Ingeniería de Software II - T

Unidad 4 – Diseño Orientado a Objetos

Lic. Marta Saenz Lopez

DOO

"*DOO es el proceso que modela el dominio de la solución, lo que incluye a las clases semánticas con posibles añadidos, y las clases de interfaz, aplicación y utilidad identificadas durante el diseño*".

(Monarchi y Puhr,1992)

- El objetivo es reexaminar las clases del dominio del problema, refinándolas, extendiéndolas y reorganizándolas, para mejorar su reutilización y tomar ventaja de la herencia.

DOO

- Crea una representación del campo del problema del mundo real y la hace corresponder con el ámbito de la solución que es el software.
- A diferencia con otros métodos de diseño, produce un diseño que interconecta objetos de datos y operaciones de procesamiento para esos objetos, de forma que se modulariza la información y el procesamiento, en lugar de aislar el procesamiento.

DOO (cont.)

- Transforma el modelo de análisis creado usando análisis orientado a objetos en un modelo de diseño, que sirve como anteproyecto para la construcción de software.
- Su naturaleza única, reside en su capacidad para construir cuatro conceptos importantes de diseño de software:
  - abstracción,
  - ocultamiento de información,
  - independencia funcional, y
  - modularidad.
- Todos los métodos de diseño procuran software que exhiba estas características fundamentales, pero sólo el DOO provee un mecanismo que permite al diseñador alcanzar las cuatro, sin complejidad ni compromiso.

DOO (cont.)

- A diferencia de los métodos de diseño de software convencionales, el DOO proporciona un diseño que alcanza diferentes niveles de modularidad.
  - La mayoría de los componentes de un sistema están organizados en subsistemas.
  - Los datos y las operaciones que manipulan los datos se encapsulan en objetos. Contribuyen a la modularidad global.
  - Describe la organización específica de los datos, de los atributos y el detalle procedural de cada operación.

Características

- Los objetos son abstracciones del mundo real o entidades del sistema que se administran ellas mismas.
- Los objetos son independientes y encapsulan estado y representan información.
- La funcionalidad del sistema se expresa en términos de servicios de los objetos.
- Las áreas de datos compartidas se eliminan. Los objetos se comunican a través de paso de mensajes.
- Los objetos pueden estar distribuidos y pueden ser ejecutados secuencialmente o en paralelo.

Ventajas

- Fácil de mantener, los objetos representan entidades auto-contenidas.
- Los objetos son componentes reutilizables.
- Para algunos sistemas, puede haber un mapeo obvio entre las entidades del mundo real y los objetos del sistema.

DOO vs Enfoque Convencional

- Al igual que el diseño convencional de software, el DOO aplica:
  - **diseño de datos** cuando se representan los atributos,
  - **diseño de interfaz** cuando se presenta el intercambio de mensajes, y
  - **diseño a nivel de componentes** (procedimental), cuando se diseñan las operaciones.
- La arquitectura de un diseño OO se centra más en las colaboraciones entre objetos que en el flujo de control de datos.

DOO vs Enfoque Convencional (cont.)

- El DOO difiere del diseño convencional (orientado a procesos) ya que no se realiza un problema en términos de tareas (subrutinas) ni en términos de datos, sino que se analiza el problema como un sistema de objetos que interactúan entre sí.

DOO vs Enfoque Convencional (cont.)

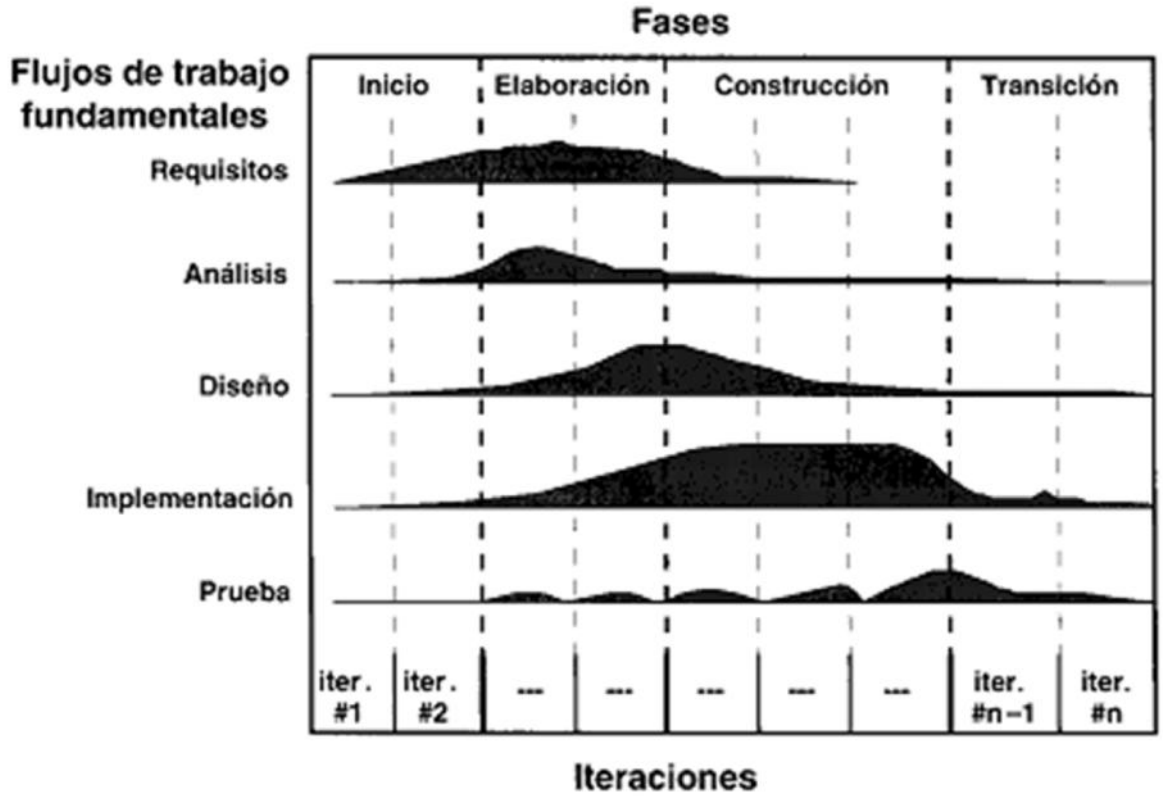
Diseño convencional (Orientado a Procesos)	DOO
Módulos construidos alrededor de las operaciones.	Módulos construidos alrededor de las clases.
Datos globales o distribuidos entre módulos.	Clases escasamente acopladas, sin datos globales.
Entrada/Proceso/Salida.	Encapsulamiento y mensajes.
Organigramas de trabajo y flujo de datos.	Diagramas jerárquicos de clases.

Proceso Unificado

El Proceso Unificado de Desarrollo de Software

Jacobson, Booch, Rumbaugh

Proceso Unificado (PU)



PU (cont.)

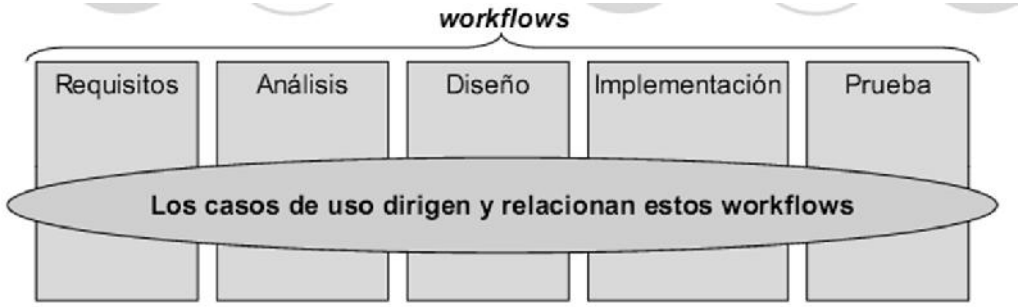
- Es un proceso:
  - Iterativo e incremental.
  - Dirigido por los casos de uso.
  - Centrado en la arquitectura.
    - La arquitectura proporciona la estructura sobre la cual guiar las iteraciones.
    - Los CU definen los objetivos y dirigen el trabajo en cada iteración.

PU (cont.)

- Proceso iterativo e incremental:
  - El proyecto se divide en una serie de partes o mini-proyectos.
  - Cada uno de esos mini-proyectos va a ser una iteración que resulta en un incremento.
  - En cada iteración se trata un conjunto de casos de uso y los riesgos más importantes.
  - Las **iteraciones** hacen referencia a pasos en el flujo de trabajo y los **incrementos**, al crecimiento del producto.

PU (cont.)

- Proceso dirigido por los casos de uso:
  - Los CU capturan requisitos, se especifican (analizan), se diseñan, se implementan y se prueban.

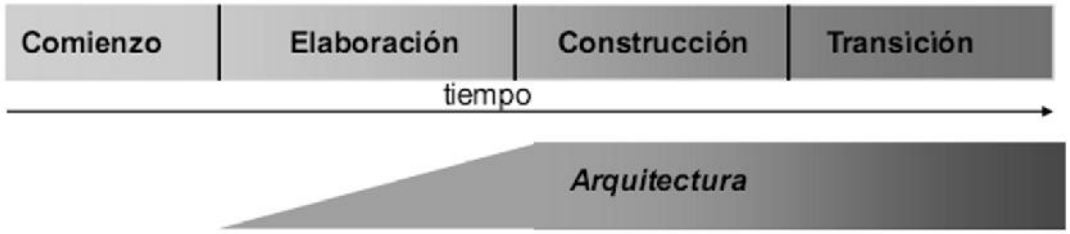


PU (cont.)

- Proceso dirigido por los casos de uso (cont.):
  - Los CU dirigen las iteraciones:
    - Creación y validación de la arquitectura.
    - Definición de los casos y procedimientos de prueba.
    - Planificación de las iteraciones.
    - Creación de la documentación de usuario.
    - Entrega del sistema.
  - Sincronización del contenido de los distintos modelos.

PU (cont.)

- Proceso centrado en la arquitectura:
  - Los modelos son el medio para visualizar, especificar, construir, generar y documentar la arquitectura del sistema.

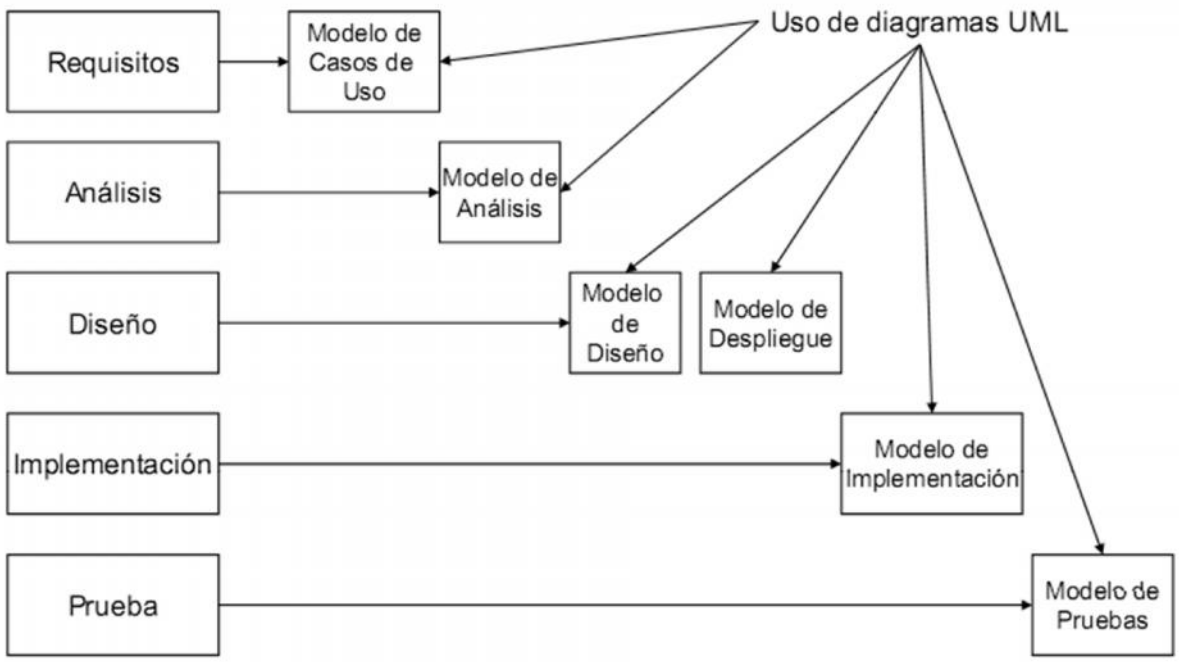


  - PU prescribe el refinamiento sucesivo de la arquitectura.

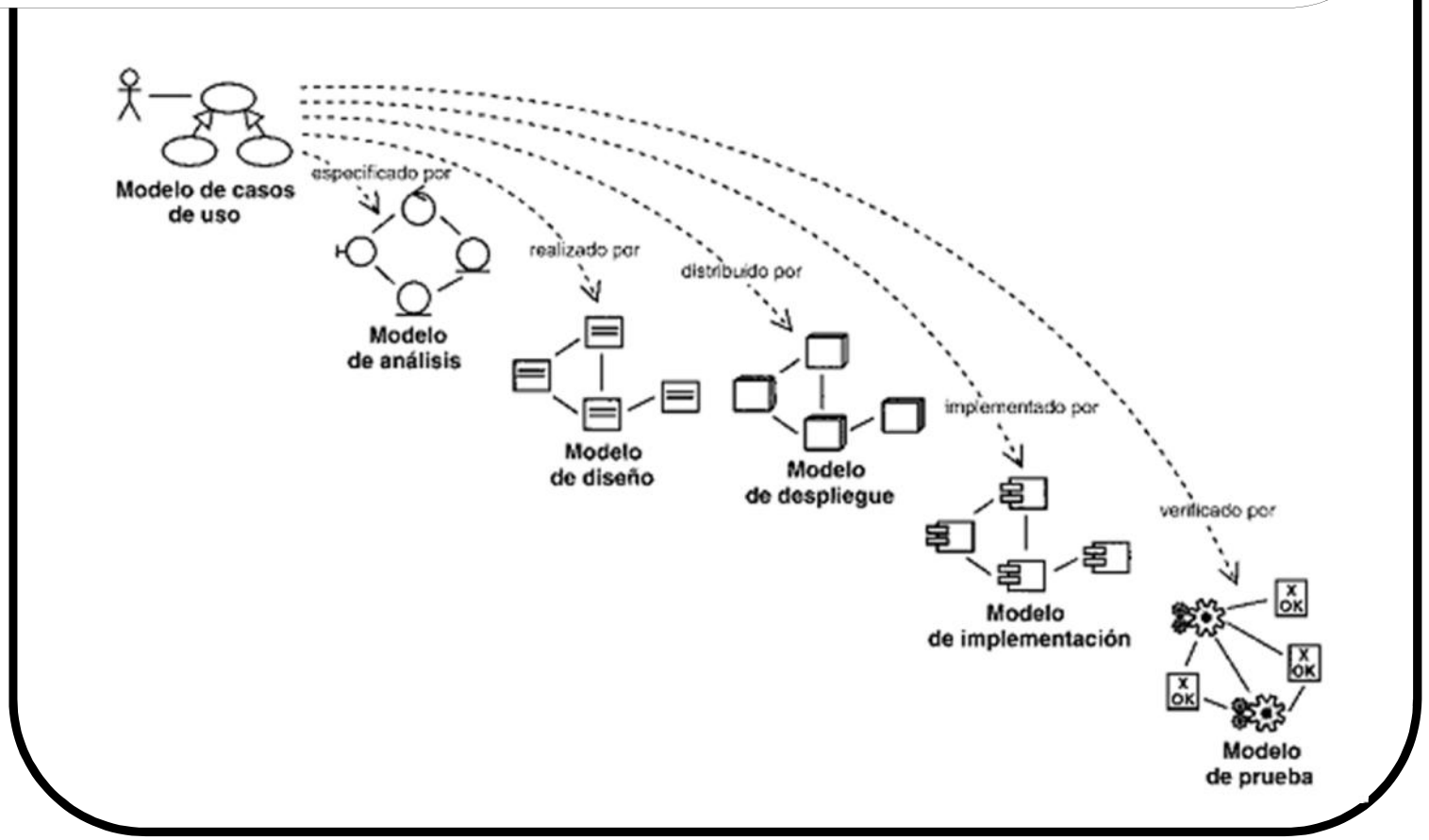
PU (cont.)

- Fases:
  - Inicio
    - Definir el ámbito del sistema.
    - Esbozar la arquitectura candidata.
  - Elaboración
    - Recopilar y refinar la mayor parte de los requisitos.
    - Desarrollar la línea base de la arquitectura.
  - Construcción
    - Desarrollar (construir) el producto.
  - Transición
    - Entrega a los usuarios.
- Cada fase se divide en iteraciones.

Modelos


Uso de diagramas UML
Requisitos
Modelo de Casos de Uso
Análisis
Modelo de Análisis
Diseño
Modelo de Diseño
Modelo de Despliegue
Implementación
Modelo de Implementación
Prueba
Modelo de Pruebas

Modelos (cont.)



Modelos (cont.)

- Modelo de Casos de Uso
  - Modelo que:
    - describe lo que el sistema debería hacer por sus usuarios y bajo que restricciones, y
    - está formado por:
      - actores,
      - casos de uso, y
      - las relaciones entre ambos.
  - Representa los requisitos funcionales.
  - El flujo de trabajo de requisitos, además, produce:
    - una descripción de la arquitectura (vista del modelo de CU), y
    - un prototipo de la interfaz de usuario.

Modelos (cont.)

- Modelo de Análisis
  - Modelo de objetos cuyo propósitos es:
    - describir los requisitos de forma precisa,
    - estructurarlos de manera que facilite su comprensión, y
    - servir de punto de partida para dar forma al sistema durante su diseño e implementación (incluyendo su arquitectura).
  - Refina los casos de uso con más detalle.
  - Establece la asignación inicial de funcionalidad del sistema a un conjunto de objetos que proporcionan el comportamiento.
  - El flujo de trabajo de análisis, además, produce:
    - una descripción de la arquitectura (vista del modelo de análisis)

Modelos (cont.)

- Modelo de Diseño
  - Modelo de objetos que:
    - describe la realización física de los casos de uso, y
    - se centra en cómo los requisitos funcionales y no funcionales, junto con otras restricciones referidas al entorno de implementación, afectan al sistema.
  - Define la estructura estática del sistema en la forma de subsistemas, clases e interfaces.
  - Define los casos de uso reflejados como colaboraciones entre los subsistemas, clases e interfaces.

Modelos (cont.)

- Modelo de Despliegue (o Distribución)
  - Modelo de objetos que:
    - describe la distribución física del sistema, en términos de cómo se distribuye la funcionalidad entre nodos computacionales.
  - Define los nodos físicos (computadoras) y la correspondencia de los componentes con esos nodos.

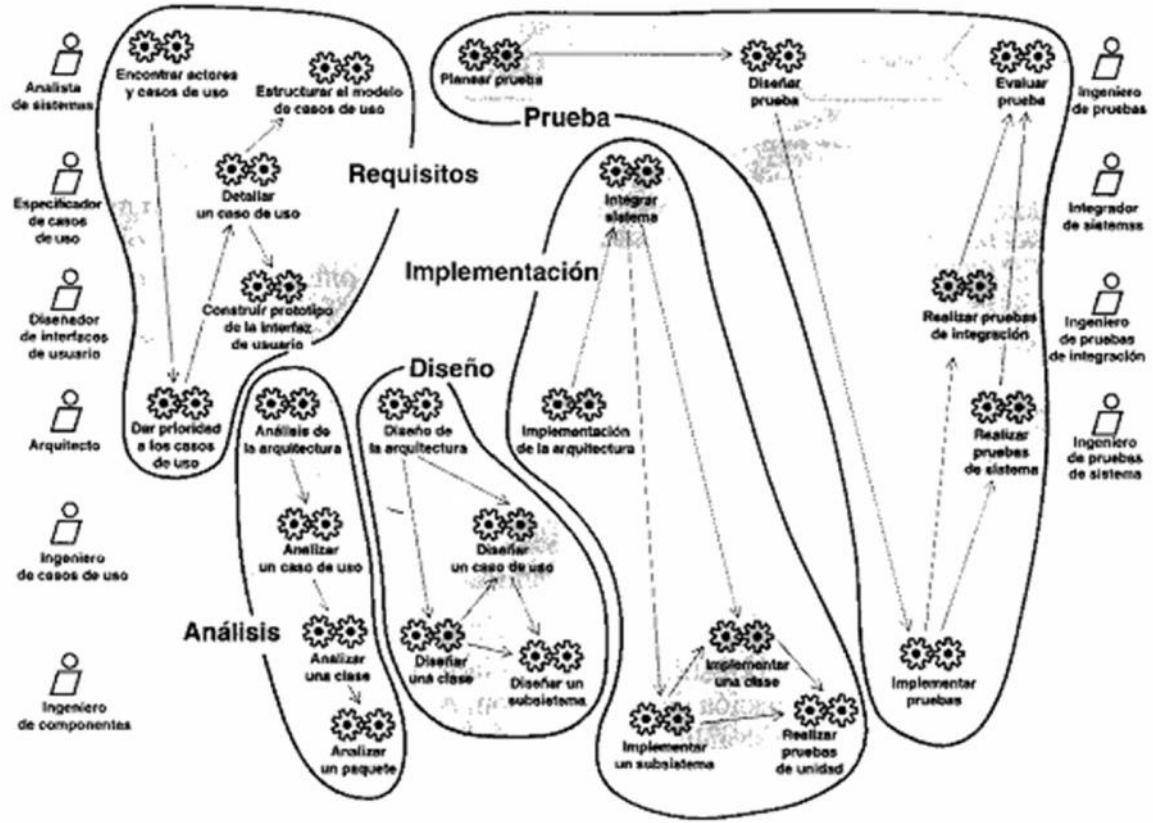
Modelos (cont.)

