenga una ventaja competitiva
3. Entregar frecuentemente software que funcione, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre una entrega y la siguiente
4. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto
5. Construir proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos
6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo
7. A intervalos regulares, el equipo reflexiona cómo llegar a ser más efectivo, y según esto ajusta su comportamiento

Principales métodos ágiles

- Crystal Methodologies, Alistair Cockburn:
  - https://en.wikiversity.org/wiki/Crystal_Methods
- SCRUM, Ken Schwaber & Jeff Sutherland:
  - www.controlchaos.com
- DSDM (Dynamic Systems Development Method):
  - https://www.agilebusiness.org/resources/dsdm-handbooks/the-dsdm-agile-project-framework-2014-onwards
- Lean Programming, Mary Poppendieck:
  - www.poppendieck.com
- Extreme Programming, Kent Beck:
  - www.extremeprogramming.org,
- Adaptative Software Development, Jim Highsmith:
  - www.adaptivesd.com

Ejemplo I: XP

- Historias del Usuario (*User-Stories*). Muy similares a los escenarios de casos de uso.
  - Establecen los requisitos del cliente
  - Trozos de funcionalidad que aportan valor
  - Se les asignan tareas de programación con un nº de horas de desarrollo
  - Las establece el cliente
  - Son la base para las pruebas funcionales
- Planificación por entregas al cliente (*releases*)
  - Se priorizan aquellas *user-stories* que el cliente selecciona porque son más importantes para el negocio
  - Entregas:
    - Son lo más pequeñas posibles
    - Se dividen en iteraciones (iteración = 2 o 3 semanas)
    - Están compuestas por historias
  - A cada programador se le asigna una tarea de la *user-story*



Entrega = puesta en producción
Iteración= supera pruebas de aceptación, pero no puesta en producción

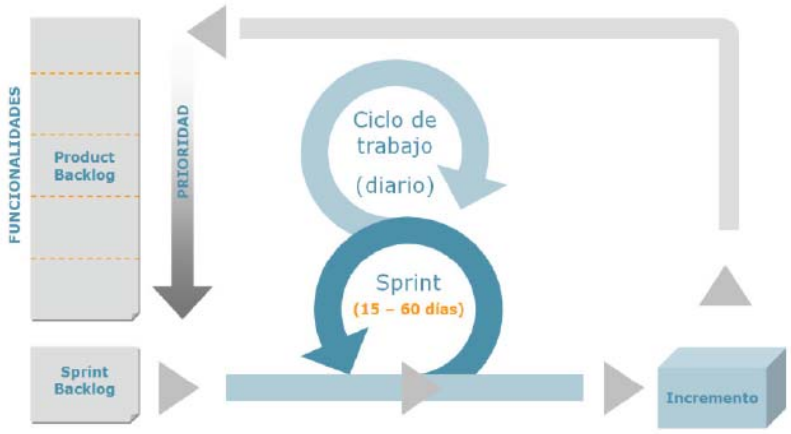
Roles en XP

- Programador (*Programmer*)
  - Responsable de decisiones técnicas
  - Responsable de construir el sistema
  - Sin distinción entre analistas, diseñadores o codificadores
  - En XP, los programadores diseñan, programan y realizan las pruebas
- Encargado de Pruebas (*Tester*)
  - Ayuda al cliente con las pruebas funcionales
  - Se asegura de que las pruebas funcionales se superan
- Rastreador (*Tracker*)
  - Metric Man
  - Observa sin molestar
  - Conserva datos históricos
- Jefe de Proyecto (*Manager*)
  - Organiza y guía las reuniones
  - Asegura condiciones adecuadas para el proyecto
- Cliente (*Customer*)
  - Parte permanente del equipo
  - Determina qué construir y cuándo
  - Establece pruebas funcionales
- Entrenador (*Coach*)
  - Responsable del proceso
  - Tiende a estar en un segundo plano a medida que el equipo madura

