Escuela Politécnica Superior

UAM

Introducción al Análisis y Diseño Orientado a Objetos

Tema 3

TACC II
Curso 2008/09

Motivación

- Un proyecto software **no consiste sólo en programar**.
- Necesitamos saber cuáles son las necesidades del cliente.
  - Identificar los requisitos, anotarlos, analizarlos, validarlos.
- Necesitamos diseñar una solución, y hacer “los planos” del software:
  - Diseño de la arquitectura, detallado, de datos, …
- Hay que asegurarse de que el software funciona:
  - Pruebas de unidad (a nivel de método y clase), de integración, del sistema, de aceptación, etc.
- Hay que mantener el software.
  - Documentación (de cada una de las fases), coherencia entre los productos de las distintas fases (ej. código vs. diseños).

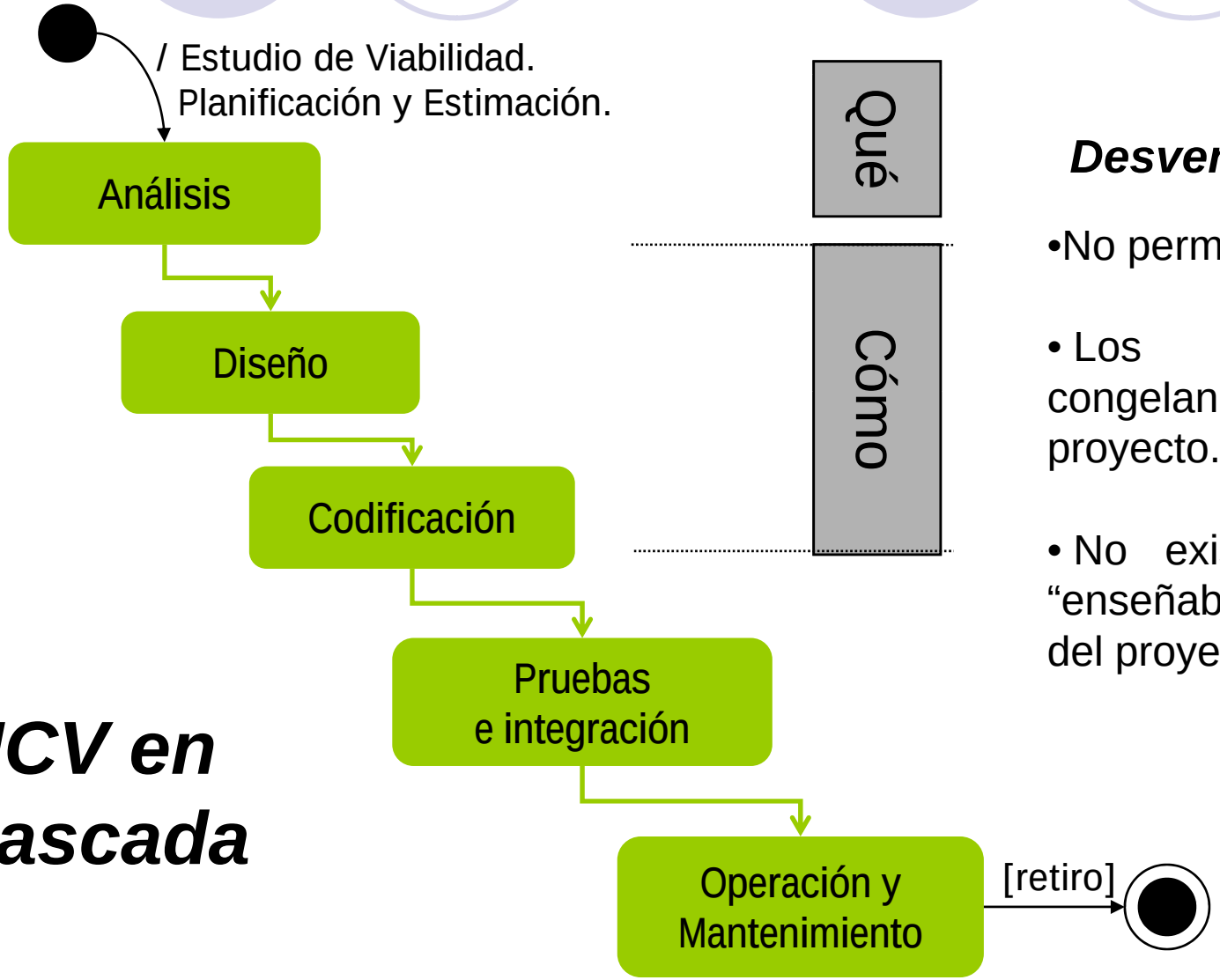
Indice

- **Modelos de Ciclo de Vida.**
- Análisis y Diseño Orientado a Objetos.
- Notaciones.
- Metodologías.