- Modelo de Implementación
  - Modelo que:
    - describe cómo se implementan los elementos del modelo de diseño, en términos de componentes (como archivos de código fuente y ejecutables).
  - Incluye componentes (que representan al código fuente) y la correspondencia de las clases con los componentes.

Modelos (cont.)

- Modelo de Prueba
  - Modelo que:
    - describe, fundamentalmente, cómo los componentes ejecutables del modelo de implementación son probados mediante las pruebas de integración y del sistema.
  - Describe los casos de prueba que verifican los casos de uso.

Trabajadores



Diseño en el PU

El Proceso Unificado de Desarrollo de Software

Jacobson, Booch, Rumbaugh

Diseño

- Se modela el sistema y se encuentra su forma (incluida la arquitectura) para que soporte todos los requisitos.

Diseño (cont.)

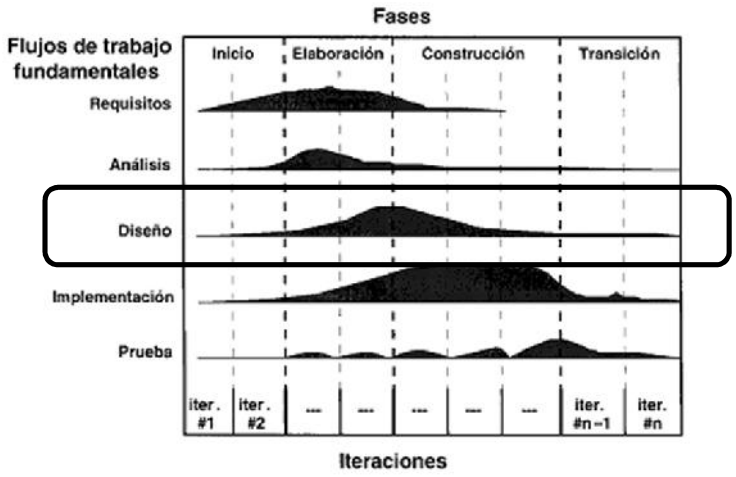
- Entradas:
  - Modelo del análisis
    - Es la entrada principal que:
      - Proporciona una comprensión detallada de los requisitos.
      - Impone una estructura del sistema que se debe conservar lo más fielmente posible.
  - Modelo de CU
  - Descripción de la arquitectura (vista del modelo del análisis)
  - Requisitos adicionales

Diseño (cont.)

- Salidas:
  - Modelo del Diseño
    - Modelo de objetos que:
      - describe la realización física de los casos de uso, y
      - se centra en cómo los requisitos funcionales y no funcionales, junto con otras restricciones referidas al entorno de implementación, afectan al sistema.
  - Modelo de Despliegue
    - Modelo de objetos que:
      - describe la distribución física del sistema, en términos de cómo se distribuye la funcionalidad entre nodos computacionales.
  - Descripción de la arquitectura (vista del modelo del diseño y vista del modelo de despliegue)

Diseño (cont.)

- Es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción.
  - Contribuye a:
    - una arquitectura estable y sólida, y
    - crear un plano del modelo de implementación.



Propósitos

- Adquirir una comprensión profunda de los aspectos relacionados con los requisitos no funcionales y las restricciones relacionadas con:
  - los lenguajes de programación,
  - componentes reutilizables,
  - sistemas operativos,
  - tecnologías de distribución y concurrencia,
  - tecnologías de interfaz de usuario,
  - tecnologías de gestión de transacciones,
  - …

Propósitos (cont.)

- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes, capturando los requisitos o subsistemas individuales, interfaces y clases.

- Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo (teniendo en cuenta posible concurrencia).

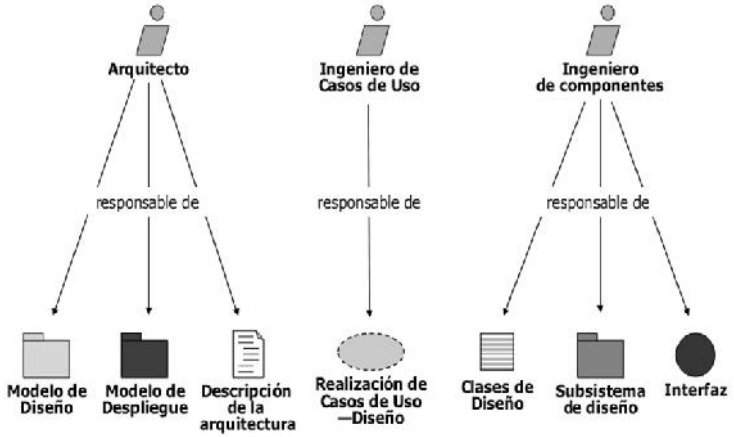
Propósitos (cont.)

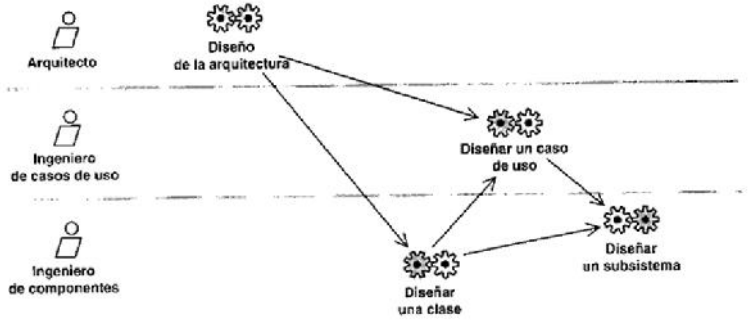
- Capturar las interfaces entre los subsistemas.
- Visualizar y reflexionar sobre el diseño utilizando una notación común.
- Crear una abstracción “sin costuras” de la implementación del sistema.
  - La implementación es un refinamiento directo del diseño que rellena lo existente sin cambiar la estructura.

M.Análisis vs M.Diseño

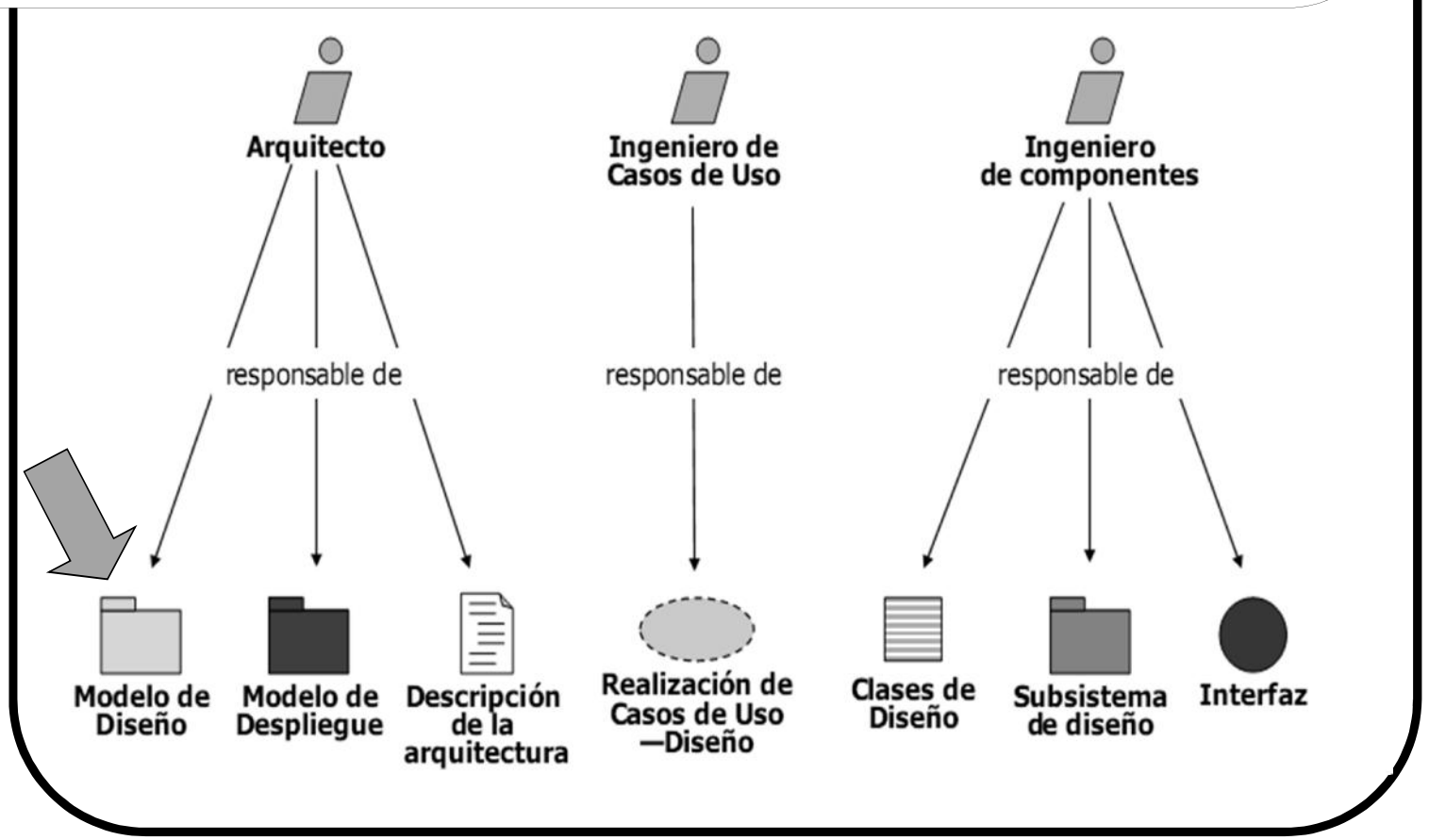
Modelo de análisis	Modelo de diseño
Modelo conceptual, porque es una abstracción del sistema y permite aspectos de la implementación.	Modelo físico, porque es un plano de la implementación.
Genérico respecto al diseño (aplicable a varios diseños).	No genérico, específico para una implemetación.
Tres estereotipos conceptuales sobre las clases: Control, Entidad e Interfaz.	Cualquier número de estereotipos (físicos) sobre las clases, dependiendo del lenguaje de implementación.
Menos formal.	Más formal.
Menos caro de desarrollar (ratio al diseño 1:5).	Más caro de desarrollar (ratio al análisis 5:1).
Menos capas.	Más capas.
Dinámico (no muy centrado en la secuencia).	Dinámico (muy centrado en las secuencias).
Bosquejo del diseño del sistema, incluyendo su arquitectura.	Manifiesto del diseño del sistema, incluyendo su arquitectura (una de sus vistas).
Creado principalmente como "trabajo de a pie" en talleres o similares.	Creado principalmente como "programación visual" en ingeniería de ida y vuelta; el modelo de diseño es realizado según la ingeriería de ida y vuelta con el modelo de implemetación
Puede no estar mantenido durante todo el ciclo de vida del software.	Debe ser mantenido durante todo el ciclo de vida del software.
Define una estructura que es una entrada esencial para modelar el sistema —incluyendo la creación del modelo de diseño.	Da forma al sistema mientras que intenta preservar la estructura definida por el modelo de análisis lo más posible.

Trabajadores, Artefactos y Flujo de Trabajo





Artefactos (1)



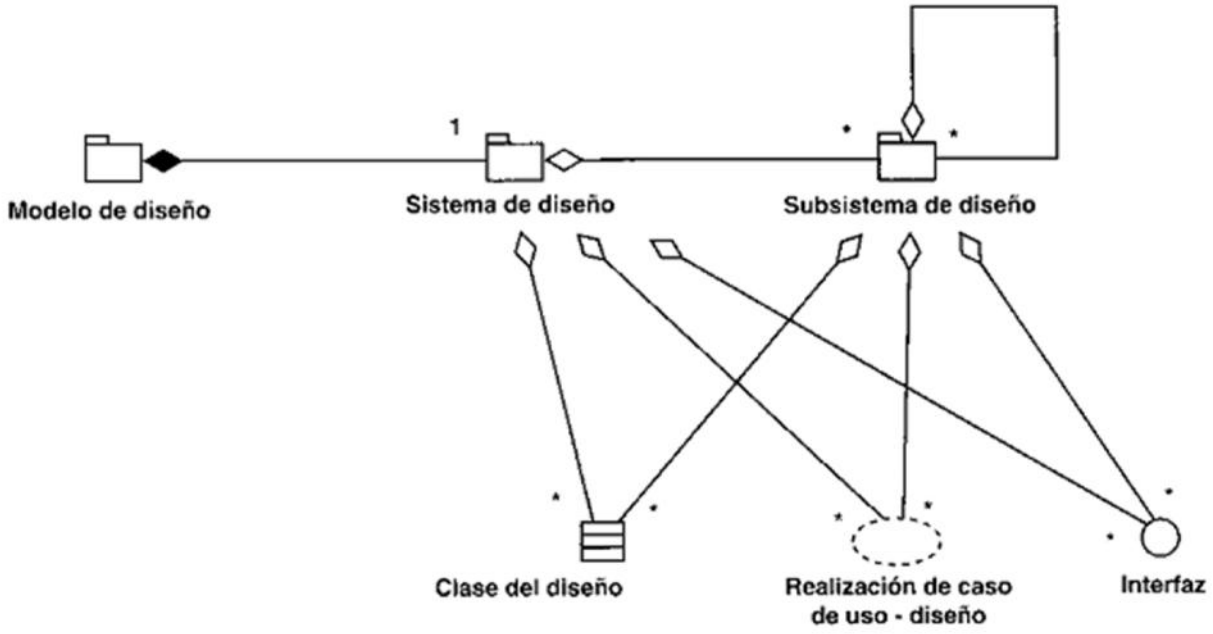
Modelo de Diseño

- Es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar.

- Sirve de abstracción de la implementación del sistema y se utiliza como una entrada fundamental de las actividades de implementación.

Modelo de Diseño (cont.)

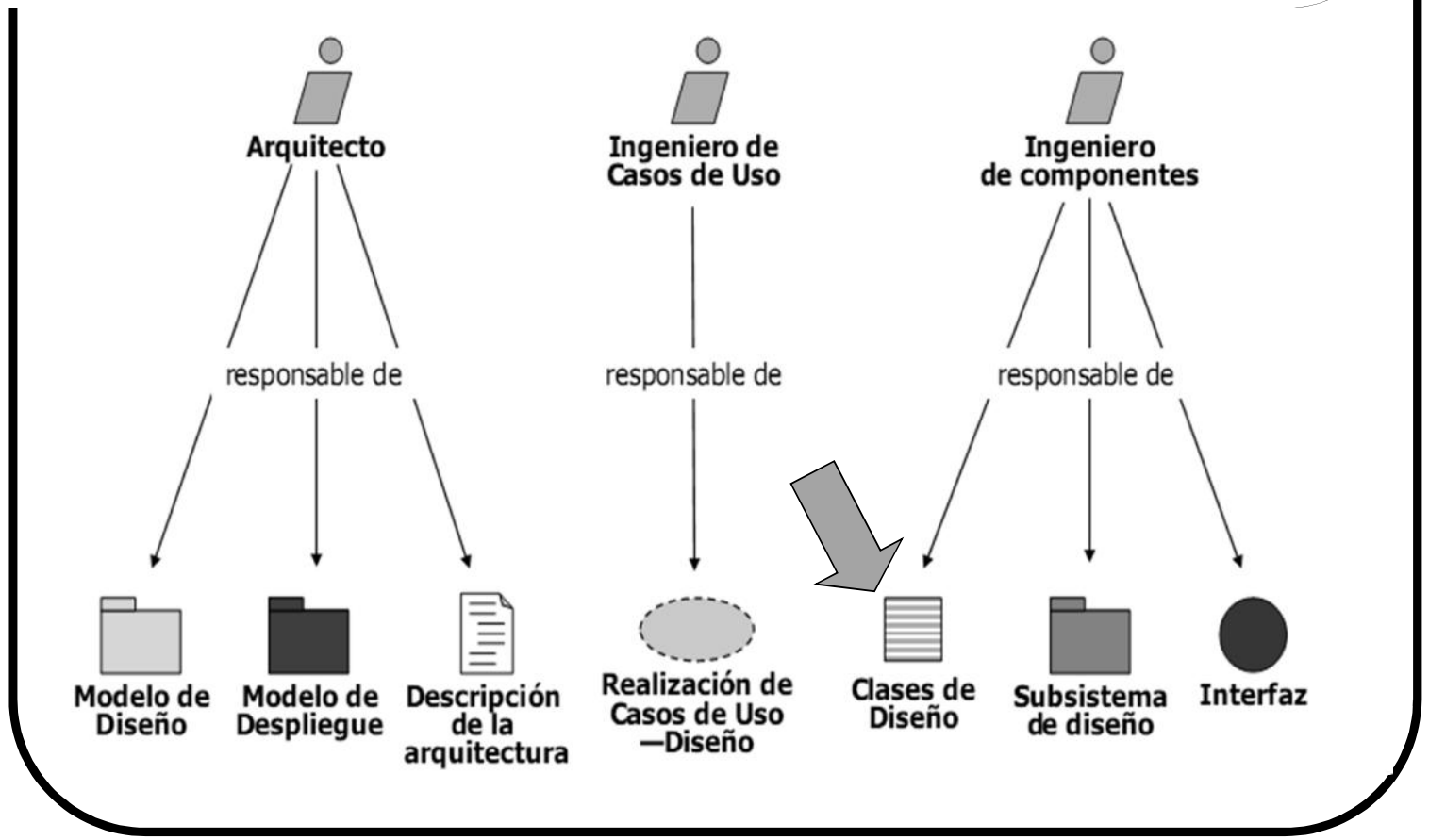
- Define la siguiente jerarquía:



Modelo de Diseño (cont.)

- Se representa por un **sistema de diseño** que denota el nivel más alto del modelo.
  - La utilización de otro subsistema es una forma de organización del modelo de diseño en partes más manejables.
- Los **subsistemas de diseño** y las **clases del diseño** representan abstracciones del subsistema y componentes de la implementación del sistema.
  - Estas abstracciones son directas, y representan una sencilla correspondencia entre el diseño y la implementación.
- Los CU son realizados por las **clases del diseño** y sus objetos.
  - Se representa por colaboraciones en el modelo de diseño => **Realización de CU-diseño**
    - Describe cómo se realiza un CU en términos de interacción entre objetos de diseño.