Prácticas XP

- El cliente debe estar permanentemente con el quipo
  - Decisiones del cliente: prioridad, composición y fecha de las entregas
  - Decisiones técnicas (programadores y otros): estimaciones de tiempo, organización del proceso de trabajo, planificación detallada (dentro de una entrega)
- Reunión diaria de todo el equipo, de pie, en circulo
- El código es de todos, por tanto es imprescindible:
  - Estándares y normas de estilo de programación
- Todo el código se escribe en parejas: uno programa, otro revisa y asesora y a la inversa (*pair-programming*)
- Pruebas (se diseñan antes de codificar): unitarias e integración, el programador (mejor automatizadas); funcionales, el cliente
- Existe un ordenador para la integración

Ejemplo II: SCRUM



SCRUM. Definiciones

- **Sprint**
  - Intervalo de tiempo prefijado durante el cual se crea un incremento entregable de producto
- **Product backlog.** Requisitos del sistema.
  - Inventario de características y funciones que el propietario del producto desea obtener, ordenado por prioridad.
  - Es un documento "vivo", en constante evolución.
- **Sprint Backlog.**
  - Lista de los trabajos que realizará el equipo durante el sprint para generar el incremento previsto.

SCRUM. Reuniones

Reunión diaria. Cada miembro responde a tres preguntas:
1. Trabajo realizado ayer
2. Trabajo que se va a realizar hoy
3. Impedimentos para desarrollar el trabajo

→ SEGUIMIENTO DEL SPRINT

Presentación de lo realizado. Análisis y revisión del incremento generado

→ REVISIÓN DEL SPRINT

Se genera la "sprint backlog" o lista de tareas que se van a realizar, y el "objetivo del sprint": lema que define la finalidad de negocio que se va a lograr.

→ PLANIFICACIÓN DEL SPRINT

SCRUM. Roles

- Propietario del producto.
  - Una persona, y sólo una, conocedora del entorno de negocio del cliente y de la visión del producto
  - Representa a todos los interesados en el producto final y es el responsable del Product Backlog
- Responsabilidad del desarrollo: El equipo
  - Multidisciplinar que cubre todas las habilidades
  - Se auto-gestiona y auto-organiza
- Scrum Manager
  - Responsabilidad de funcionamiento del modelo
  - A nivel de proyecto o a nivel de organización
  - En algunos casos un rol exclusivo (Scrum Master) y en otros, varias personas (responsables de calidad o procesos o de gestión de proyectos…)

Métodos ágiles. Saber más…

- Ventajas y limitaciones:
  - Beneficios en equipos y aplicaciones pequeñas que requieren muchos cambios y dinámica
  - Problemas de gestión en aplicaciones más grandes
  - Buenos resultados necesitan muy buenos profesionales
- Bibliografía:
  - B.Meyer. *Agile! The Good, the Hype ande the Ugly*. Springer, 2014

Security Software Development Life Cycle (S-SDLC) - I

- Conjunto de principios y buenas prácticas a llevar a cabo durante el desarrollo del ciclo de vida del software para detectar, prevenir y corregir defectos de seguridad.
- Microsoft Security Development Lifecycle (Microsoft SDL)
  - Contiene una colección de actividades de seguridad agrupada por las fases del SDLC tradicional.
- Seven Touchpoints for Software Security
  - Conjunto de mejores prácticas. Han sido adoptadas por el Departamento de Seguridad Nacional de EEUU y por Erns and Young entre otros.

Security Software Development Life Cycle (S-SDLC) - II

- Oracle Software Security Assurance (OSSA)
  - Metodología de Oracle para abordar la seguridad en las fases de diseño, codificación, pruebas y mantenimiento para productos Oracle y su nube.
- NIST 800-64-Seguridad en el ciclo de vida de desarrollo software
  - Se describen las funciones y responsabilidades clave de seguridad en cada fase de su ciclo de vida.
  - No es una metodología flexible, ya que se proyecta principalmente para organizaciones gubernamentales.