Modelos de ciclo de vida

- ¿Qué estrategia seguimos para organizar las fases de un proyecto?.
- Un modelo de ciclo de vida nos guia en las fases que hay que realizar, sus entradas y salidas, y los criterios de transición.
- La elección de uno u otro depende de las características del proyecto.
- Con teconologías orientadas a objetos se tiende a ciclos de vida iterativos e incrementales.

Modelos de Ciclo de Vida

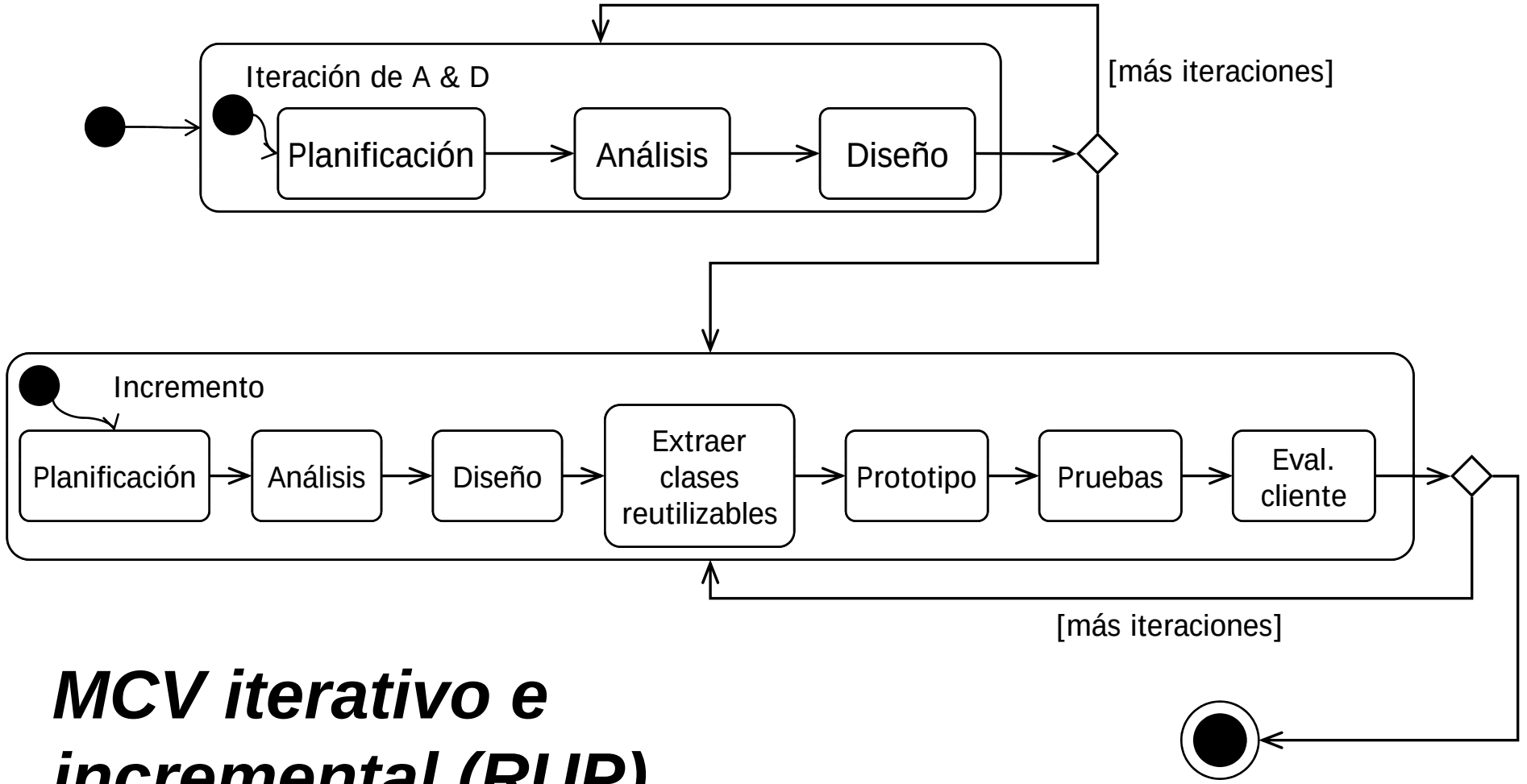


MCV en Cascada

Desventajas:

- No permite iteraciones.
- Los requisitos se congelan al principio del proyecto.
- No existe un proyecto "enseñable" hasta el final del proyecto.

Modelos de Ciclo de Vida



MCV iterativo e incremental (RUP)

Indice

- Modelos de Ciclo de Vida.
- **Análisis y Diseño Orientado a Objetos.**
  - **Ventajas e inconvenientes.**
  - **Análisis.**
  - **Diseño.**
- Notaciones.
- Metodologías.