Artefactos (2)



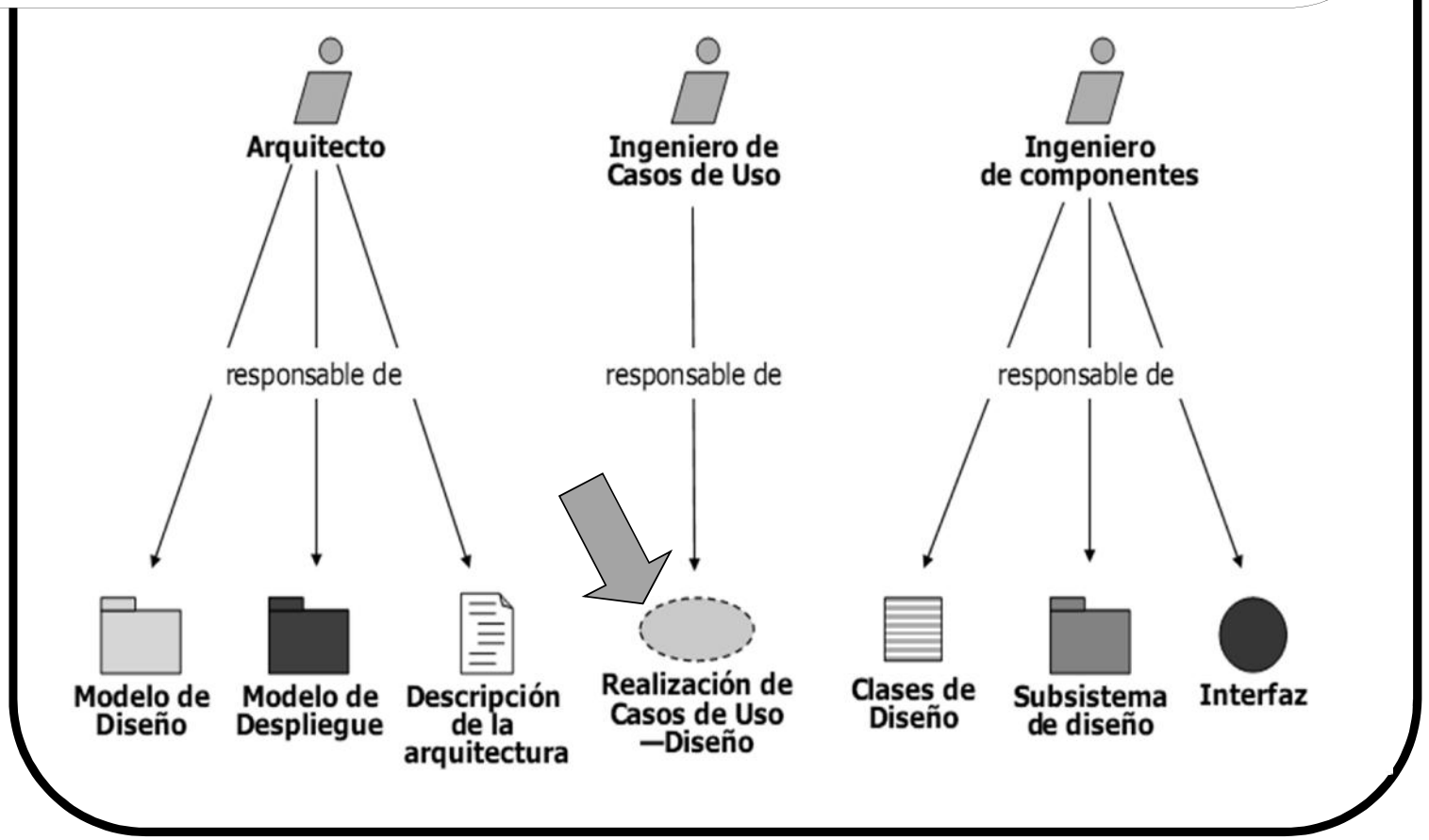
Clases del Diseño

- Representan una abstracción de una o varias clases (o construcción similar) en la implementación del sistema.
- Atributos clave y asociaciones:

Clases del Diseño (cont.)

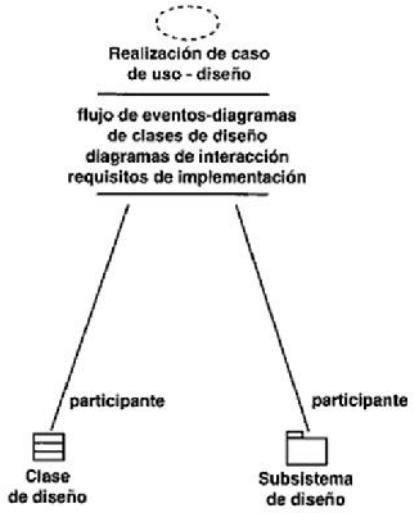
- Una clase del diseño es una abstracción “sin costuras”:
  - El lenguaje utilizado para especificar una clase del diseño es el mismo que el lenguaje de programación utilizado.
  - Se especifica la visibilidad de atributos y operaciones (public, protected, private).
  - Las relaciones entre clases del diseño suelen tener un significado directo cuando la clase se implementa.
  - Los métodos de una clase del diseño tienen correspondencia directa con el correspondiente método en la implementación de la clase.
  - Una clase del diseño puede aparecer como un estereotipo que se corresponde con una construcción en el lenguaje de programación dado (class module, form, …).
  - Una clase del diseño puede realizar, y proporcionar, interfaces si tiene sentido hacerlo en el lenguaje de programación.
  - Una clase del diseño se puede activar, implicando que objetos de la clase mantengan su propio hilo de control y se ejecuten concurrentemente con otros objetos activos.

Artefactos (3)



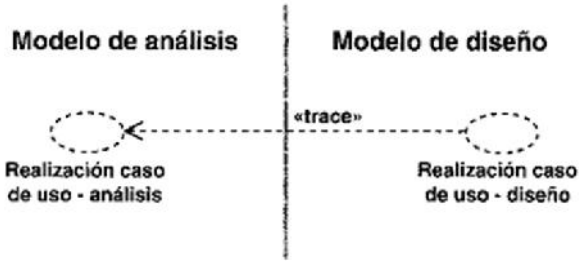
Realización de CU-diseño

- Es una colaboración en el modelo de diseño que describe:
  - cómo se realiza un CU específico, y
  - cómo se ejecuta (en términos de clases del diseño y sus objetos).
- Atributos clave y asociaciones:



Realización de CU-diseño (cont.)

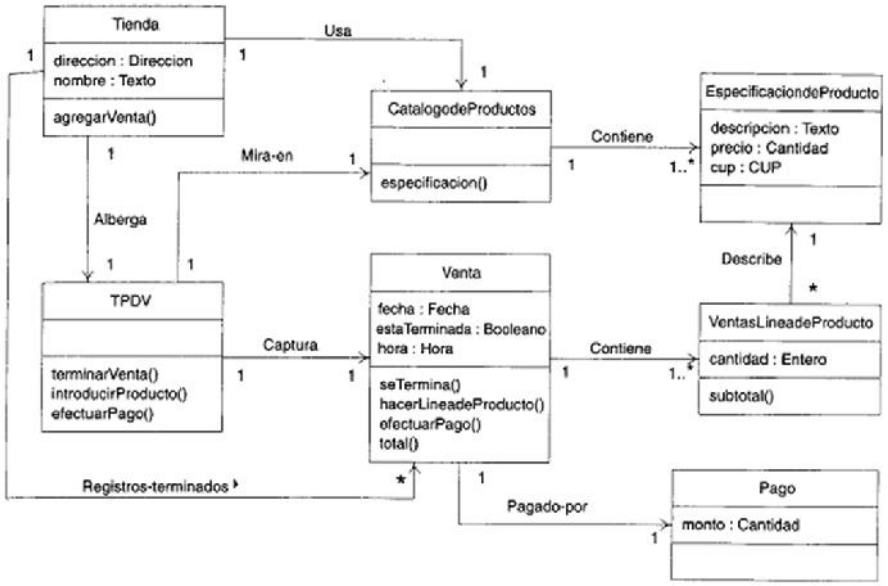
- Proporciona una traza directa a una realización de CU-análisis.



- Contiene:
  - Flujo de eventos.
  - Diagramas de clases de diseño.
  - Diagramas de interacción.
  - Requisitos de implementación.

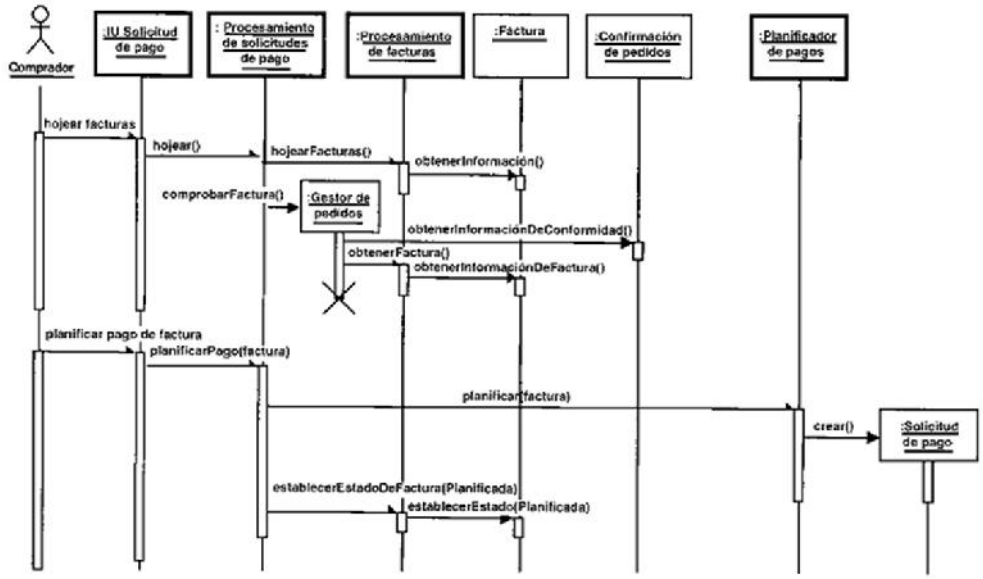
Realización de CU-diseño (cont.)

- Diagramas de clases de diseño
  - Con las clases del diseño participantes, subsistemas y sus relaciones.



Realización de CU-diseño (cont.)

- Diagramas de interacción
  - Diagramas de secuencia
    - Muestran las interacciones entre objetos mediante transferencia de mensajes entre objetos o subsistemas.



Realización de CU-diseño (cont.)

- Flujo de eventos
  - Descripción textual que explica y complementa a los diagramas.
  - Redactado en términos de objetos que interactúan para llevar a cabo el CU o en términos de los subsistemas que participan en él.

El comprador utiliza el sistema mediante el applet IU Solicitud de Pago y mediante la aplicación de Procesamiento de Solicitudes de Pago para hojear las solicitudes de pago recibidas. Procesamiento de Solicitudes de Pago utiliza al Gestor de Pedidos para comprobar las facturas con sus confirmaciones de pedido asociadas, antes de que el IU Solicitud de Pago muestre la lista de facturas al comprador.

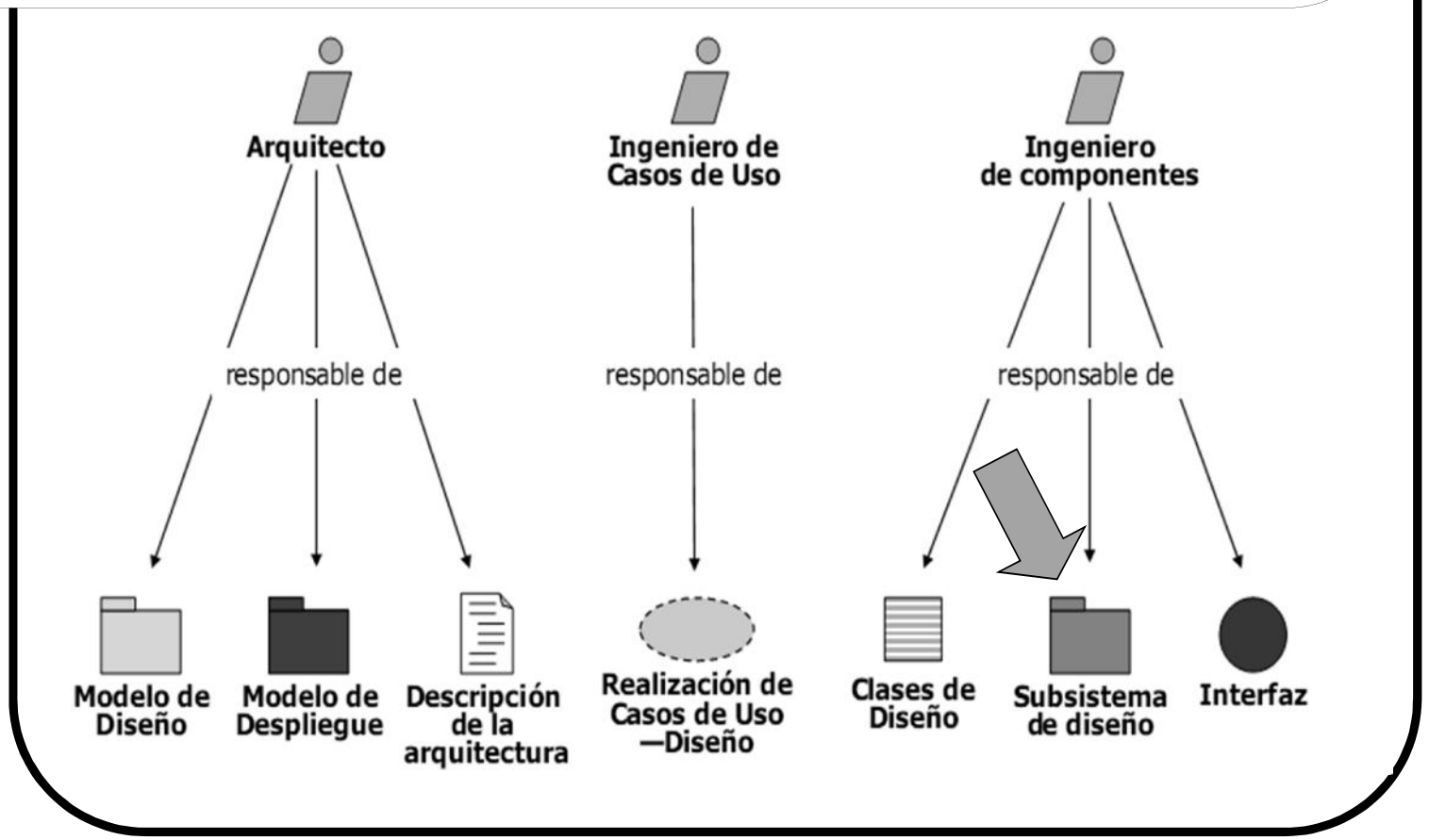
El comprador selecciona una factura mediante el IU Solicitud de Pago y planifica su pago, y a su vez el IU Solicitud de Pago pasa esta solicitud a Procesamiento de Solicitudes de Pago. Procesamiento de Solicitudes de Pago solicita al Planificador de Pagos la planificación del pago de la factura, y éste crea una Solicitud de Pago. La aplicación de Procesamiento de Solicitudes de Pago solicita después a la aplicación de Procesamiento de Facturas que cambie el estado de la factura a "Planificada".

Realización de CU-diseño (cont.)

- Requisitos de implementación
  - Descripción textual que recoge requisitos sobre una realización de CU.
  - Requisitos que se capturan en el diseño pero que se tratan en la implementación.

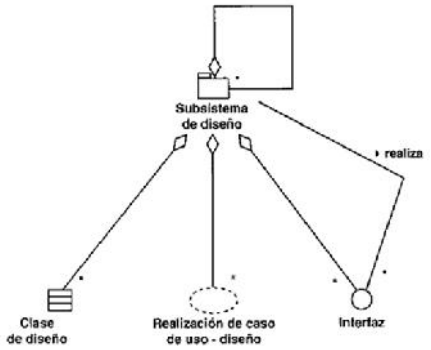
Un objeto de la clase activa Procesamiento de Solicitudes de Pago debería ser capaz de soportar 10 clientes de comprador diferentes sin un retraso perceptible para cada determinado comprador.

Artefactos (4)



Subsistemas de Diseño

- Son una forma de organizar los artefactos del modelo del diseño en piezas más manejables.
- Deben ser una colección cohesiva de:
  - clases de diseño,
  - realizaciones de caso de uso,
  - interfaces, y
  - otros subsistemas (recursivamente).

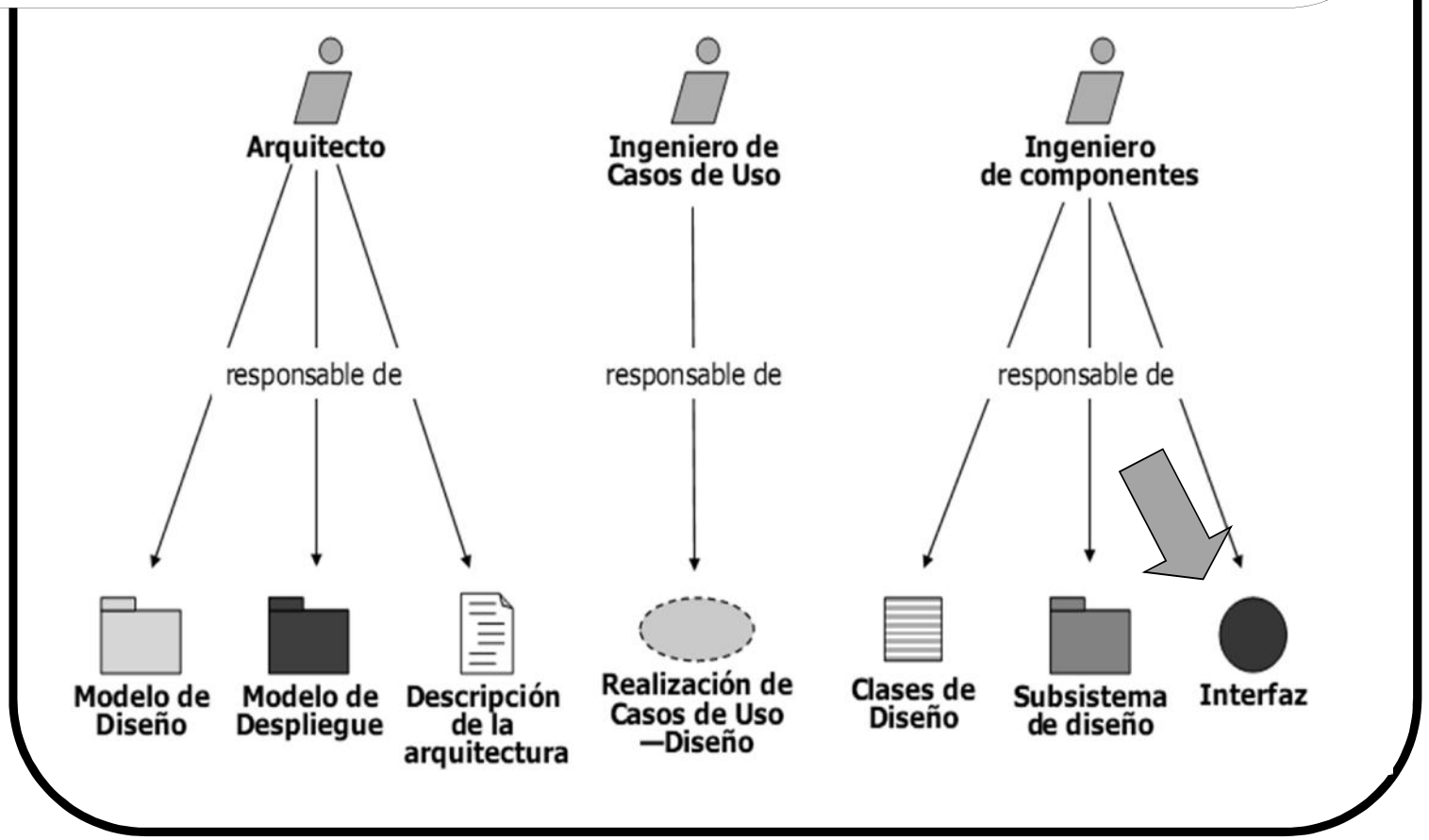


- El acoplamiento entre subsistemas debe de ser mínimo.

Subsistemas de Diseño (cont.)

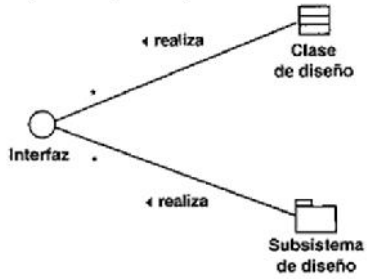
- Características:
  - Utilizados para separar los aspectos del diseño.
  - Las dos capas de aplicación de más alto nivel y sus subsistemas dentro del modelo de diseño suelen tener trazas directas hacia paquetes y/o clases del análisis.
  - Los subsistemas pueden representar componentes de grano grueso en la implementación del sistema, que:
    - proporcionan varias interfaces, compuestas a partir de otros componentes de grano más fino, y
    - se convierten ellos mismos en ejecutables, archivos binarios o entidades similares que pueden distribuirse en diferentes nodos.
  - Pueden representar productos software reutilizados que han sido encapsulados en ellos.
  - Pueden representar sistemas heredados o partes de ellos.