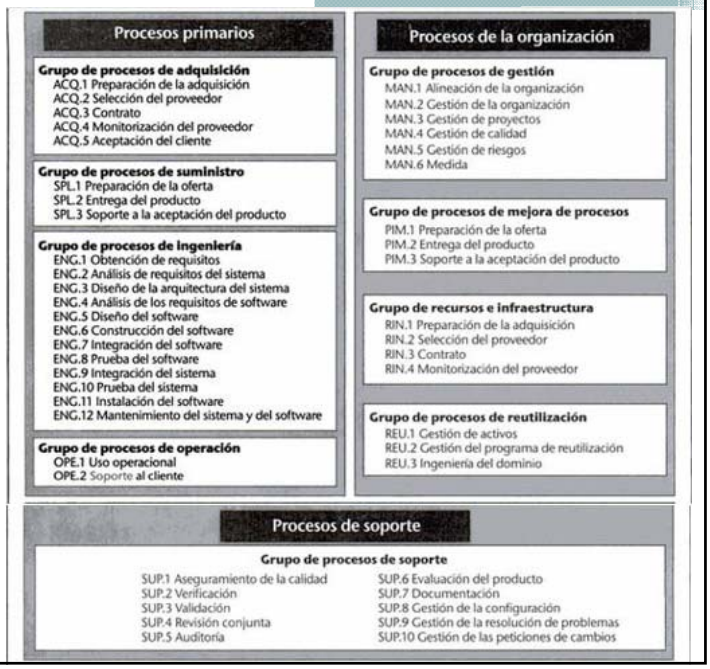
Mejora de procesos

Ingeniería del Software Avanzada
Proceso software

Estándares de proceso software

- No obligan a un tipo de ciclo de vida ni metodología
- Son un marco de referencia: presentan procesos (objetivos, participantes, etc.)
- No indican cómo realizar las actividades
- ISO 12207: Software Life Cycle Processes

ISO 12207



Modelos de madurez de proceso

- Proporcionan referencia para evaluar madurez o calidad de proceso:
  - CMMi. Capability Maturity Model
  - ISO 15504 ("Spice": Software Process Improvement and Capability dEtermination)
  - ISO 9000-3: Aplicación de norma ISO 9000 al desarrollo y mantenimiento de software

CMM

- Surgió por petición del DoD de EE.UU.:
  - Evaluación eficaz de contratistas según sus métodos de trabajo para prevenir problemas en proyectos
  - Creación del Software Engineering Institute en la Universidad Carnegie-Mellon
    - Documentación gratuita en www.sei.cmu.edu
  - Primeras versiones en 1987 pero lanzamiento de CMM en 1991-92.
- El modelo permitía valorar la madurez de proceso de una organización en una escala de 5 niveles
  - A mayor nivel, se demostraba mayor productividad, menor riesgo de proyecto y más calidad
  - Se asume que mejor proceso lleva a mejor producto
  - Primeras evaluaciones demostraron bajo nivel de empresas contratistas de defensa

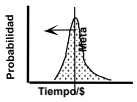
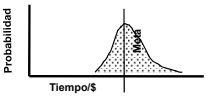
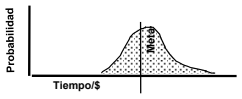
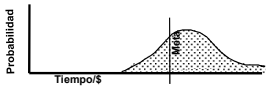
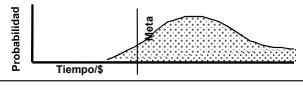
CMMi

- CMMi surge para mejorar CMM integrando ideas de ISO 9000 y otros modelos
  - Se generan especializaciones del modelo, por ejemplo para desarrollo de software CMMI-DEV
  - Define áreas de proceso o KPA (key process áreas), parecidas a los procesos de ISO 12207
  - Dentro de cada KPA define actividades y prácticas clave a evaluar
  - Permite dos esquemas: continuo y por etapas
    - Por etapas: se pasa al siguiente una vez superado el anterior
    - Continuo: permite trabajar en una sola área independiente del nivel de otras, por lo que áreas diferentes pueden tener niveles diferentes