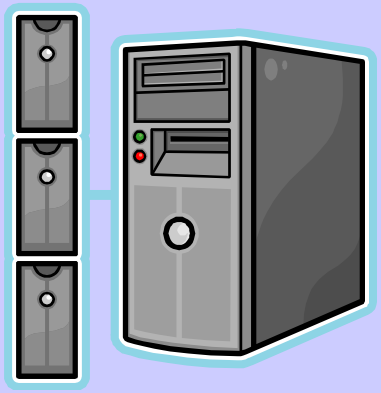
Análisis y Diseño

Problema vs. Solución

Análisis (qué)

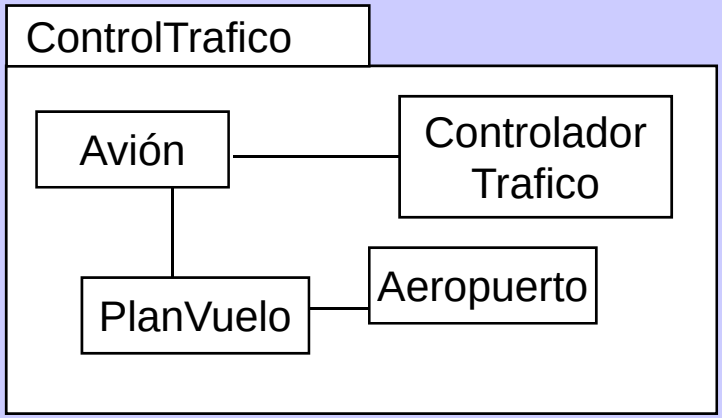
Diseño (cómo)

Dominio del problema

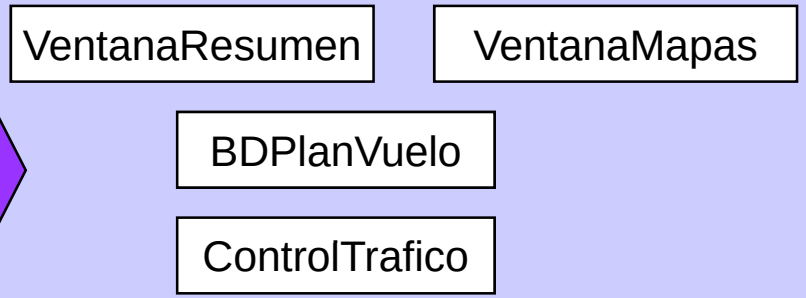


Dominio de la Solución

Modelo del Dominio del Problema



Modelo del Dominio de la Solución



Paradigma de Orientación a Objetos

Características

- Diseños modulares.
- Efectos laterales mínimos(encapsulamiento)
- Extensibilidad.
- Fácil de modificar.
- Orientado a datos.
- Explota la herencia (jerárquico).
- Reutilización de clases.

Ventajas

- Módulos con fuerte cohesión interna y escaso acoplamiento externo (sin variables globales, …)
- Facilita el funcionamiento en entorno multiprocesador (objetos distribuidos)
- Correspondencia directa con el mundo real
- Prototipos rápidos
- Herramientas y bibliotecas muy amplias
- Aplicaciones construidas enganchando objetos
- Mejor comprensión y mantenimiento
- Apropiado para aplicaciones dirigidas por eventos.

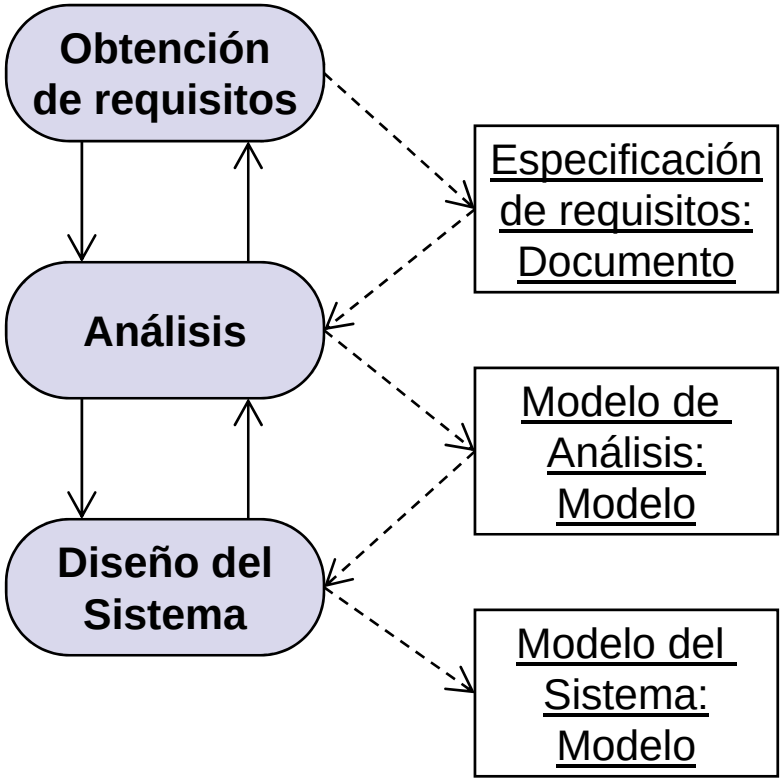
Inconvenientes

- Impactos desfavorables sobre espacio y tiempo de ejecución.
- Forma de pensar diferente: curva de aprendizaje lenta.
- Herencia y ligadura dinámica dificultan las pruebas.
- Difícil seguir el flujo de ejecución (e.j. llamádas implicitas a constructores, conversiones implícitas, etc.)
- Frameworks grandes y complicados (e.j. MFCs).