Artefactos (5)



Interfaces

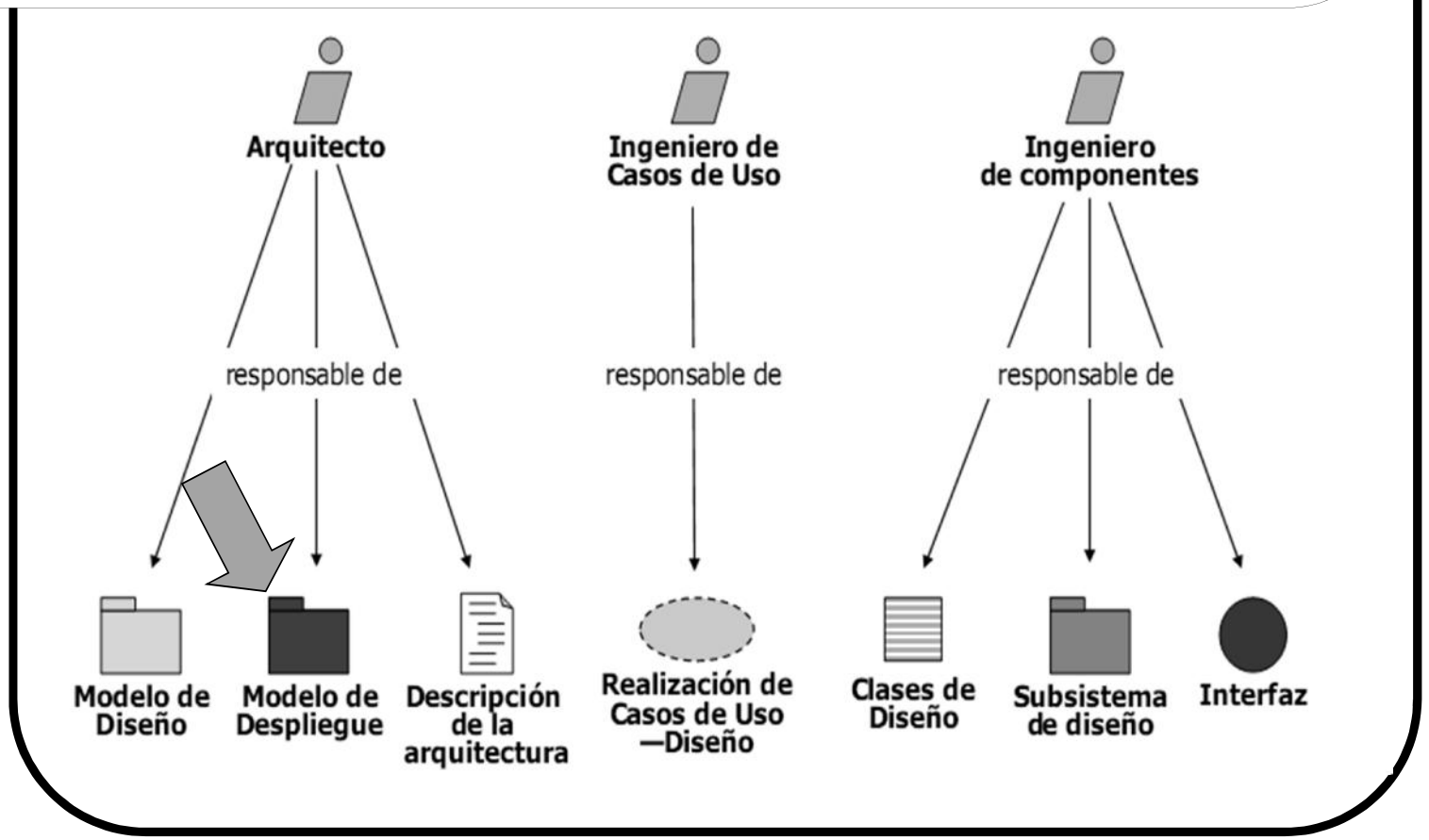
- Especifican una colección de operaciones públicas, tipos y parámetros necesarios para acceder y usar las capacidades de una clase del diseño o un subsistema.
  - Una clase de diseño que proporcione una interfaz debe proporcionar métodos que realicen las operaciones de la interfaz.
  - Un subsistema que proporcione una interfaz debe contener clases del diseño u otros subsistemas (recursivamente) que proporcionen la interfaz.
- Asociaciones:



Interfaces (cont.)

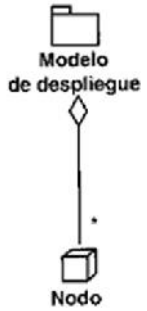
- Constituyen una forma de separar la especificación de la funcionalidad.
  - Separar operación de método.
  - Un cliente de la interfaz es independiente de la implementación de la misma.
- La mayoría de las interfaces entre subsistemas se consideran relevantes para la arquitectura debido a que definen las interacciones permitidas entre los subsistemas.

Artefactos (6)



Modelo de Despliegue

- Es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo.

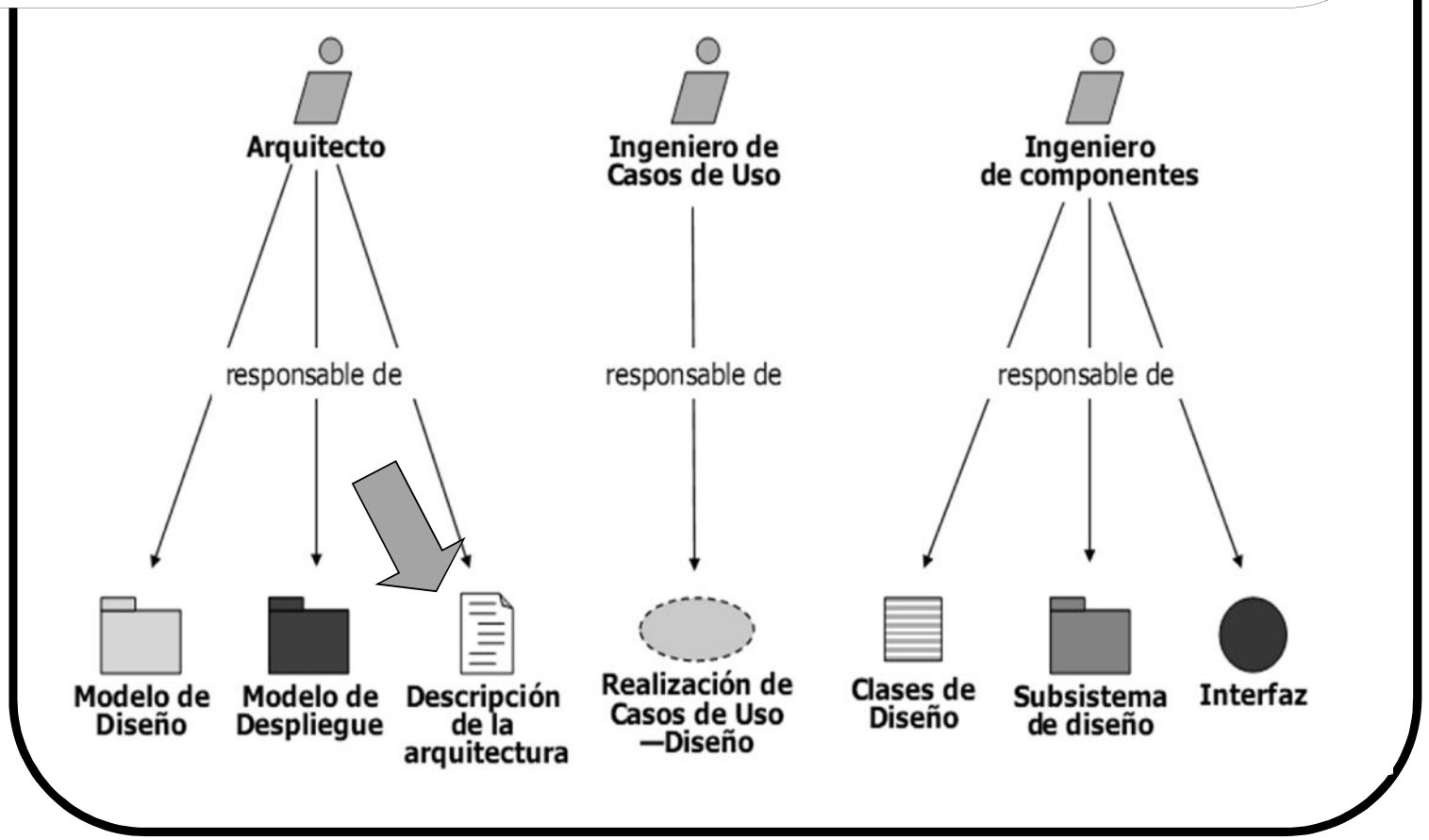


- Se utiliza como entrada fundamental en las actividades de diseño e implementación, ya que la distribución del sistema tiene una influencia principal en su diseño.

Modelo de Despliegue (cont.)

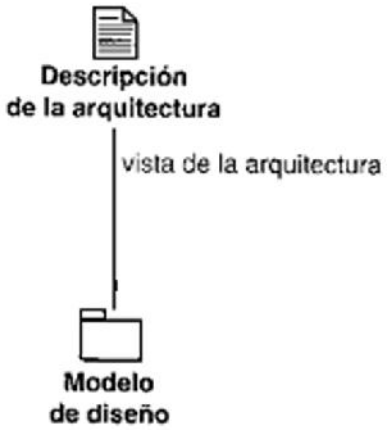
- Cada nodo representa un recurso de cómputo, normalmente un procesador o un dispositivo hardware similar.
- Los nodos poseen relaciones que representan medios de comunicación entre ellos.
  - Internet, intranet, bus, …
- Puede describir diferentes configuraciones de red.
- La funcionalidad (procesos) de un nodo se define por los componentes que se distribuyen en ese nodo.
- Representa una correspondencia entre la **arquitectura software** y la **arquitectura del sistema** (hardware).

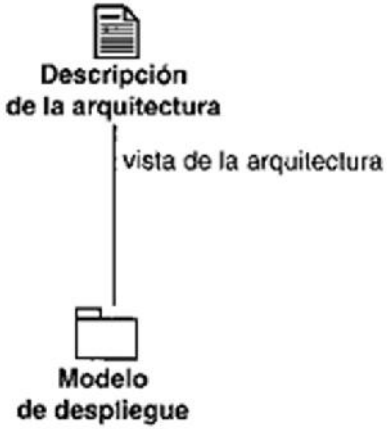
Artefactos (7)



Descripción de la Arquitectura

- Contiene una **vista de la arquitectura del modelo de diseño** y una **vista de la arquitectura del modelo de despliegue**.
  - Cada vista muestra sus artefactos relevantes para la arquitectura.





Descripción de la Arquitectura (cont.)

- Artefactos del Modelo de Diseño significativos:
  - Subsistemas, sus interfaces y las dependencias entre ellos.
    - Ya que los subsistemas y sus interfaces constituyen la estructura fundamental del sistema.
  - Clases del diseño fundamentales:
    - Clases con una traza a las clases de análisis significativas.
    - Clases activas.
    - Clases del diseño que sean generales y centrales, que representen mecanismos de diseño genéricos y que tengan muchas relaciones con otras clases del diseño.

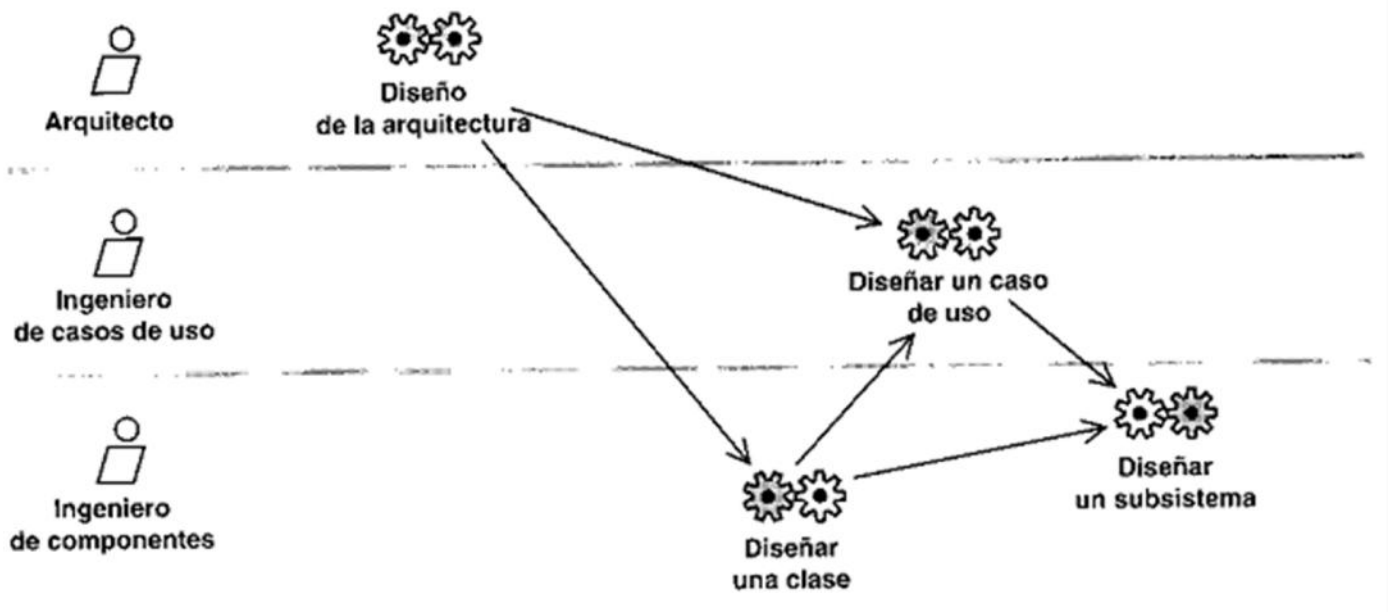
Descripción de la Arquitectura (cont.)

- Artefactos del Modelo de Diseño significativos (cont.):
  - Realizaciones de CU-diseño que:
    - Describan una funcionalidad importante y crítica y que deban desarrollarse pronto en el ciclo de vida.
    - Impliquen muchas clases del diseño y tengan una cobertura amplia, posiblemente a lo largo de varios subsistemas.
    - Impliquen clases del diseño fundamentales.

Descripción de la Arquitectura (cont.)

- Artefactos del Modelo de Despliegue significativos:
  - Deberían mostrarse todos los aspectos del modelo de despliegue.

Flujo de Trabajo



Trabajadores y Flujo de Trabajo

- Los **arquitectos** inician la creación de los modelos de diseño y de despliegue. Esbozan:
  - nodos del modelo de despliegue,
  - subsistemas principales y sus interfaces,
  - clases del diseño importantes, como las activas, y
  - mecanismos genéricos de diseño del modelo de diseño.

- Luego, los **ingenieros de CU** realizan cada CU en términos de:
    - clases y/o subsistemas participantes,
    - y sus interfaces.
  - Las realizaciones de CU resultantes establecen los requisitos de comportamiento para cada clase/subsistema que participe en ellos.

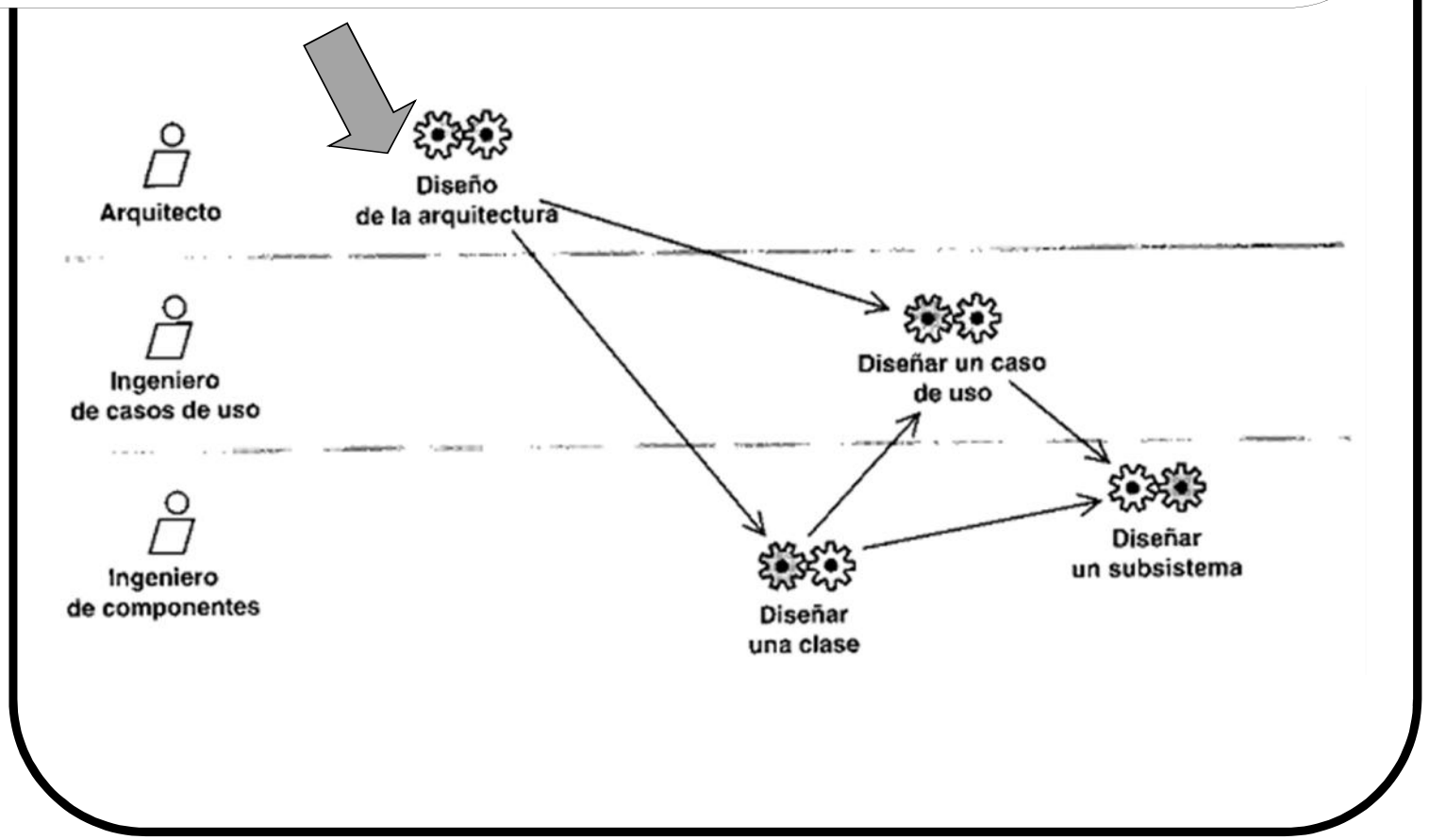
Trabajadores y Flujo de Trabajo

(cont.)

- A continuación, los **ingenieros de componentes** especifican los requisitos, y los integran dentro de cada clase mediante:
  - la creación de operaciones, atributos y relaciones consistentes sobre cada clase, o
  - la creación de operaciones consistentes en cada interfaz que proporcione el subsistema.

- A lo largo de todo el flujo de trabajo:
  - los **desarrolladores** identificarán, a medida que evolucione el diseño, nuevos candidatos a subsistemas, interfaces, clases y mecanismos de diseño genéricos, y
  - los **ingenieros de componentes** responsables de los subsistemas individuales deberán refinarlos y mantenerlos.

Flujo de Trabajo (1)

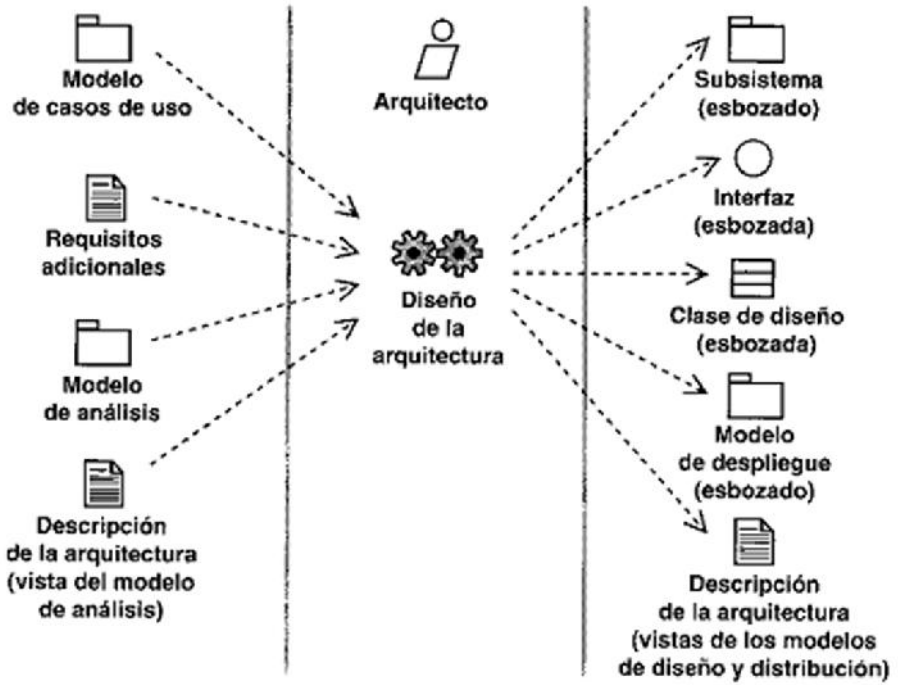


Diseño de la Arquitectura

- Su objetivo es esbozar los modelos de diseño y de despliegue y su arquitectura mediante la identificación de:
  1. nodos y sus configuraciones de red,
  2. subsistemas y sus interfaces,
  3. clases del diseño relevantes para la arquitectura, como las activas, y
  4. mecanismos genéricos de diseño que tratan requisitos comunes:
     - requisitos especiales de persistencia,
     - requisitos especiales de distribución,
     - requisitos especiales de rendimiento,
     - …

Diseño de la Arquitectura (cont.)

- Entradas y resultados:



Diseño de la Arquitectura (cont.)