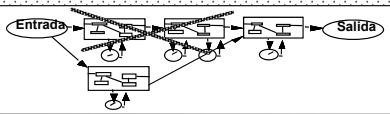
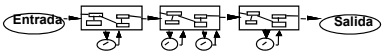
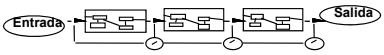
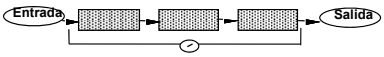
Niveles CMMi

- 5 niveles principales con áreas de proceso en cada uno en orden de madurez:
  - Inicial o caótico: proceso indefinido, codificar y probar sin ingeniería, resultados impredecibles
  - Repetible/gestionado: repetición de prácticas de gestión e ingeniería, con ciertos mecanismos de control pero con demasiada variabilidad de resultados
  - Definido: actividades de ingeniería y gestión definidas y documentadas con métodos conocidos, previsiones fiables y calidad y proceso controlados (al menos, cualitativamente)
  - Gestionado cuantitativamente: objetivos y mediciones cuantitativas en todos los productos y actividades, control cuantitativo
  - De mejora continua: mejora continua de proceso con incorporación de tecnología, control total y optimización

Capacidad de Planificación

Nivel	Características Proceso	Estimación & Realidad
5 Optimizado	Se institucionaliza la mejora del proceso	
4 Gestionado	El producto y el proceso se controlan cuantitativamente	
3 Definido	Las prácticas técnicas se integran con las prácticas de gestión y se institucionalizan	
2 Repetible	Se institucionalizan las prácticas de gestión del proyecto	
1 Inicial	El proceso es informal y ad hoc	

Visibilidad de la Dirección

Nivel	Características Proceso	Visibilidad del Proceso
5 Optimizado	Se institucionaliza la mejora del proceso	
4 Gestionado	El producto y el proceso se controlan cuantitativamente	
3 Definido	Las prácticas técnicas se integran con las prácticas de gestión y se institucionalizan	
2 Repetible	Se institucionalizan las prácticas de gestión del proyecto	
1 Inicial	El proceso es informal y ad hoc	

Situación actual

empresas PYME de modelos de certificación en determinadas materias tendrán la forma de subvención, por una cantidad alzada según tipo de certificado de acuerdo con la tabla siguiente:

Actualmente no vigente

Tipo de certificación	Importe de la ayuda
CMMI o SPICE Nivel 5.	37.000 euros
CMMI o SPICE Nivel 4.	32.000 euros
CMMI o SPICE Nivel 3.	27.000 euros
CMMI o SPICE Nivel 2.	22.000 euros
UNE-ISO/IEC 20000-1:2007.	13.000 euros
UNE-ISO/IEC 27001:2007.	10.000 euros

- España:
  - En 2018, 11 empresas certificadas con nivel 5 en CMMI-DEV
  - Subvenciones de MITYC en 2011 (tabla)
- Mundial:
  - Informe 2018:
    - https://cmmiinstitute.com/getattachment/b9ce903c-92af-4e9f-a339-4600459556c0/attachment.aspx
    - Inicial (0,7%), Repetible (15%), Definido (74%), Gestionado (2,3%), Optimizado(8%)
  - Resultados actualizados: https://sas.cmmiinstitute.com/pars/pars.aspx

ISO 15504

- ISO es una federación mundial de organismos nacionales de normalización
  - En España es AENOR, que puede lanzar normas nacionales
  - Las normas internacionales se preparan en los Comités Técnicos (ISO/TC), en los que participan organizaciones públicas y privadas
  - La publicación como Norma Internacional requiere la aprobación de, al menos, el 75% de los miembros requeridos a votar
- Conocido como SPICE (Sofwtare Process Improvement and Capability Determination)
  - Promovido por el European Software Institute desde 1993
  - Se basa en CMMi e ISO 12207 (y otros modelos, incluso ISO 9000) para proponer un estándar mundial
- Comparando como CMMi:
  - Incluye nivel previo (nivel 0)
  - Evalúa una serie de aspectos particulares (atributos) de cada proceso
  - Obtiene un nivel para cada proceso