Análisis Orientado a Objetos

- Centrarse en el “qué”.
- Identificar los requisitos: documentos de análisis.
  - Entrevistas.
  - Identificar requisitos funcionales y no funcionales (ej.: rendimiento, fiabilidad)
- Especificar los requisitos: documento de especificación de requisitos.
  - Documento técnico. Organización y clasificación de los requisitos.
- Analizar: Modelos de análisis.
  - Estudio de posibles escenarios: casos de uso.
  - Otras técnicas: fichas CRC, orientados al flujo, etc.
- Validar.

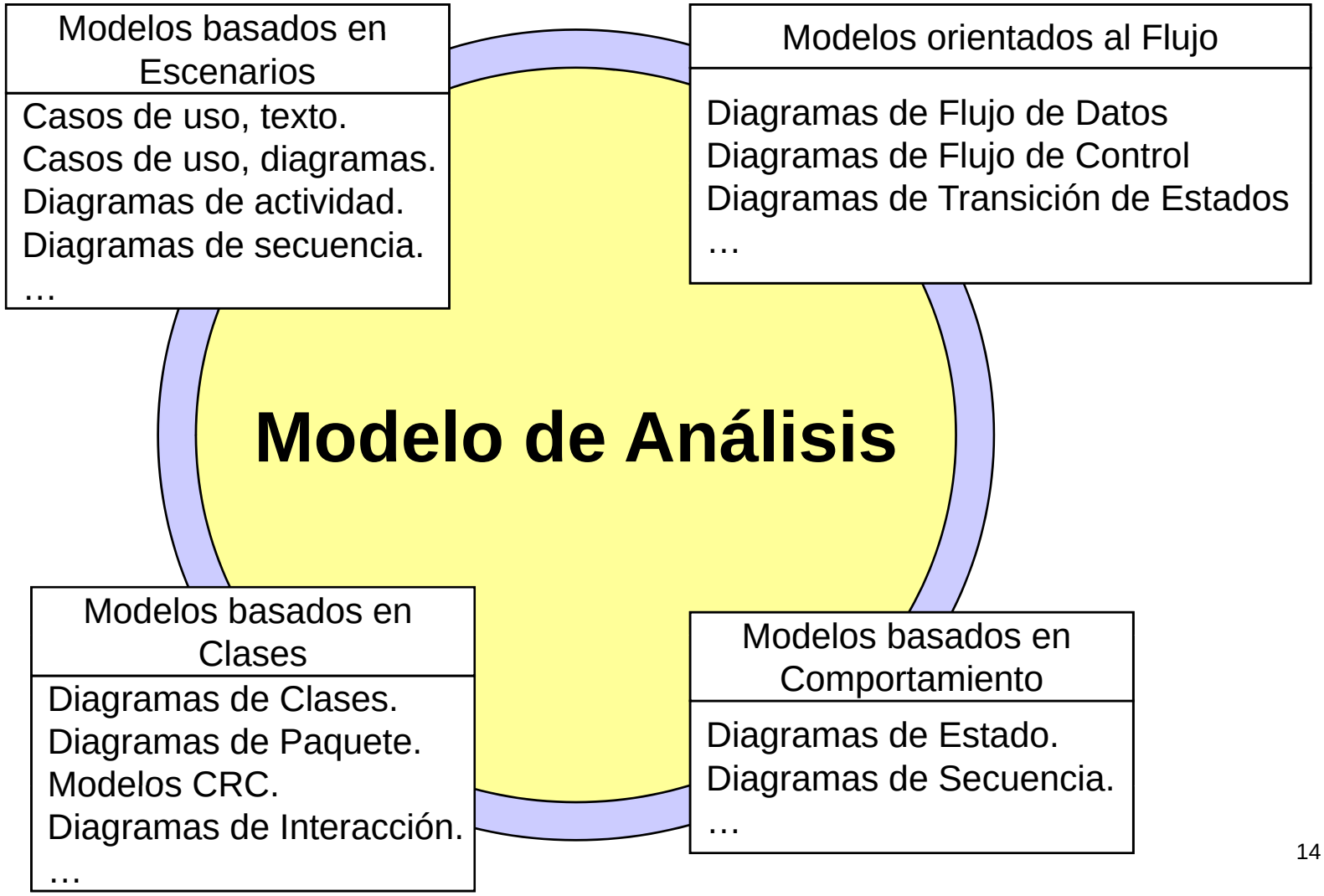
Análisis Orientado a Objetos



- La especificación de requisitos describe el sistema, en lenguaje natural.
- Sirve de comunicación entre desarrolladores y clientes, “contrato”.
- El modelo de análisis usa notación formal (ej.: Z, Alloy) o semi-formal (ej.: UML).
- Sirve de comunicación entre desarrolladores.