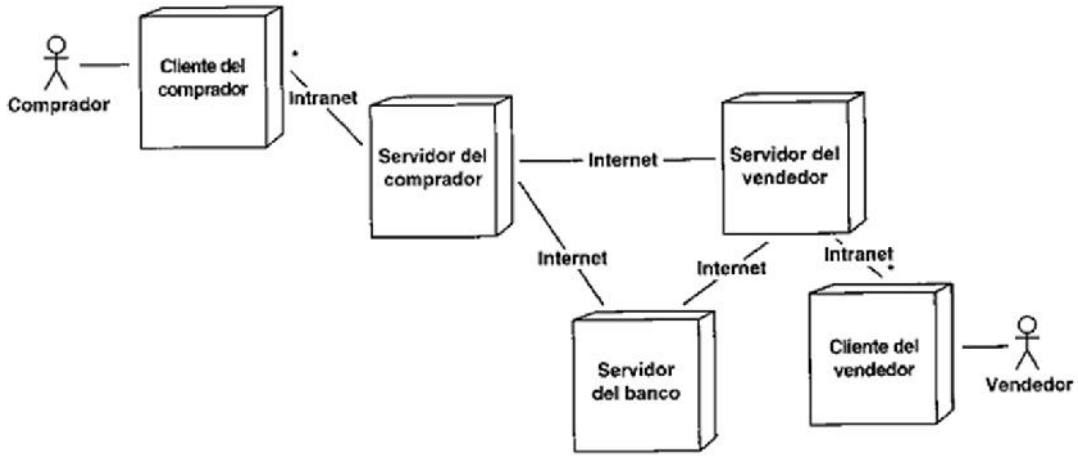
- A lo largo del proceso, los arquitectos deben considerar distintas posibilidades de reutilización (de sistemas parecidos u otros).
- Los elementos de diseño resultantes se añadirán luego al modelo de diseño.
- El arquitecto mantiene, refina y actualiza la descripción de la arquitectura y sus vistas arquitectónicas de los modelos de diseño y despliegue.

Diseño de la Arquitectura (cont.)

1. Identificación de nodos y configuraciones de red:
   - Las configuraciones físicas de red suelen tener una gran influencia sobre la arquitectura del software, incluyendo las clases activas que se necesitan y la distribución de la funcionalidad entre los nodos de la red.
   - Aspectos a considerar:
     - ¿Qué nodos se necesitan? ¿Cuál debe ser su capacidad en términos de potencia de procesamiento y tamaño de memoria?
     - ¿Qué tipo de conexiones debe haber entre los nodos? ¿Qué protocolos de comunicaciones deben utilizarse?
     - ¿Qué características deben tener las conexiones y los protocolos de comunicaciones (ancho de banda, disponibilidad, calidad, …)?
     - ¿Es necesario tener alguna capacidad de proceso redundante, modos de fallo, migración de procesos, mantenimiento de copias de seguridad de los datos, o aspectos similares?

Diseño de la Arquitectura (cont.)

1. Identificación de nodos y configuraciones de red (cont.):
   - Cada configuración de red (incluidas las de prueba y simulación) debería mostrarse en un **diagrama de despliegue** separado.



Diseño de la Arquitectura (cont.)

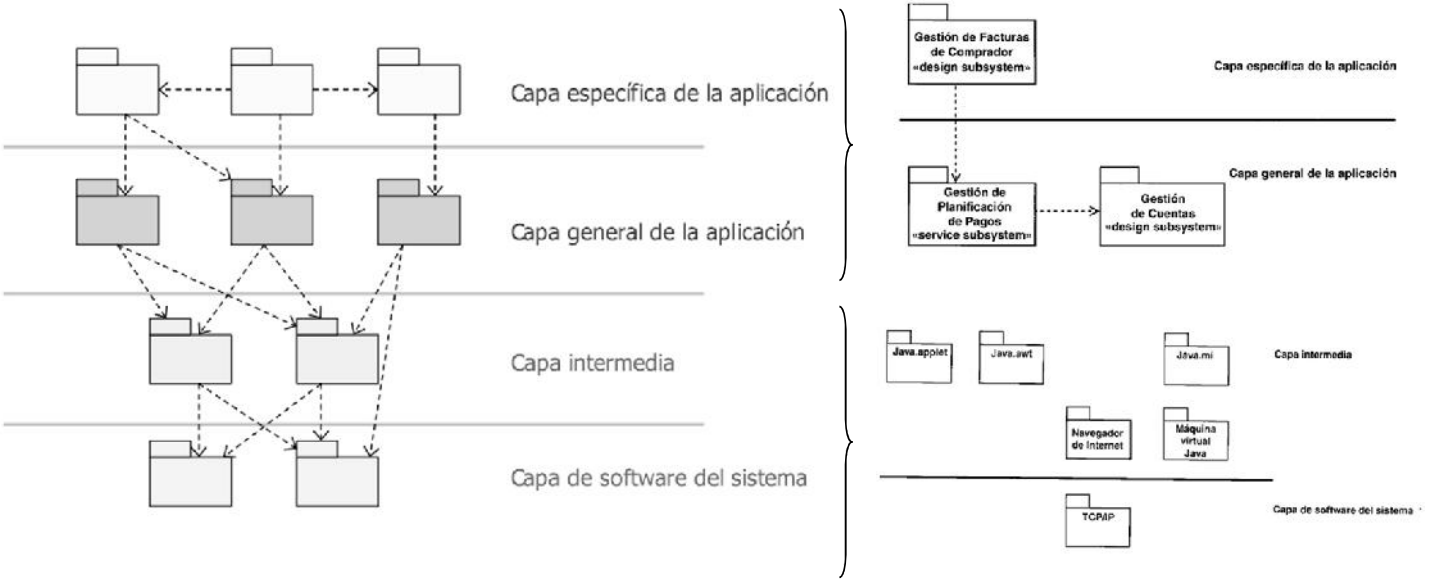
2. Identificación de subsistemas y sus interfaces:
   - Los subsistemas son un medio para organizar el modelo de diseño en piezas manejables.
   - Los subsistemas pueden identificarse:
     - inicialmente (para dividir el trabajo de diseño), o
     - a medida que el diseño evoluciona.
   - No todos los subsistemas se desarrollan internamente en el proyecto en curso.
     - Algunos representan productos reutilizados/heredados.
       - El arquitecto debe verificar que los productos de software elegidos encajan en la arquitectura y permiten una implementación económica del sistema.

Diseño de la Arquitectura (cont.)

2. Identificación de subsistemas y sus interfaces (cont.):
   - Los subsistemas se suelen organizar siguiendo un patrón de capas:
     - facilita la organización jerárquica de los subsistemas en capas,
     - los subsistemas de una capa sólo pueden referenciar subsistemas de un nivel igual o inferior,
     - la comunicación entre los subsistemas de diferentes capas se lleva a cabo mediante un conjunto de interfaces bien definidas.

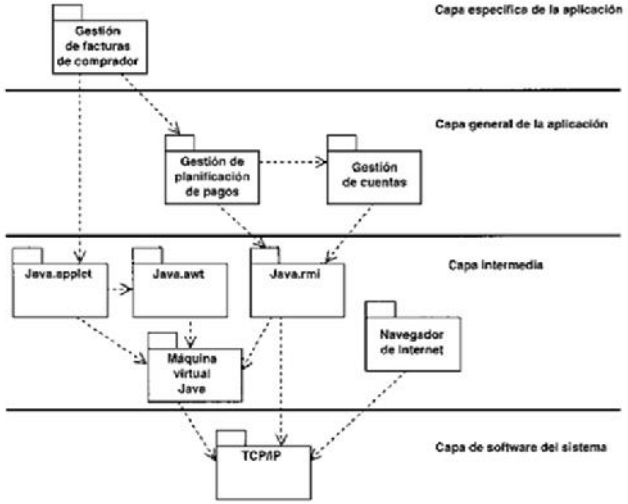
Diseño de la Arquitectura (cont.)

2. Identificación de subsistemas y sus interfaces (cont.):



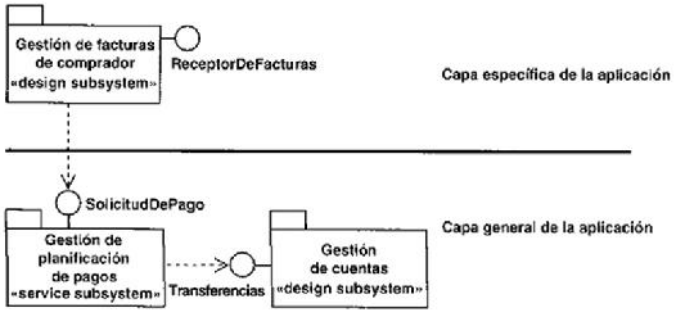
Diseño de la Arquitectura (cont.)

2. Identificación de subsistemas y sus interfaces (cont.):
   - Definición de las dependencias:
     - Deberían definirse dependencias entre subsistemas si sus contenidos tienen relación entre sí.
     - La dirección de la dependencia debería ser la misma que la dirección de la navegabilidad de la relación.
       - Si se utilizan interfaces entre subsistemas, las dependencias deberían ir hacia las interfaces.



Diseño de la Arquitectura (cont.)

2. Identificación de subsistemas y sus interfaces (cont.):
   - Identificación de interfaces entre subsistemas:
     - Las interfaces proporcionadas por un subsistema definen las operaciones accesibles desde el exterior del subsistema.
     - Las interfaces son proporcionadas por clases o subsistemas internos al subsistema.
     - Cuando un subsistema tiene una dependencia que apunta hacia él, es probable que deba proporcionar una interfaz.
     - Si los subsistemas de las capas inferiores encapsulan productos software, éstos generalmente ya tienen interfaces definidas.



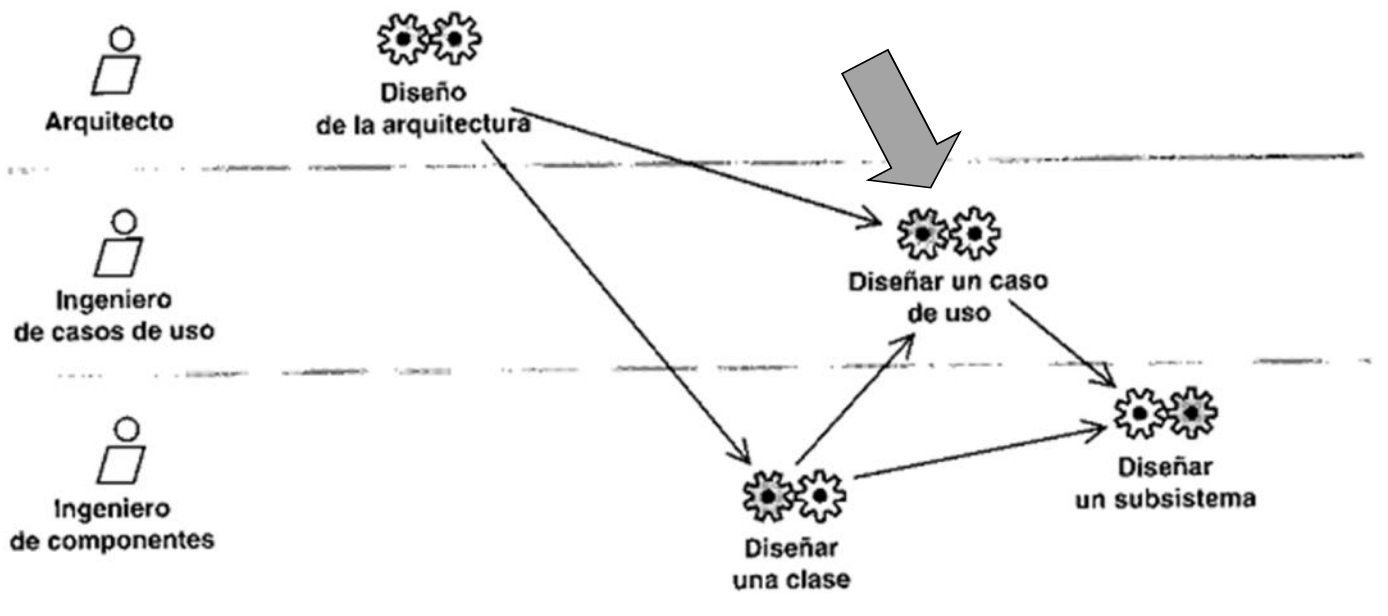
Diseño de la Arquitectura (cont.)

3. Identificación de clases del diseño relevantes para la arquitectura:
   - Se debe evitar identificar demasiadas clases en esta etapa o tener demasiados detalles.
     - Conviene sólo un esbozo inicial de las clases relevantes.
   - Se pueden esbozar inicialmente algunas clases a partir de las clases del análisis.
   - Se deben identificar las clases activas necesarias, considerando los requisitos de concurrencia:
     - Requisitos de rendimiento, tiempo de respuesta y disponibilidad de los diferentes actores en su interacción con el sistema.
     - Distribución del sistema sobre los nodos.
     - Otros requisitos (arranque y terminación del sistema, progresión, evitar interbloqueos, evitar inanición, reconfiguración de los nodos, la capacidad de los nodos…).
   - Los objetos activos se asignan a los nodos del modelo de despliegue.

Diseño de la Arquitectura (cont.)

4. Identificación de mecanismos genéricos de diseño:
   - Se tratan los requisitos especiales identificados durante el análisis.
   - Se tienen en cuenta las tecnologías de diseño e implementación disponibles.
   - También se deben identificar colaboraciones genéricas que puedan servir como patrones utilizados por varias realizaciones de CU dentro del modelo de diseño.
   - El resultado es un conjunto de mecanismos genéricos de diseño que pueden manifestarse como clases, colaboraciones o incluso como subsistemas.
   - Los requisitos que deben tratarse suelen estar relacionados con aspectos como:
     - Persistencia
     - Distribución transparente de objetos
     - Características de seguridad
     - Detección y recuperación de errores
     - Gestión de transacciones

Flujo de Trabajo (2)



Diseño de un CU

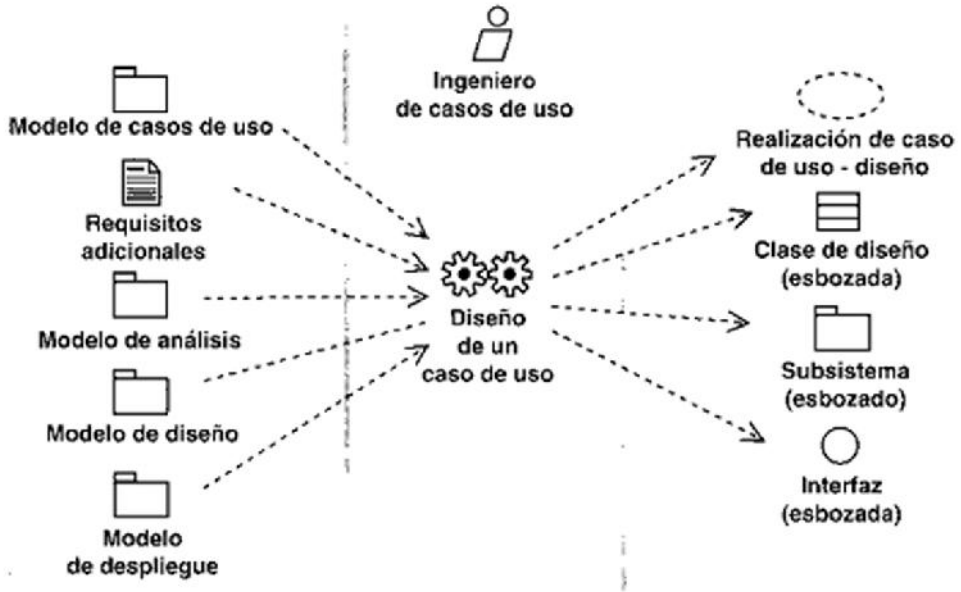
- Sus objetivos son:
  - Identificar las clases del diseño y/o los subsistemas cuyas instancias son necesarias para llevar a cabo el flujo de eventos del CU.
  - Distribuir el comportamiento del CU entre los objetos del diseño que interactúan, y/o los subsistemas participantes.
  - Definir los requisitos sobre las operaciones de las clases del diseño, y/o los subsistemas y sus interfaces.
  - Capturar los requisitos de implementación del CU.

Diseño de un CU (cont.)

1. Identificación de clases del diseño participantes.
2. Identificación de las interacciones entre objetos del diseño.
3. Identificación de subsistemas e interfaces participantes.
4. Identificación de las interacciones entre subsistemas.
5. Captura de requisitos de implementación.

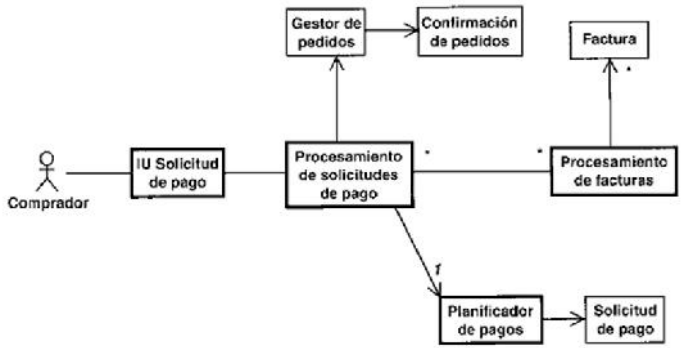
Diseño de un CU (cont.)

- Entradas y resultados:



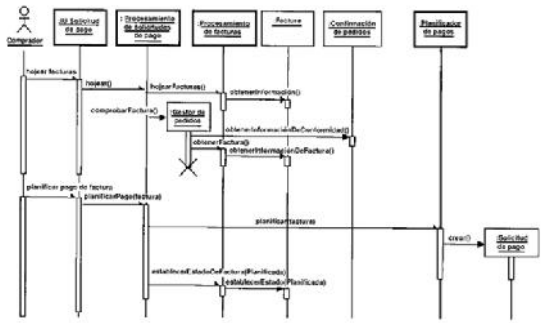
Diseño de un CU (cont.)

1. Identificación de clases del diseño participantes:
   - Se debe:
     - identificar las clases del diseño que se necesitan para realizar el CU (incluyendo las que realizan los requisitos especiales), y
     - recogerlas en un diagrama de clases asociado con la realización.



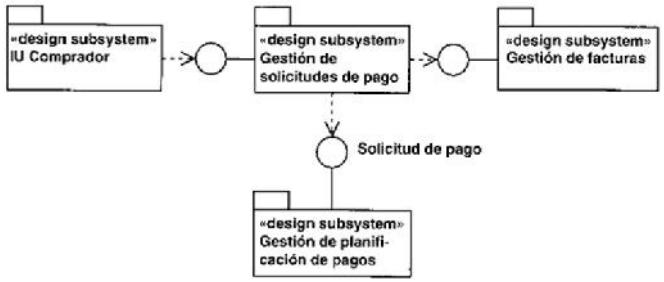
Diseño de un CU (cont.)

2. Identificación de las interacciones entre objetos del diseño:
   - Se debe describir cómo interactúan los objetos del diseño del CU mediante diagramas de secuencia (DS), que contienen:
     - las instancias de los actores,
     - los objetos del diseño, y
     - las transmisiones de mensajes entre éstos.



Diseño de un CU (cont.)

3. Identificación de subsistemas e interfaces participantes:
   - Se debe:
     - identificar los subsistemas que se necesitan para realizar el CU (incluyendo a los que contienen las clases que realizan los requisitos especiales), y
     - recogerlos en un diagrama de clases asociado con la realización.



Diseño de un CU (cont.)