Comparación categorías de procesos

Los procesos se clasifican por categorías

Categorías de Áreas de proceso CMMI	Categorías de procesos ISO 15504
Gestión de proceso	Organización
Gestión de proyecto	Gestión
Ingeniería	Ingeniería
Soporte	Soporte
	Cliente-Proveedor

Ejemplo: Para nivel 2 de CMMI existen 7 Áreas de Proceso:
REQM Gestión de Requisitos; PP Planificación de Proyectos; PMC Seguimiento y Control de Proyectos; SAM Acuerdos con Proveedores; MA Medición y Análisis; PPQA Aseguramiento de la Calidad en el Proceso y el Producto; CM Gestión de configuraciones

¿Qué es ISO 9000?

- Familia de Normas Internacionales sobre las buenas prácticas de Sistemas de Gestión de la Calidad
  - Objetivo:
    - Aseguramiento de la calidad del producto
    - Aumento de la satisfacción del cliente
  - Aplicable a cualquier empresa (independiente del tamaño) y producto en cualquier sector de actividad
  - Establece lo que hace la empresa para gestionar sus procesos
- Establece QUÉ requisitos debe cumplir el sistema de calidad, pero NO establece CÓMO deben cumplirse

ISO 9001 e ISO 9000-3

- La serie de normas ISO 9000 para sistemas de gestión de calidad se iniciaron en 1987
  - Modelos genéricos para cualquier sector
  - ISO 9001: aplicable a actividad de software
  - ISO 9000-3: amplia la parte de diseño de ISO 9001
- Muy conocidas para obtener una certificación de calidad
  - No hay niveles, muy centrado en registro de evidencias
  - Auditoría, recertificación cada 3 años
- Equivalencia
  - Un enfoque distinto al adaptar normas genéricas de calidad para certificar organizaciones con un único umbral y sin valoración de escala
  - Nivel 3 de CMMi o todos los procesos nivel 3 de ISO 15504 debería obtener fácilmente certificación ISO 9001
  - Al revés no se puede asegurar

Sistema de Gestión de la Calidad (SGC)

- Define y establece la política y objetivos de calidad de una empresa
- Permite documentar e implementar los procedimientos necesarios para alcanzar los objetivos
- Un SGC implementado adecuadamente asegura:
  - Los procedimientos se realizan consistentemente
  - Los problemas se identifican y resuelven
  - La empresa revisa y mejora continuamente sus procedimientos, productos y servicios
- Suele concretarse documentalmente en un Manual de Calidad con sus procedimientos asociados

Beneficios de un SGC ISO 9000

- Externos
  - Acceso a los mercados mundiales (ISO 9000 estándar nacional de muchos países)
  - Necesidad de aseguramiento de la calidad de los productos que se compran (suministradores registrados)
  - Estímulo competitivo (distinguirse de no registrados)
- Internos
  - Incremento de satisfacción del cliente
  - Pocas decisiones de procedimiento: la norma ISO 9000 establece los requisitos que debe poseer un SGC

Otros modelos de madurez: Seguridad

- System Security Engineering Capability Maturity Model (SSE-CMM)
  - Derivado del CMM. Describe las características esenciales de los procesos que deben existir en una organización para asegurar una buena seguridad de sistemas.
- ISM3 - Information Security Management Maturity Model
  - Estándar que define los procesos de seguridad para administrar el SGSI (sistema de gestión de la seguridad de la información) de una organización
- BSIMM - Building Security In Maturity Model
  - Construido a partir del estudio de diferentes iniciativas de seguridad de varias empresas.
  - Descriptivo: se observa y se informa sobre ideas y actividades ejecutadas en el desarrollo de software.