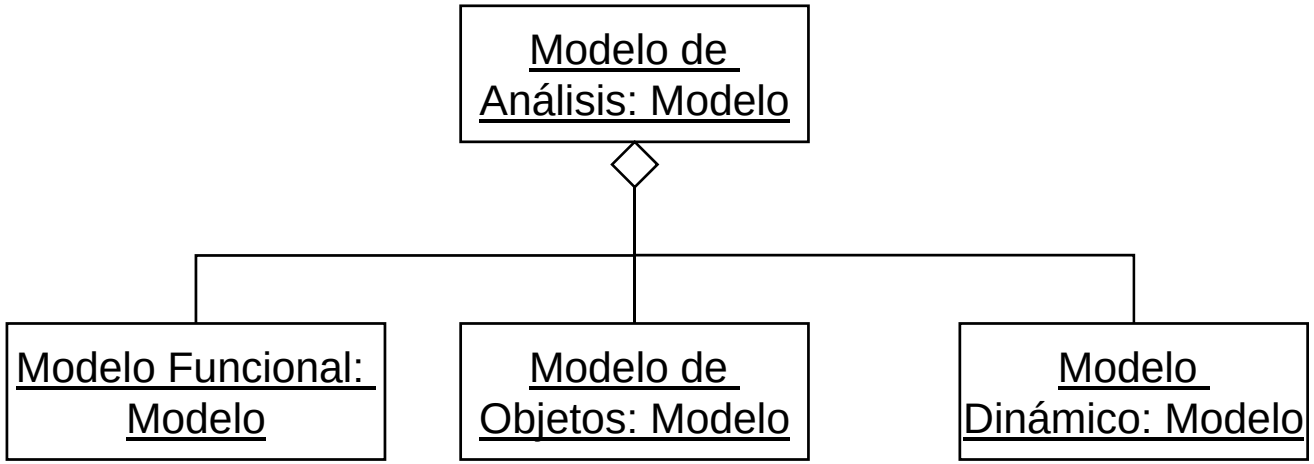
Análisis Orientado a Objetos

Modelos de Análisis



Análisis Orientado a Objetos

Modelos de Análisis. Basado en Escenarios.



- Modelo funcional: casos de uso y escenarios.
- Modelo de Objetos: diagramas de clases y objetos.
- Modelo dinámico: diagramas de secuencia,…

Casos de uso

- Describen **qué** hace el sistema desde el punto de vista de un observador externo.
- Actores: ¿quién interactúa con el sistema?. También pueden ser otros sistemas.
- Un escenario es un ejemplo de lo que ocurre cuando uno o varios actores interactúan con el sistema.
- Caso de uso: conjunto de escenarios (secuencias de interacción de los actores y el sistema)

Casos de uso

- Pasos:
  - Identificar los límites (alcance) del sistema.
  - Identificar los actores principales.
  - Para cada uno, identificar sus objetivos.
  - Definir casos de uso que satisfagan sus objetivos.

Casos de Uso: Ejemplo

CASO DE USO 1: Procesar venta

Actor Primario:

- Cajero.

Interesados y objetivos:

- Cajero: Quiere anotaciones precisas y rápidas de precios, sin errores.
- Cliente: Quiere que el pago sea rápido con el mínimo esfuerzo. Quiere una prueba de compra para justificar devoluciones.
- Compañía: Quieren almacenar las transacciones y satisfacer los intereses de los clientes.
- Comercial: Quiere que se le actualicen sus comisiones por venta.
- Agencias de impuestos gubernamentales: Quieren recolectar impuestos de cada venta. Puede que haya varias agencias (nacionales, regionales, etc.)
- Servicios de Autorización de Pagos (por tarjetas de crédito): Quiere recibir peticiones digitales de autorizaciones en el formato y protocolo correcto.

Precondiciones:

- El cajero se ha identificado y autentificado.

Casos de Uso: Ejemplo

Garantía de éxito (Postcondiciones):

Se registra la compra en el sistema. Se calcula el impuesto aplicable. Se actualizan los sistemas de inventario y de contabilidad. Se registran las comisiones. Se genera un recibo. Se registran las aprobaciones de pago por tarjeta.

Escenario principal de Exito:

1. Llega un clienta al TPV con bienes o servicios que comprar.
2. El cajero comienza una nueva compra.
3. El cajero introduce un identificador de producto.
4. El sistema registra el elemento y presenta una descripción del mismo, su precio y total actual. Se calcula el precio de una lista de reglas.