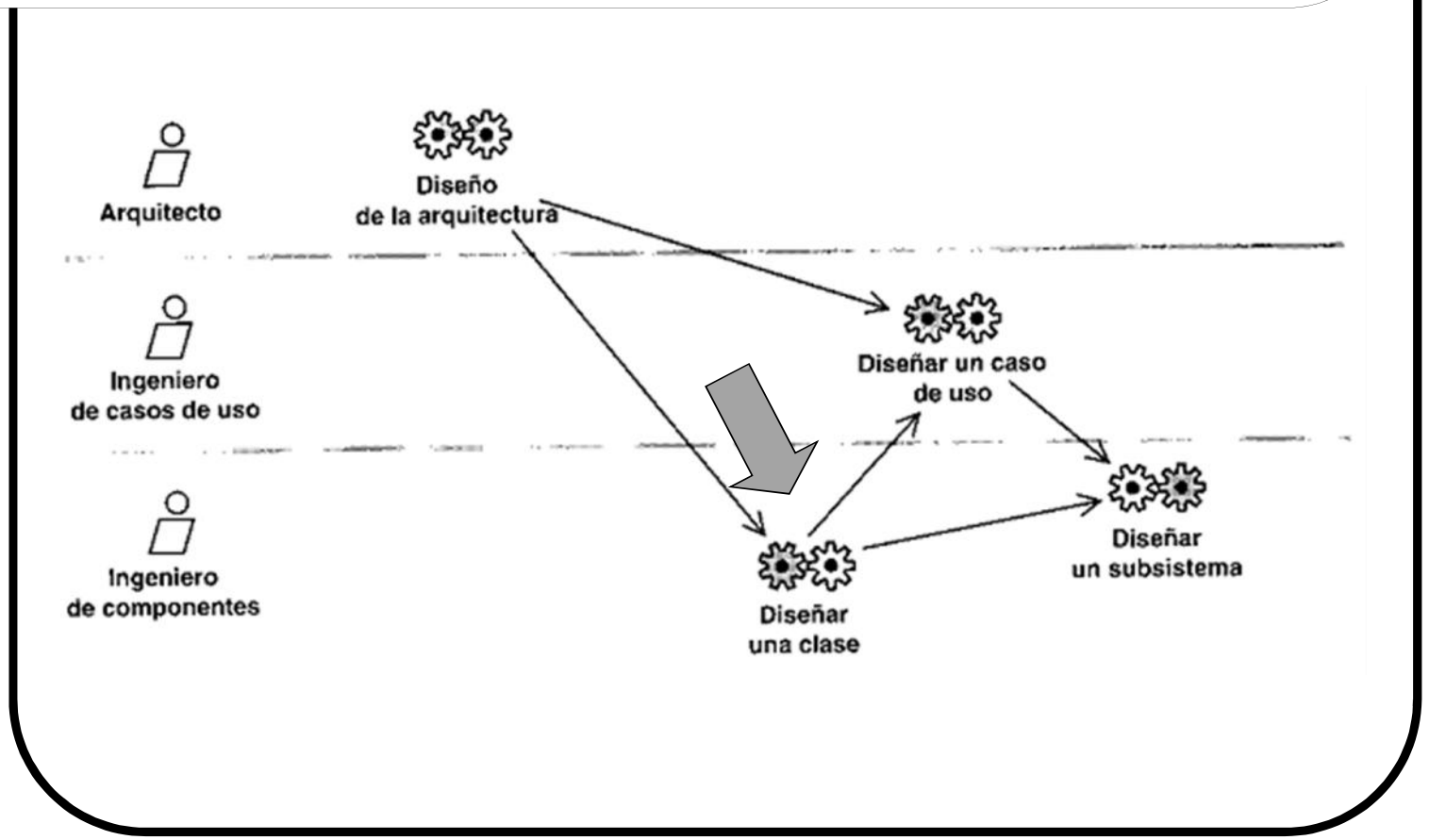
4. Identificación de las interacciones entre subsistemas:
   - Se debe describir cómo interactúan los objetos de las clases que contiene el CU a nivel de subsistema mediante diagramas de secuencia (DS), que contienen:
     - las instancias de los actores,
     - los subsistemas, y
     - las transmisiones de mensajes entre éstos.

Diseño de un CU (cont.)

5. Captura de requisitos de implementación:
   - Se incluye en la realización del CU todos los requisitos identificados durante el diseño que deberían tratarse en la implementación (por ejemplo, los no funcionales).

Un objeto de la clase activa Procesamiento de Solicitudes de Pago debería ser capaz de soportar 10 clientes de comprador diferentes sin un retraso perceptible para cada determinado comprador.

Flujo de Trabajo (3)



Diseño de una Clase

- El propósito de diseñar una clase es crear una clase del diseño que cumpla su papel en las realizaciones de los CU y los requisitos no funcionales que se aplican a éstos.

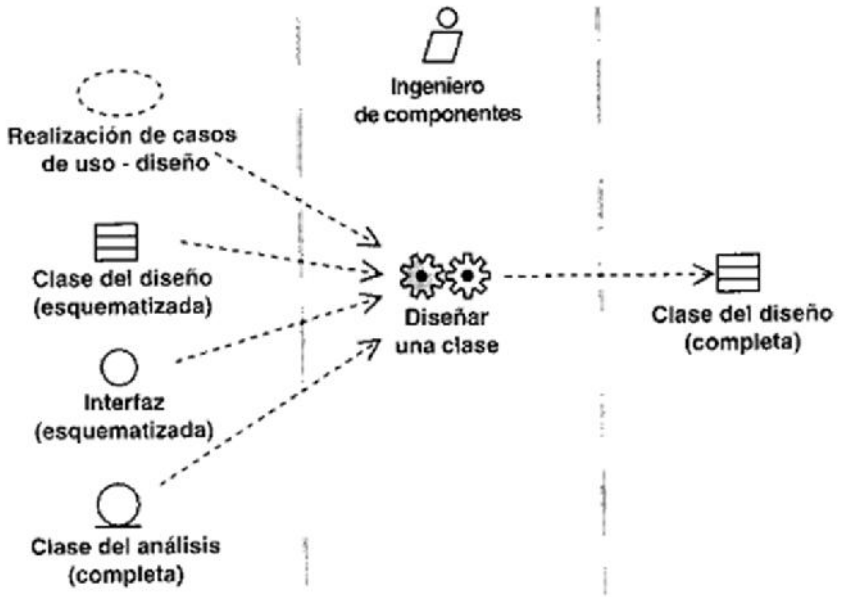
- Incluye:
  - El mantenimiento del diseño de clases en sí mismo.
  - Sus operaciones
  - Sus atributos.
  - Las relaciones en las que participa.
  - Sus métodos (que realizan sus operaciones).
  - Los estados impuestos.
  - Sus dependencias con cualquier mecanismo de diseño genérico.
  - Los requisitos relevantes a su implementación.
  - La correcta realización de cualquier interfaz requerida.

Diseño de una Clase (cont.)

1. Esbozar la clase del diseño.
2. Identificar operaciones.
3. Identificar atributos.
4. Identificar asociaciones y agregaciones.
5. Identificar las generalizaciones.
6. Describir los métodos.
7. Describir estados.
8. Tratar requisitos especiales.

Diseño de una Clase (cont.)

- Entradas y resultados:



Diseño de una Clase (cont.)

1. Esbozar la clase del diseño:
   - Se debe:
     - esbozar una o varias clases del diseño (teniendo como entrada las clases del análisis y/o interfaces), y
     - asignarlas trazando dependencias a las correspondientes clases de análisis que son diseñadas.
   - Si la entrada es una interfaz:
     - Se asigna una clase de diseño para que proporcione dicha interfaz.

Diseño de una Clase (cont.)

2. Identificar operaciones:
   - Se debe:
     - identificar las operaciones que las clases del diseño van a necesitar (teniendo en cuenta responsabilidades, requisitos especiales, interfaces, realizaciones de CU-diseño), y
       - si el comportamiento de los objetos de la clase depende fuertemente del estado del objeto $\Rightarrow$ analizarlo utilizando un diagrama de estados (7)
     - describir esas operaciones utilizando la sintaxis de los lenguajes de programación (incluyendo la visibilidad).
   - Las operaciones de la clase del diseño tienen que soportar todos los roles que las clases desempeñan en las diferentes realizaciones de CU.
     - Los roles se encuentran yendo a través de las realizaciones de los CU y mirando si la clase y sus objetos están incluidos en los diagramas y en los flujos de eventos de las realizaciones.

Diseño de una Clase (cont.)

3. Identificar atributos:
   - Se debe:
     - identificar los atributos requeridos por la clase de diseño, y
     - describir esos atributos utilizando la sintaxis de los lenguajes de programación.
   - Un atributo especifica una propiedad de una clase del diseño.
     - A menudo está implicado, y es requerido, por las operaciones de la clase.

Diseño de una Clase (cont.)

4. Identificar asociaciones y agregaciones:
   - Se deben estudiar las transmisiones de mensajes en los DS para determinar qué asociaciones son necesarias.
     - Las instancias de las asociaciones deben utilizarse:
       - para abarcar las referencias con otros objetos, y
       - para agrupar objetos en agregaciones con el propósito de enviarles mensajes.
   - Se debe:
     - Minimizar el número de asociaciones entre clases, modelando rutas de búsqueda óptimas a través de las asociaciones y agregaciones.
     - Redefinir la multiplicidad de las asociaciones, nombres de rol, clases asociativas, asociaciones n-arias, de acuerdo a lo soportado por el lenguaje de programación.
     - Redefinir la navegabilidad de las asociaciones, de acuerdo a la dirección de las transmisiones de los mensajes entre los objetos del diseño

Diseño de una Clase (cont.)

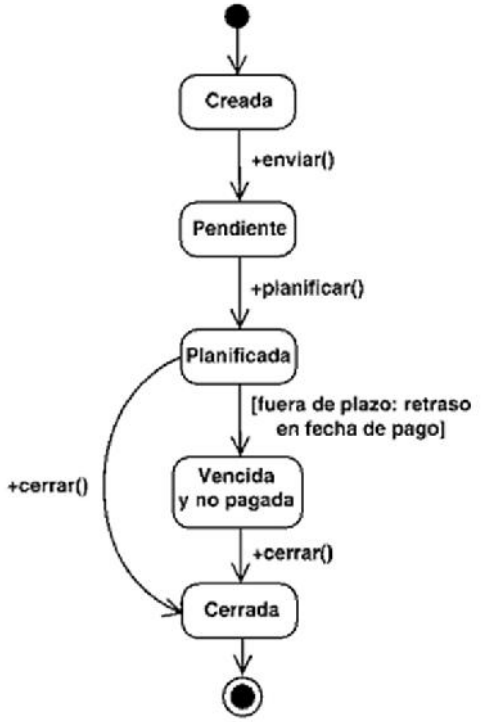
5. Identificar las generalizaciones:
   - Las generalizaciones deben ser utilizadas con la misma semántica definida en el lenguaje de programación.
     - Si no las admite, las asociaciones y/o agregaciones deben utilizarse en su lugar.

Diseño de una Clase (cont.)

6. Describir los métodos:
   - A veces, los métodos pueden utilizarse en el diseño para especificar cómo se deben realizar las operaciones.
     - Se puede utilizar lenguaje natural o pseudocódico.

Diseño de una Clase (cont.)

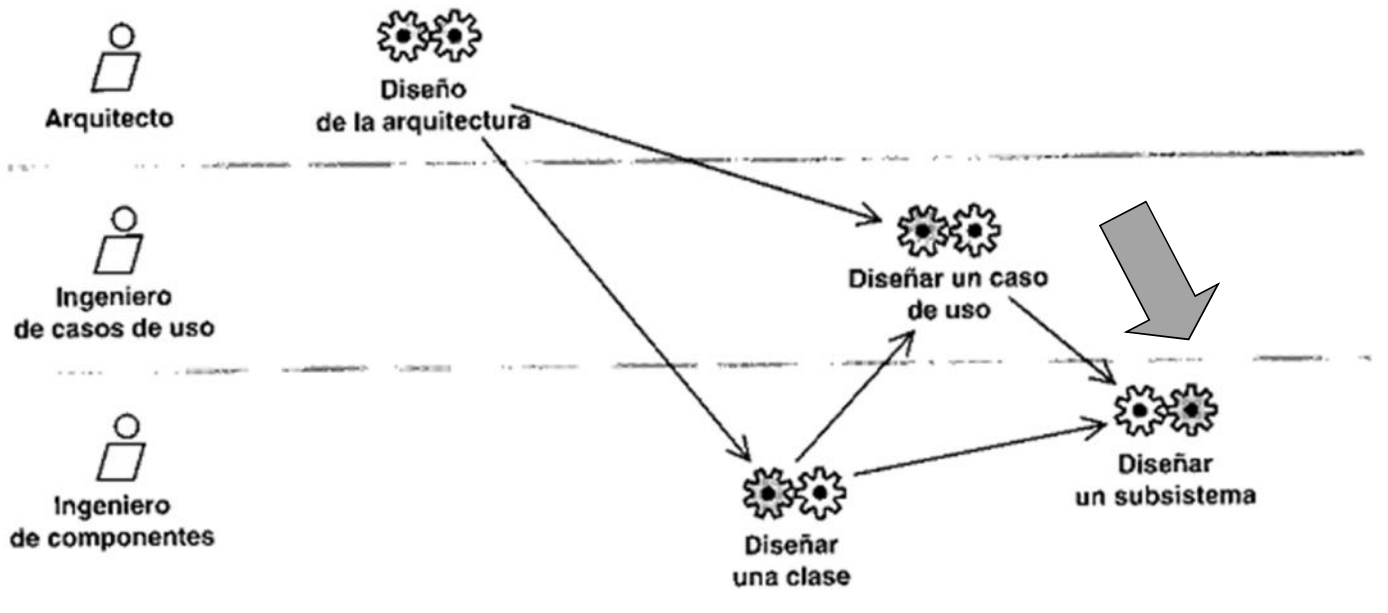
7. Describir estados:
   - Si el comportamiento de los objetos de una clase depende fuertemente del estado del objeto, se describen las diferentes transiciones de estado del objeto mediante un **diagrama de estados**.



Diseño de una Clase (cont.)

8. Tratar requisitos especiales:
   - Hay que tratar cualquier requisito que no se haya considerado en pasos anteriores, utilizando (en lo posible) los mecanismos de diseño que identificó el arquitecto.
     - Estudiar los requisitos formulados en la realización de los CU en los que la clase participa, ya que pueden establecer requisitos no funcionales para la clase del diseño.
     - Estudiar los requisitos especiales de cualquier clase del análisis que trace la clase del diseño, ya que a veces necesitan ser manejados por la clase del diseño.
     - Si se decide posponer el tratamiento de algún requisito hasta la implementación, entonces indicarlo como requisito de implementación para la clase del diseño.

Flujo de Trabajo (4)



Diseño de un Subsistema

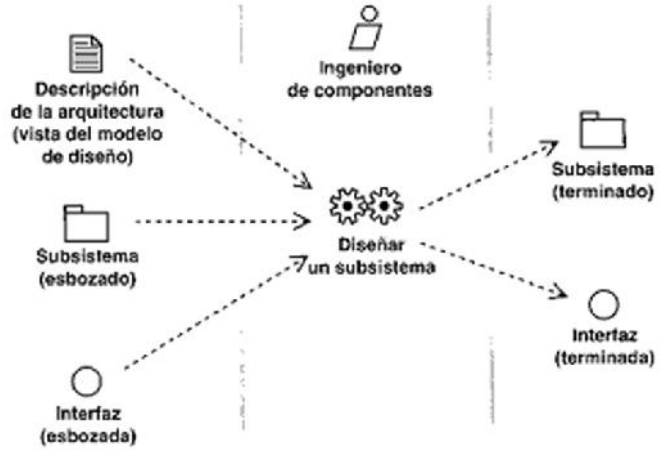
- Sus objetivos son:
  - Garantizar que el subsistema se tan independiente como sea posible de otros subsistemas y/o sus interfaces.
  - Garantizar que el subsistema proporcione las interfaces correctas.
  - Garantizar que el subsistema cumple su propósito de ofrecer una realización correcta de las operaciones tal y como se definen en las interfaces que proporciona.

Diseño de un Subsistema (cont.)

1. Mantenimiento de las dependencias entre subsistemas.
2. Mantenimiento de interfaces proporcionadas por el subsistema.
3. Mantenimiento de los contenidos de los subsistemas.

Diseño de un Subsistema (cont.)

- Entradas y resultados:



Diseño de un Subsistema (cont.)

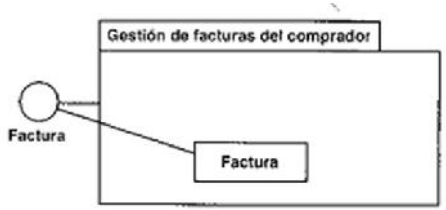
1. Mantenimiento de las dependencias entre subsistemas:
   - Deben definirse y mantenerse dependencias de un subsistema respecto a otros cuando los elementos contenidos en ellos estén asociados.
     - Si los otros subsistemas proporcionan interfaces, las dependencias de uso deberían declararse hacia las interfaces.
   - Se debe minimizar estas dependencias, tratando de reubicar las clases contenidas en un subsistema que sean demasiado dependientes de otros subsistemas.

Diseño de un Subsistema (cont.)

2. Mantenimiento de interfaces proporcionadas por el subsistema:
   - Las operaciones definidas por las interfaces que proporciona un subsistema deben soportar todos los roles que cumple el subsistema en las diferentes realizaciones del CU.
   - La técnica para mantener interfaces y sus operaciones es similar a la de mantener las clases del diseño y sus operaciones.

Diseño de un Subsistema (cont.)

3. Mantenimiento de los contenidos de los subsistemas:
   - Por cada interfaz que proporcione el subsistema, debería haber clases del diseño u otros subsistemas dentro del subsistema que también proporcione la interfaz.



   - Se pueden crear colaboraciones (en términos de los elementos contenidos en el subsistema) para clarificar cómo el diseño interno de un subsistema realiza cualquiera de sus interfaces o CU.

Del Diseño a la Implementación y Pruebas

- Los subsistemas del diseño y los subsistemas de servicio se implementarán mediante subsistemas de implementación que contienen los verdaderos componentes (archivos de código fuente, scripts, binarios, ejecutables,…).
  - Los subsistemas de implementación poseen una traza 1-1 hacia los subsistemas del diseño.

Del Diseño a la Implementación y Pruebas (cont.)

- Las clases del diseño se implementarán mediante componentes de archivo que contendrán el código fuente.
  - Dependiendo del lenguaje de programación, generalmente se implementan varias clases del diseño dentro de un mismo componente de archivo.

Del Diseño a la Implementación y Pruebas (cont.)

- Las realizaciones de CU-diseño se utilizarán al planificar y llevar a cabo el esfuerzo de implementación en pasos pequeños y manejables.
  - En cada uno se implementará principalmente un conjunto de realizaciones de CU o partes de ellos.

Del Diseño a la Implementación y Pruebas (cont.)

- Las clases activas del diseño que representen procesos pesados se utilizarán como entrada cuando se identifiquen los componentes ejecutables.

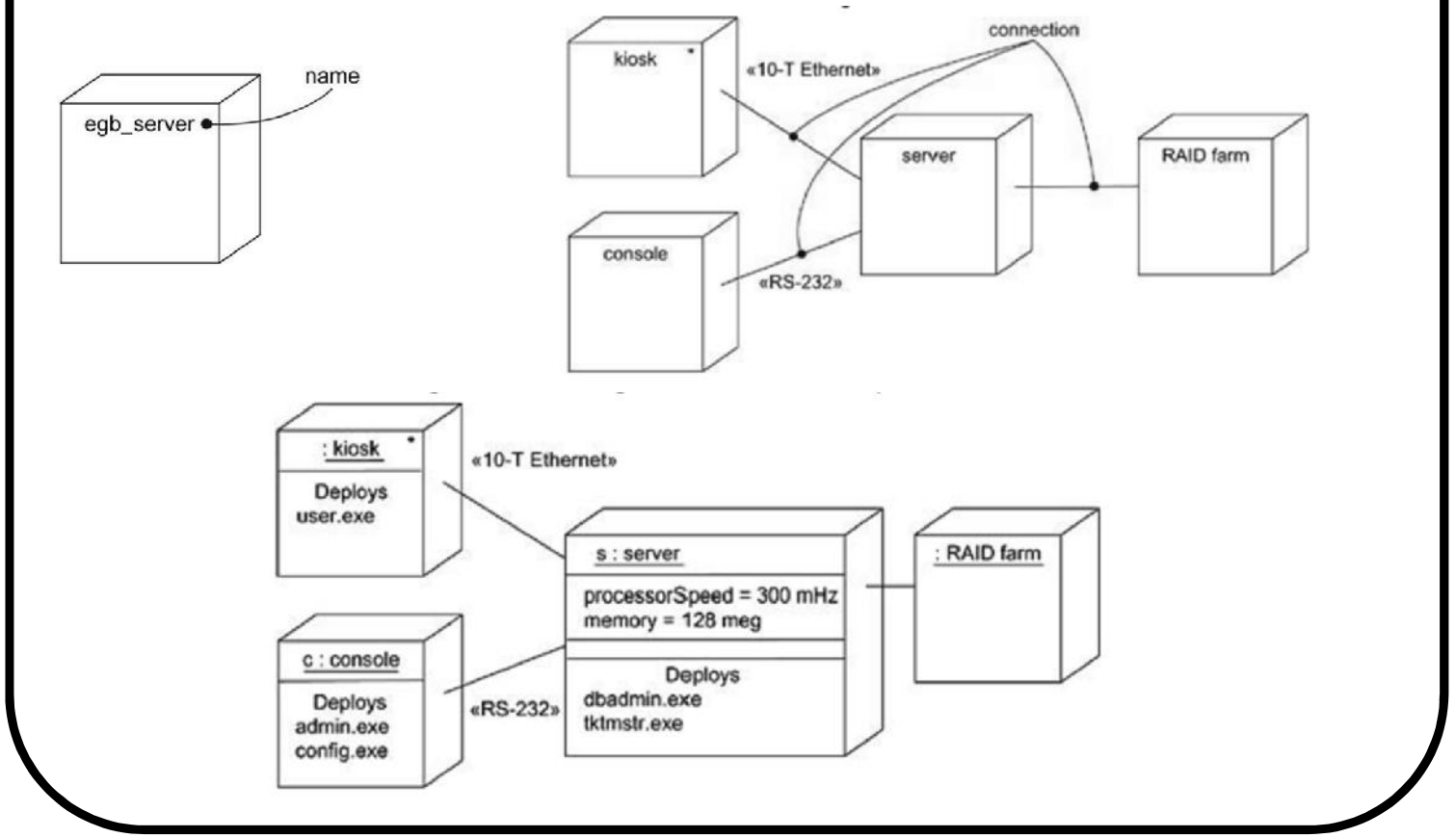
- El modelo de despliegue y las configuraciones de red se utilizarán al distribuir el sistema instalando componentes ejecutables en los nodos.

Diagramas de Despliegue

Diagramas de Despliegue

- Un diagrama de despliegue muestra la disposición física de los distintos nodos (recursos materiales) que entran en la composición de un sistema y el reparto de los programas ejecutables entre estos.
- Cada nodo se representa por un cubo.
  - Evoca la presencia física del equipo en el sistema.
  - Puede incluir el nombre de los procesos que alberga.
    - Cada proceso con nombre ejecuta un programa principal del mismo nombre.
- Los nodos se conectan entre sí por líneas que simbolizan un soporte de comunicación.
  - Se puede colocar la naturaleza de los enlaces de comunicación.
- Un diagrama de despliegue puede mostrar clases de nodos o instancias de nodos.

Diagramas de Despliegue



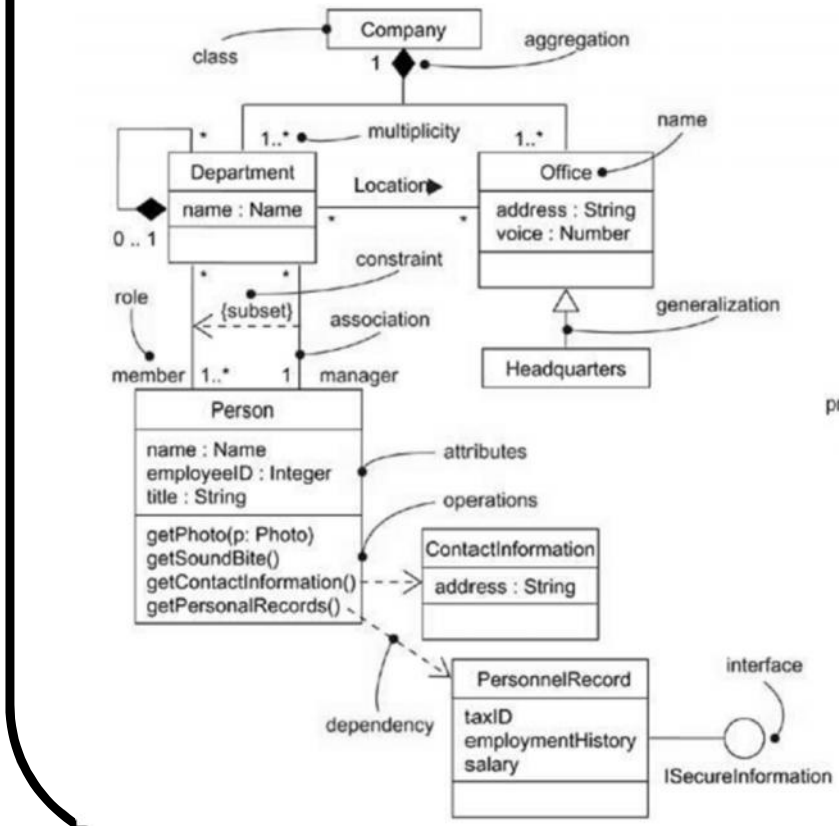
Diagramas de Clases del Diseño

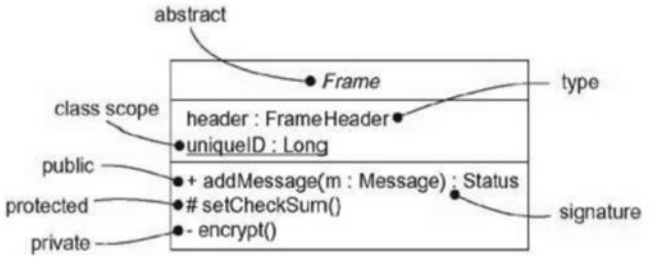
Diagramas de Clases del Diseño

- Un diagrama de clases del diseño describe gráficamente las especificaciones de las clases software y de la interfaz en una aplicación.
- Contiene:
  - Clases, asociaciones y atributos.
  - Interfaces, con sus operaciones y constantes.
  - Métodos.
  - Información sobre los tipos de los atributos.
  - Navegabilidad.
  - Dependencias.

Diagramas de Clases del Diseño

(cont.)





Diagramas de Interacción

Diagramas de Interacción

- Muestran interacciones entre los objetos.
- Existen dos tipos:
  - Diagramas de secuencia.
  - Diagramas de colaboración.

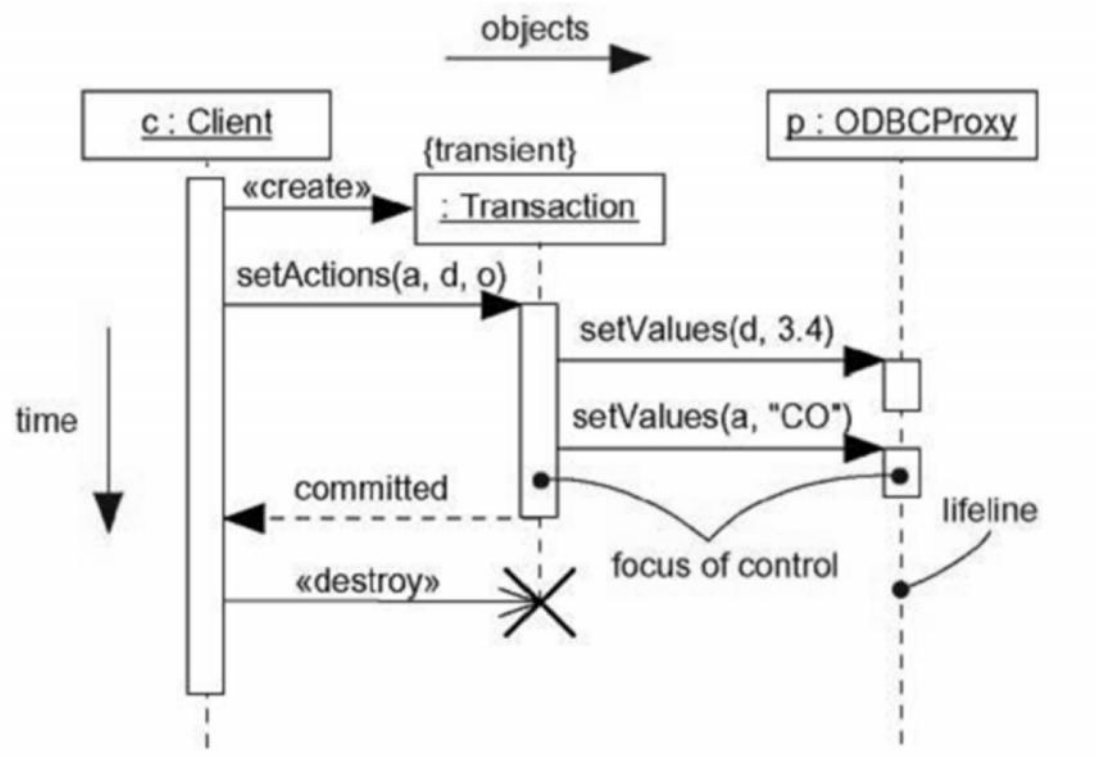
	Ventajas	**Desventajas**
Secuencia	Muestra claramente la secuencia u ordenación en el tiempo de los mensajes. Notación simple.	Fuerza a extender por la derecha cuando se añaden nuevos objetos; consume espacio horizontal.
Colaboración	Economiza espacio; flexibilidad al añadir nuevos objetos en dos dimensiones. Es mejor para ilustrar bifurcaciones complejas, iteraciones y comportamiento concurrente.	Difícil de ver la secuencia de mensajes. Notación más compleja.

Diagramas de Secuencia

- Los diagramas de secuencia (DS) muestran interacciones entre objetos según un punto de vista temporal.

- Consta de una tabla en donde:
  - Eje x: objetos
  - Eje y: mensajes (en el tiempo)

Diagramas de Secuencia (cont.)

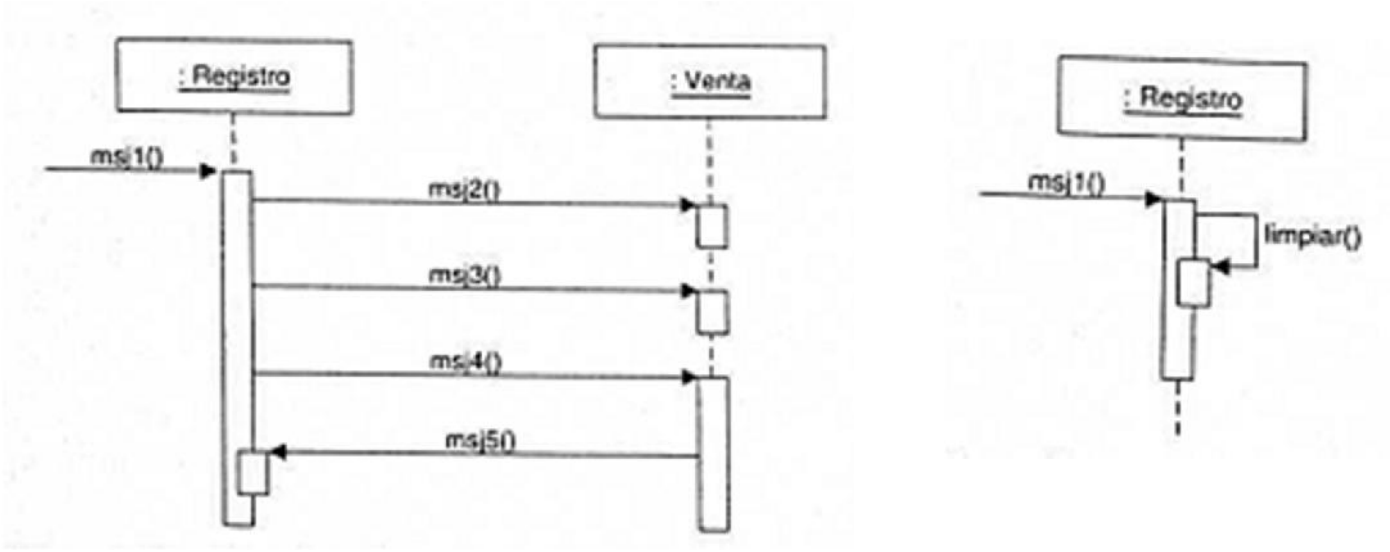


Diagramas de Secuencia (cont.)

- A la izquierda se coloca el objeto que inicia la interacción.
- Cada objeto se muestra con:
  - un rectángulo, y
  - una barra vertical discontinuada (línea de vida) que representa la existencia del objeto.
- Los mensajes se:
  - colocan de arriba hacia abajo según el orden temporal, y
  - muestran con una flecha horizontal, del emisor al receptor.
  - Sintaxis:

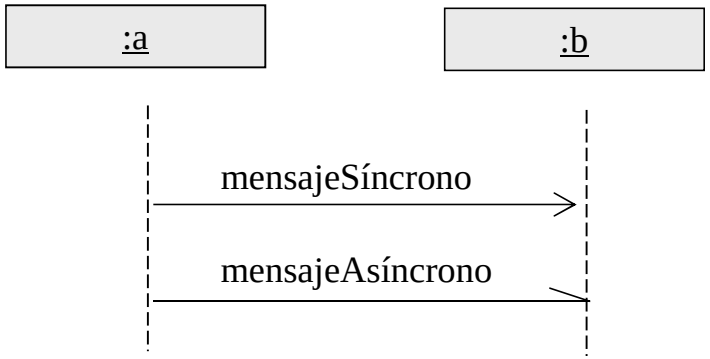
    retorno := **mensaje** (parámetro : tipo parámetro) : tipo retorno
- Cada objeto puede presentar períodos de actividad (foco de control).
  - Corresponde al período de tiempo en que un objeto ejecuta una acción, bien directamente o mediante un procedimiento subordinado.
  - Se representa por un rectángulo delgado sobre la línea de vida.
    - El inicio y el fin del rectángulo corresponden al inicio y fin del período de actividad.

Diagramas de Secuencia (cont.)



Diagramas de Secuencia (cont.)

- Los envíos de mensajes pueden ser:
  - Síncronos
    - El emisor está bloqueado y espera que el receptor haya terminado de tratar el mensaje.
    - Se representan con una flecha.
  - Asíncronos
    - El emisor no está bloqueado y puede continuar su ejecución.
    - Se representan con media flecha.



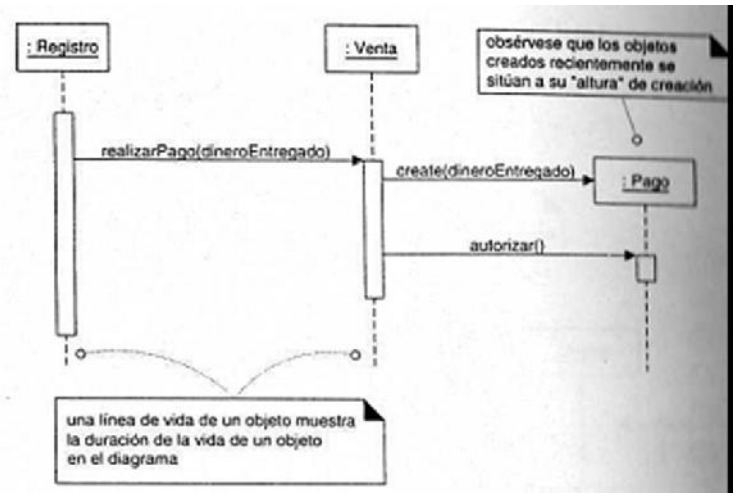
Diagramas de Secuencia (cont.)

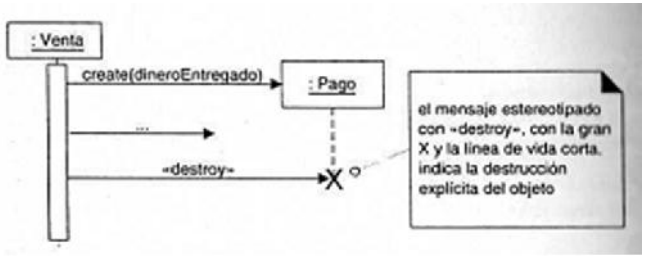
- El retorno de una operación puede ser:
  - Implícito:
    - No es necesario mostrar el retorno en los mensajes síncronos.
      - El retorno es implícito al fin de la actividad.
    - Si se quiere mostrar, se utiliza una flecha punteada desde receptor al emisor.
  - Explícito:
    - Se representa con una flecha desde el receptor al emisor.
    - Es necesario en los mensajes asíncronos.
      - El retorno debe materializarse cuando existe.

- Si se llama a una operación de clase o estática, en el eje de los objetos de coloca un rectángulo para la clase (sin : ni subrayar).

Diagramas de Secuencia (cont.)

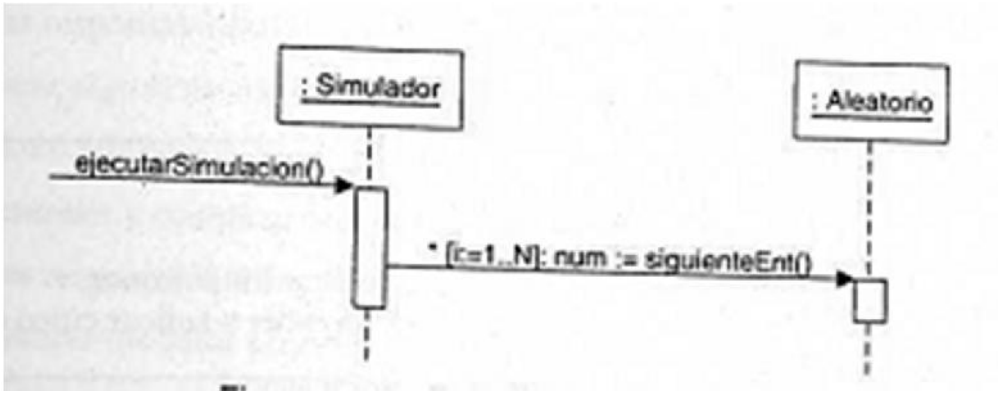
- La creación de un objeto se muestra haciendo apuntar el mensaje de creación sobre el rectángulo del objeto creado.
- La destrucción de un objeto se muestra con una **X** sobre la línea de vida del objeto, a la altura del mensaje que causa la destrucción o a la altura en que el propio objeto se destruye.





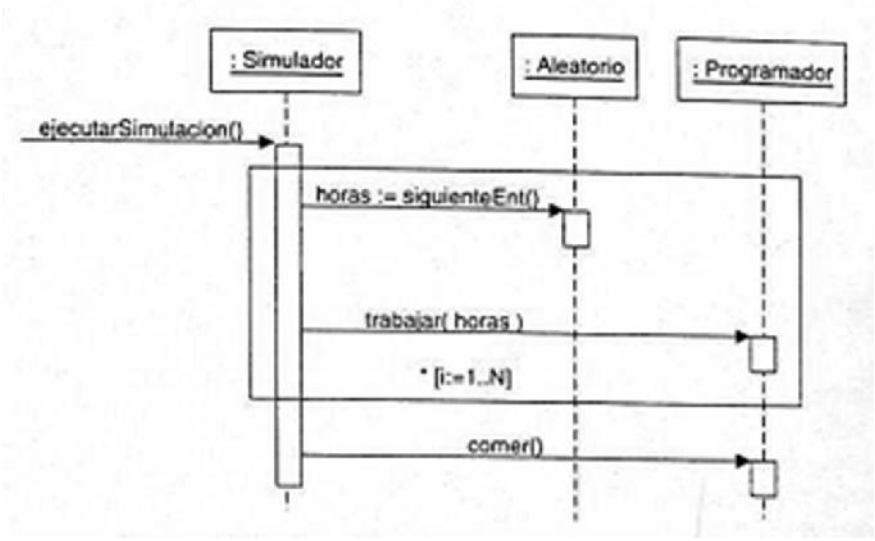
Diagramas de Secuencia (cont.)

- La iteración de un mensaje se representa mediante la condición de repetición con * y entre [ ] colocada delante del mensaje.



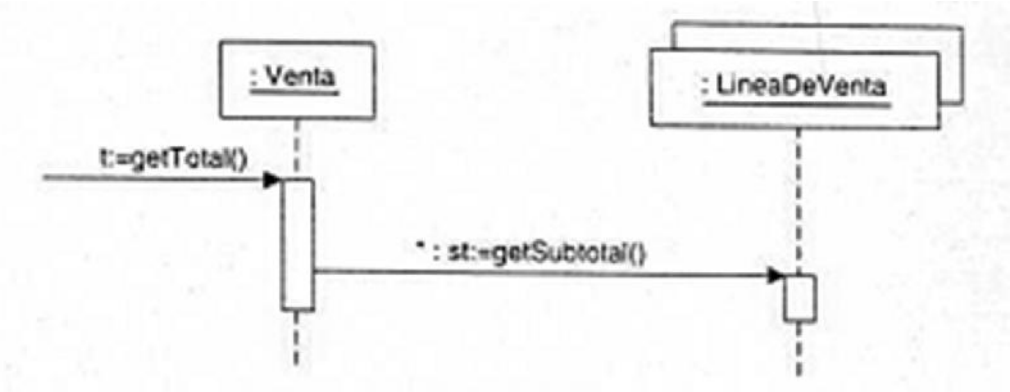
Diagramas de Secuencia (cont.)

- La iteración de una secuencia de mensajes se representa mediante:
  - un rectángulo que contiene a la secuencia de mensajes a repetir, y
  - en su parte inferior se coloca la condición de repetición con * y entre [ ].



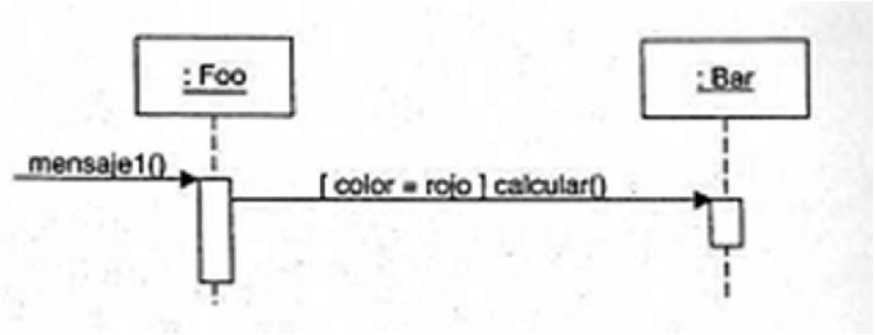
Diagramas de Secuencia (cont.)

- La iteración sobre una colección se representa mediante:
  - * colocado delante del mensaje, y
  - la colección se indica con múltiples rectángulos en el eje de los objetos.