El cajero repite los pasos 3-4 hasta que no hay más elementos.

5. El sistema presenta el total con los impuestos calculados.
6. El cajero le dice el total al cliente, y le pide que pague.
7. El cliente paga y el sistema procesa el pago.
8. El sistema registra la venta completada y manda la información a los sistemas externos de inventario y contabilidad.
9. El sistema genera el recibo.
10. El cliente se va.

Casos de Uso: Ejemplo

Extensiones (Flujos alternativos):
a*. En cualquier momento, el sistema falla.

3a. Identificador inválido.
    1. El sistema señala un error y rechaza la entrada.
7a. Pago en efectivo.
    ...
7b. Pago con tarjeta.
    ...

Requisitos especiales:

- Pantalla táctil en panel grande y plano. El texto debe ser visible desde un metro.
- Respuesta de autorización de crédito en menos de 30 secs, el 90% de las veces.
- Recuperación robusta cuando el acceso a sistemas externos (tales como el inventario, impuestos, etc.) falla.
- Posibilidades de internazionalización de texto.
- Reglas de negocio insertables en los pasos 3 y 7.

...

Casos de Uso: Ejemplo

Lista de variaciones de tecnología y datos:

3a. Se introduce el identificador del elemento mediante escáner de código de barras o mediante el teclado.

3b. Distintos esquemas de identificadores: UPC, EAN, JAN o SKU.

7a. La información sobre el pago con tarjeta se puede introducir mediante el teclado o lector.

7b. Se pide firma en papel. En dos años, creemos que muchos clientes van a querer captura de firma digital.

Frecuencia de ocurrencia:

Puede ser casi continua.

Temas abiertos:

¿Cuáles son las posibles variaciones en las leyes sobre impuestos?

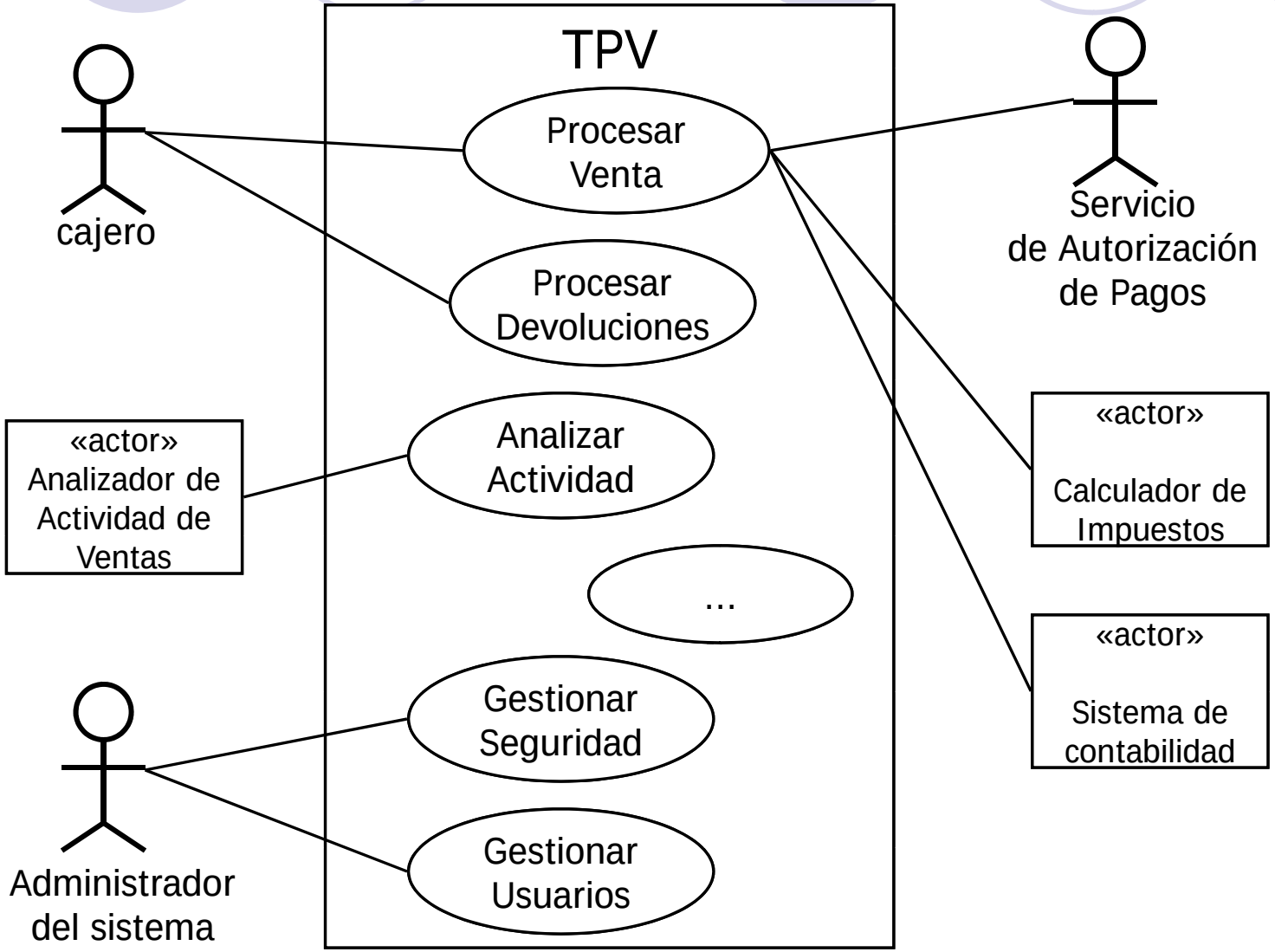
Explorar el tema de recuperación en caso de fallo de sistemas externos.