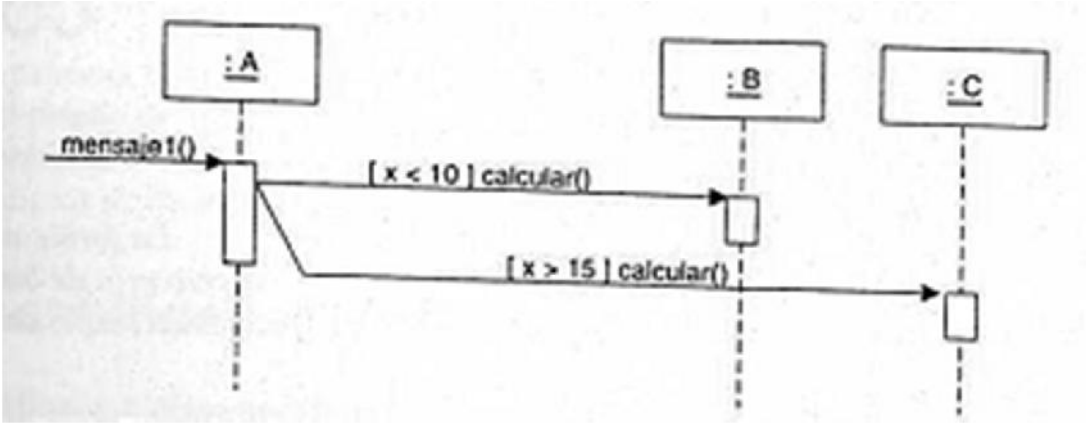
Diagramas de Secuencia (cont.)

- Un mensaje condicional se muestra mediante la condición entre [ ] colocada delante del mensaje.



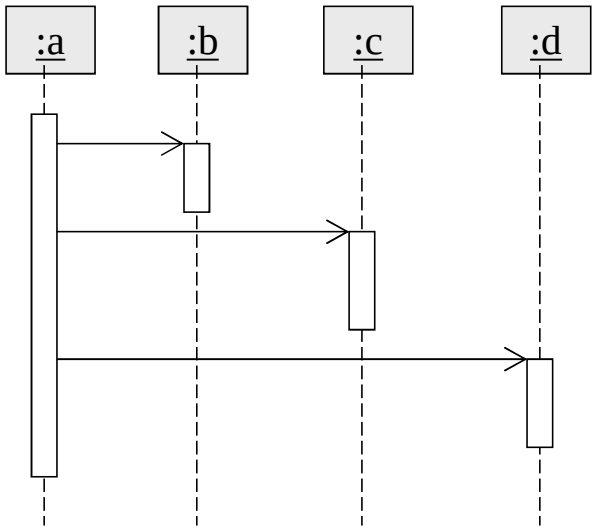
Diagramas de Secuencia (cont.)

- Las bifurcaciones condicionales (mutuamente exclusivas) se muestran mediante:
  - las condiciones entre [ ] colocadas delante de los mensajes, y
  - las diferentes ramas se muestran como flechas que tienen su origen en el mismo instante.

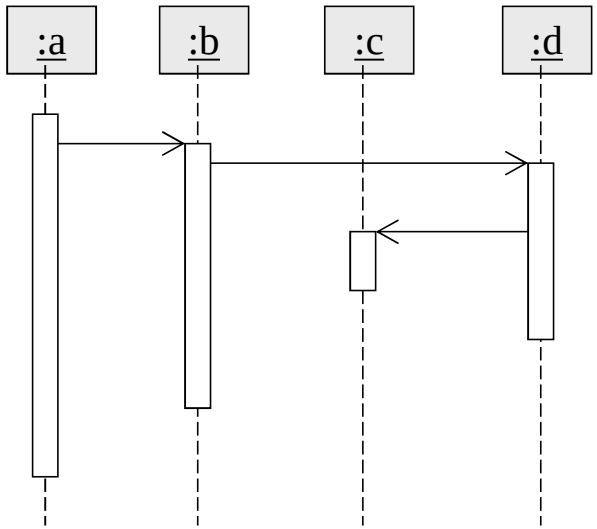


Diagramas de Secuencia (cont.)

- Estructuras de control:
  - La forma del DS refleja indirectamente las opciones de estructura.



Control centralizado



Control descentralizado

Diagramas de Secuencia (cont.)

- Identificación de las interacciones entre objetos del diseño al diseñar un CU:
  a) Estudiar la correspondiente realización del CU-análisis.
     - Obtener un esbozo de la secuencia necesaria entre los objetos del diseño.
  b) Crear el DS comenzando por el principio del flujo del CU.
  c) Seguir ese flujo paso a paso, decidiendo qué objetos del diseño y qué interacciones de instancias de actores se necesitan en cada paso.

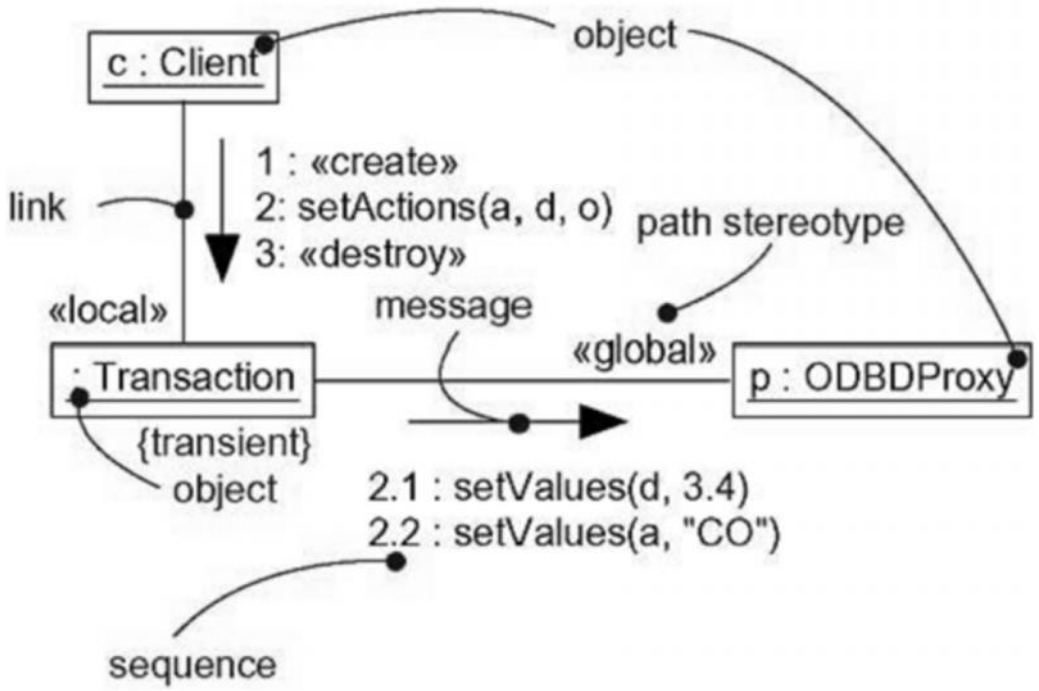
Diagramas de Secuencia (cont.)

- Identificación de las interacciones entre objetos del diseño al diseñar un CU (cont.):
  - El causante de la invocación del CU es un mensaje de una instancia de un actor hacia un objeto del diseño.
  - Cada clase del diseño identificada en la realización del CU debería tener al menos un objeto del diseño participante en el DS.
  - Los mensajes que realizan el CU, se envían entre líneas de vida de los objetos.
    - Un mensaje puede tener un nombre temporal que pasará a ser el nombre de una operación cuando sea identificado por el ingeniero de componentes.
  - Es importante comprender el orden cronológico de las transferencias de mensajes entre objetos.
  - Se utilizan etiquetas y flujo de eventos para complementar el DS.
  - El DS debería tratar todas las relaciones del CU que realiza.
  - Si el CU tiene varios flujos o subflujos distintos se crea un DS para cada uno.

Diagramas de Colaboración

- Los diagramas de colaboración (DC) muestran interacciones entre objetos según un punto de vista de la estructura espacial estática.
- Expresan a la vez:
  - el contexto de un grupo de objetos (a través de objetos y enlaces), y
  - la interacción entre los objetos (mediante el envío de mensajes).
- La distribución de los objetos en el DC permite observar adecuadamente la interacción de un objeto con respecto de los demás.
- Consta de nodos (objetos) y arcos (enlaces).

Diagramas de Colaboración (cont.)



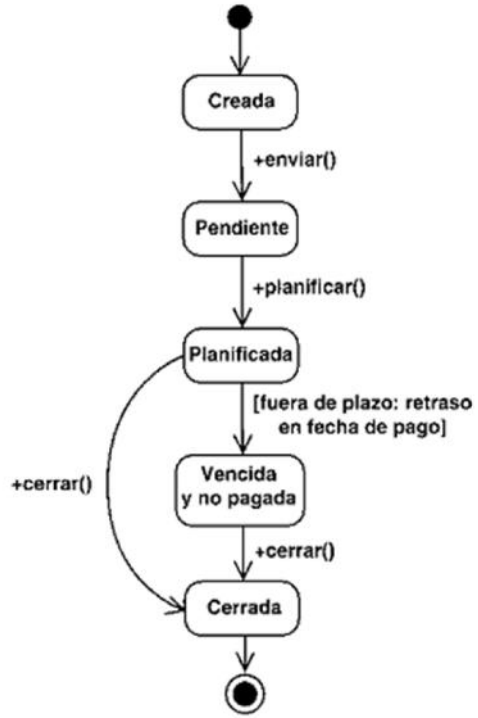
Diagramas de Estados

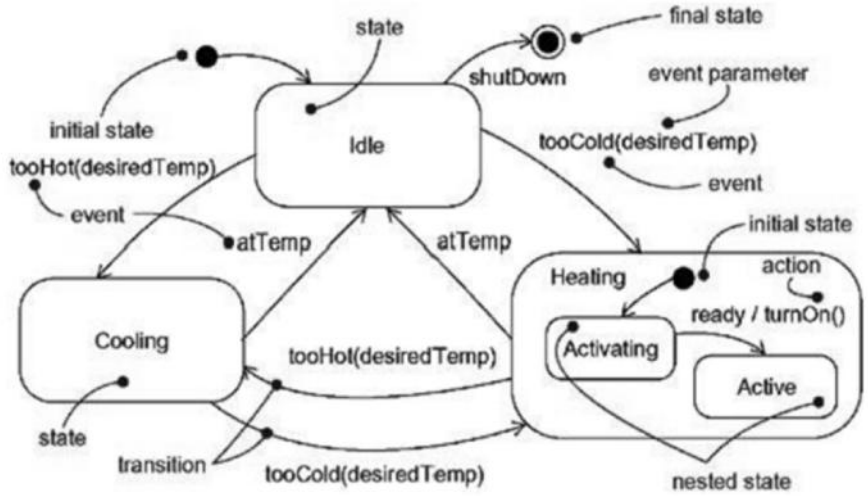
Diagramas de Estados

- Un diagrama de estado (DE) muestra el comportamiento de los objetos de una clase, en términos de estados y eventos, mediante un autómata de estados finito vinculado a la clase.

- Un DE es un grafo dirigido.

Diagramas de Estados (cont.)





Diagramas de Estados (cont.)

- Cada objeto está en un momento dado en un estado particular.
  - Un estado es siempre la imagen de la conjunción instantánea de los valores contenidos por los atributos del objeto y la presencia o no de enlaces (del objeto hacia otros objetos).

- El vínculo entre las operaciones definidas en la clase y los eventos que aparecen en el DE se efectúa por medio de acciones y actividades.
  - Una acción / actividad corresponde a una de las operaciones declaradas en la clase del objeto.
  - Una acción puede tener acceso a los parámetros del evento y a los atributos del objeto.
  - Una actividad puede tener acceso a los atributos del objeto.

Patrones GRASP

UML y Patrones
C. Larman

Responsabilidad

- "*Contrato u obligación de un tipo o clase*".
- Las responsabilidades:
  - se relacionan con las obligaciones de un objeto respecto a su comportamiento, y
  - se asignan a los objetos durante el DOO.
- Su granularidad influye en su traducción a clases y métodos.
- Responsabilidad ≠ Método
  - Los métodos se ponen en práctica para cumplir con las responsabilidades.
  - Las responsabilidades se implementan usando métodos que operen solos o en colaboración con otros métodos y objetos.

Responsabilidad (cont.)

- Responsabilidades de ***conocer*** de un objeto:
  - Conocer los datos privados encapsulados.
  - Conocer los objetos relacionados.
  - Conocer las cosas que puede derivar o calcular.
  - ...
- Responsabilidades de ***hacer*** de un objeto:
  - Hacer algo con él mismo (crear un objeto, calcular algo…)
  - Iniciar una acción en otros objetos.
  - Controlar y coordinar actividades en otros objetos.
  - ....

Patrones GRASP

- Son patrones de asignación de responsabilidades (**G**eneral **R**esponsability **A**ssignment **S**oftware **P**atterns).
  - Un sistema orientado a objetos se compone de objetos que envían mensajes a otros objetos para que lleven a cabo operaciones.
  - Las decisiones de diseño poco acertadas dan origen a sistemas y componentes frágiles y difíciles de mantener, entender, reutilizar o extender.
- Recogen principios/directrices que se aplican al preparar los diagramas de interacción y/o asignar responsabilidades a los objetos, para obtener un buen DOO.

Patrones GRASP (cont.)

- Experto en Información
- Creador
- Bajo Acoplamiento
- Alta Cohesión
- Controlador

Experto en Información

- Problema:

  ¿Cuál es el principio fundamental por el cual se asignan las responsabilidades en el DOO?

- Solución:

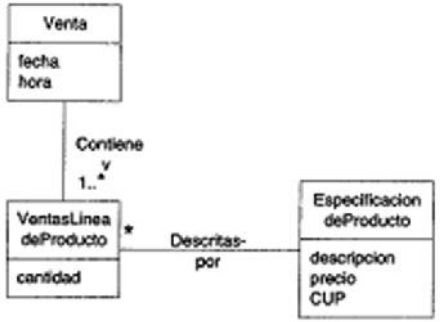
  Asignar una responsabilidad al experto en información (la clase que cuenta o conoce la información necesaria para cumplir la responsabilidad).

Experto en Información (cont.)

- Es el más usado.
  - Expresa la “intuición” común que los objetos hacen las cosas relacionadas con la información que tienen.
- El cumplimiento de una responsabilidad frecuentemente requiere información distribuida en varias clases => puede haber expertos “parciales”.
- Beneficios:
  - Se conserva el encapsulamiento.
    - Favorece el bajo acoplamiento (sistemas más robustos y de fácil mantenimiento).
  - Promueve clases sencillas y cohesivas, más fáciles de mantener y comprender.
    - Favorece la alta cohesión.

Experto en Información (cont.)

- ¿Quién es el responsable de calcular el total de una venta?



Clase	Responsabilidad
Venta	conoce el total de la venta
VentasLineadeProducto	conoce el subtotal de la línea de producto
EspecificaciondeProducto	conoce el precio del producto

Creador

- Problema:

  ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

- Solución:

  La clase B es responsable de crear una instancia de la clase A en uno de los siguientes casos:

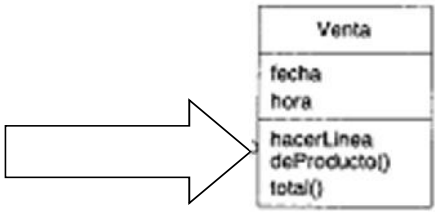
  - B agrega los objetos A (es una agregación de).
  - B contiene los objetos A.
  - B registra a las instancias de los objetos A.
  - B utiliza específicamente los objetos de A.
  - B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado (B es un experto respecto a la creación de A).

Creador (cont.)

- Si existe más de una opción, optar por la clase B que agregue o contenga a la clase A.
- Guía la asignación de responsabilidades relacionadas con la creación de objetos.
- El propósito fundamental es encontrar un creador que se deba conectar con el objeto producido.
- Beneficios:
  - Proporciona un bajo acoplamiento.
    - Menos dependencias respecto al mantenimiento.
    - Mayores oportunidades de reuso.

Creador (cont.)

- ¿Quién debería encargarse de ordenar la creación de una instancia de VentasLineadeProducto?



Bajo Acoplamiento

- Problema:

  ¿Cómo soportar bajas dependencias, bajo impacto del cambio e incremento en la reutilización?

- Solución:

  Asignar una responsabilidad de manera que el acoplamiento permanezca bajo.

Bajo Acoplamiento (cont.)

- Principio evaluativo para mejorar el diseño.
  - Es un principio a tener en cuenta en todas las decisiones de diseño.
  - El nivel de acoplamiento no se puede considerar de manera aislada a Experto o Alta Cohesión.
- Beneficios:
  - Clases más independientes:
    - No afectan los cambios en otros componentes.
    - Fácil de entender de manera aislada.
    - Conveniente para reutilizar.

Alta Cohesión

- Problema:

  ¿Cómo mantener la complejidad dentro de límites manejables?

- Solución:

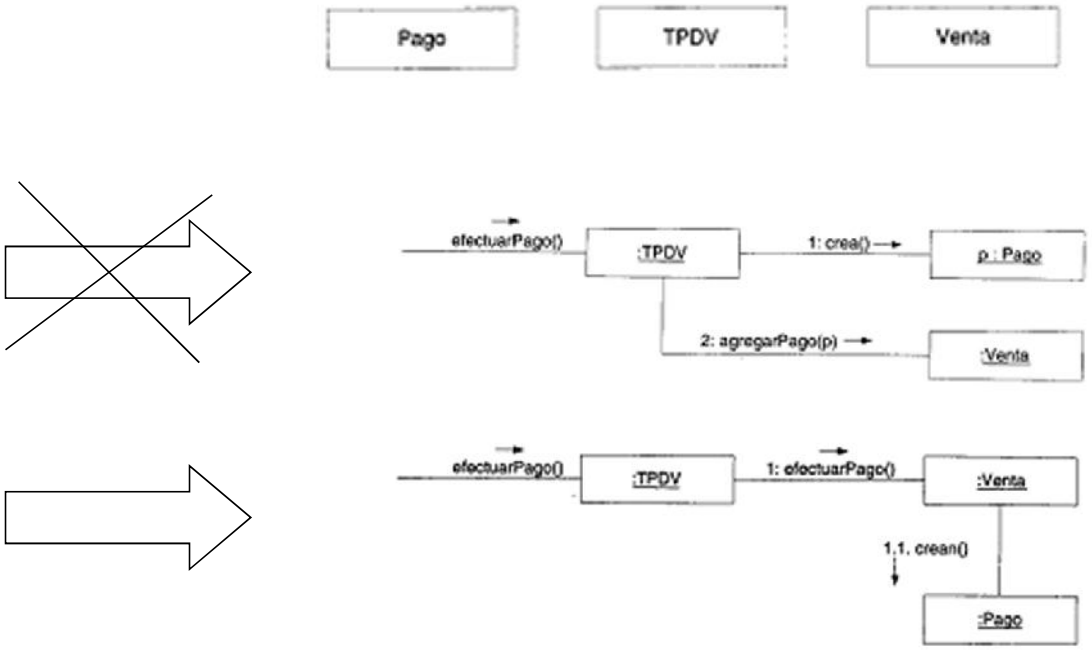
  Asignar una responsabilidad de modo que la cohesión siga siendo alta.

Alta Cohesión (cont.)

- Principio evaluativo para mejorar el diseño.
  - Es un principio a tener en cuenta en todas las decisiones de diseño.
  - El nivel de cohesión no se puede considerar de manera aislada a Experto o Bajo Acoplamiento.
- Una clase con alta cohesión colabora con otros objetos para compartir el esfuerzo si la tarea es grande.
- Beneficios:
  - Se incrementa la claridad y facilita la comprensión del diseño.
  - Se simplifica el mantenimiento y las mejoras en la funcionalidad.
  - Se incrementa la reutilización.
  - A menudo se genera un bajo acoplamiento.

Bajo Acoplamiento y Alta Cohesión

- ¿Quién es el responsable de crear una instancia Pago y asociarla a Venta?



Controlador

- Problema:

  ¿Quién debe ser el responsable de gestionar un evento de entrada al sistema?

- Solución:

  Asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase que represente una de las siguientes opciones:

  - El sistema global, dispositivo o subsistema (controlador de fachada).
  - Un escenario de CU en el que tiene lugar el evento del sistema, generalmente denominado “<NombreCU> Manejador / Coordinador / Sesión” (controlador de sesión o de CU).

Controlador (cont.)

- Un evento del sistema:
  - es un evento generado por un actor externo, y
  - se asocia con operaciones del sistema (respuesta a los eventos).
- Un controlador:
  - es un objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema, y
  - define el método para la operación del sistema.
- En el mismo CU es conveniente usar la misma clase controlador con todos los eventos del sistema.
- Normalmente un controlador debería delegar en otros objetos el trabajo que se necesita hacer.
  - Sólo coordina o controla la actividad.
  - No realiza mucho trabajo por sí mismo.
- Beneficios:
  - Aumenta el potencial para reutilizar y las interfaces conectables (pluggable).
  - Razonar sobre el estado de los CU.

Controlador (cont.)

- ¿Quién es el responsable de controlar a introducirProducto?