¿Qué modificaciones se necesitan para negocios distintos?

¿Debe el cajero extraer el cajón con la recaudación al terminar?

¿Puede el cliente usar directamente el lector de tarjetas o es el cajero el que lo hace?

Diagrama de Casos de Uso (UML)

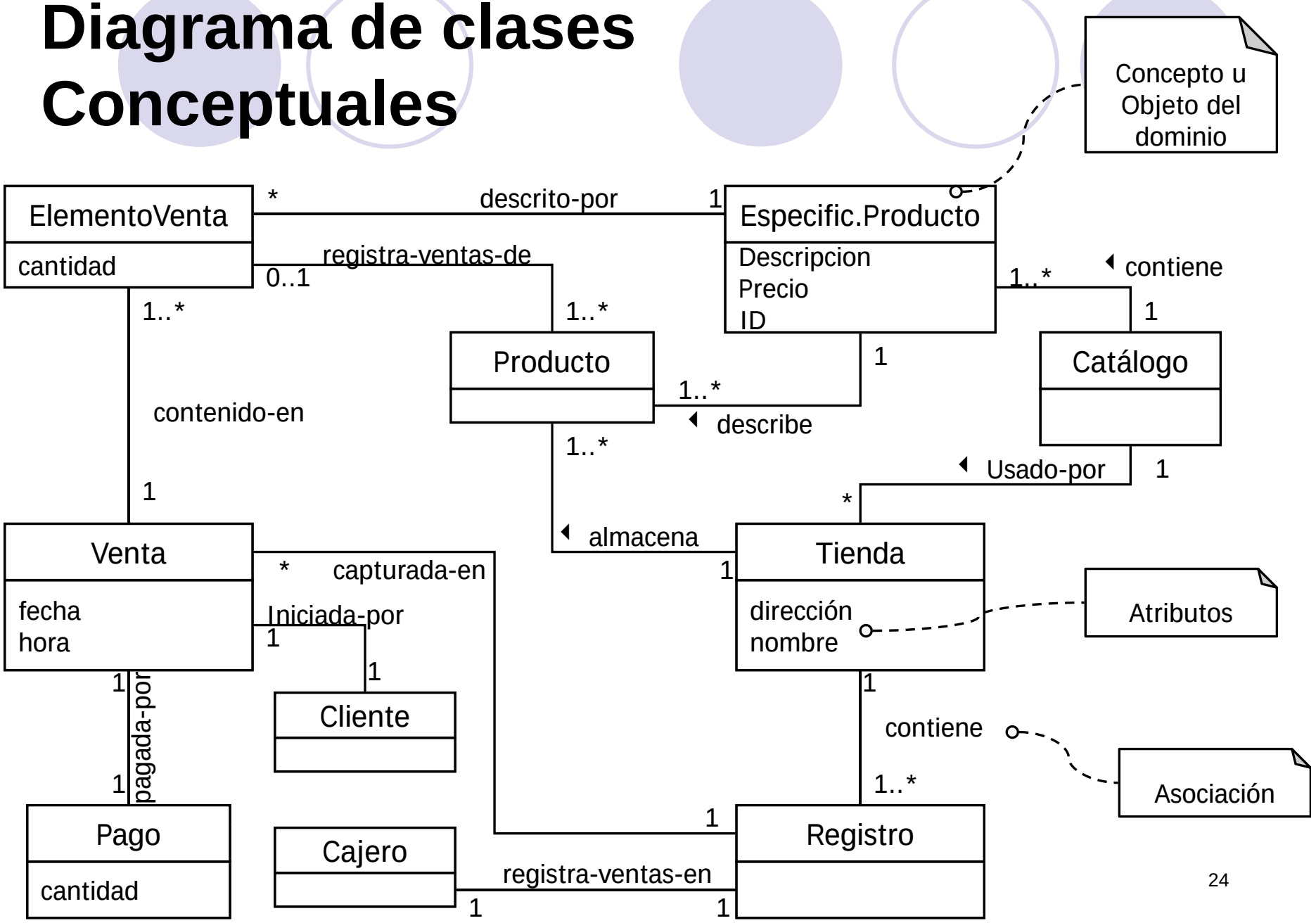


Modelos de Análisis Basados en Clases

Identificar las clases

- Analizar los documentos de análisis, o casos de uso.
- Clases que pertenecen al espacio de la solución vs. espacio del problema.
- Aislar los sustantivos:
  - Entidades externas: producen o consumen información que usa el sistema.
  - Cosas (informes, señales, etc.): información que necesita el sistema.
  - Sucesos o eventos que ocurren durante la operación del sistema.
  - Papeles que desempeñan los usuarios.
  - Unidades organizacionales.
  - Sitios que establecen el contexto y la función global del sistema.
  - Estructuras que definen una clase de objetos o clases relacionadas.

Diagrama de clases Conceptuales

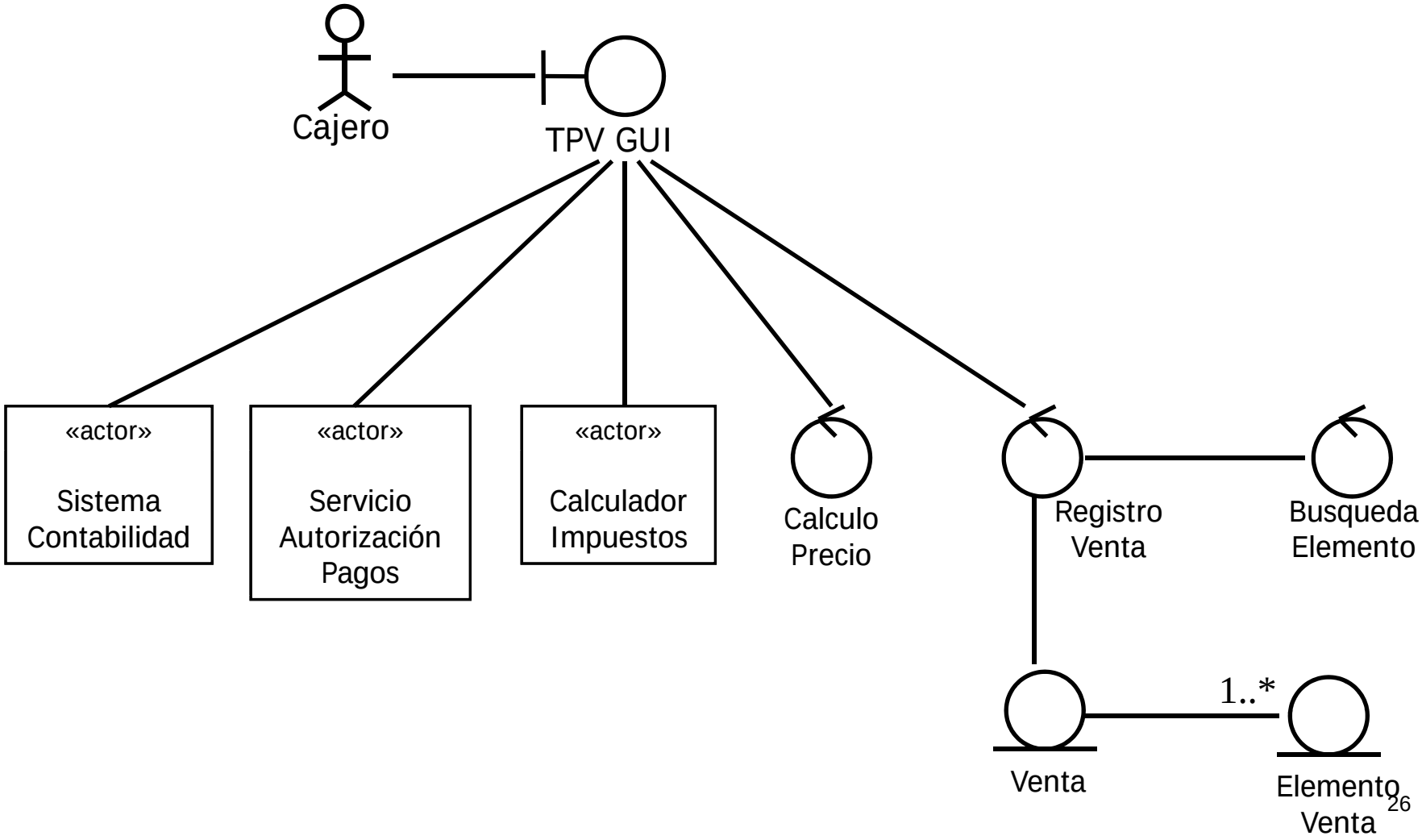


Clasificación de clases

- Tipos de clases:
  - De ***entidad*** (a.k.a. de modelo o de negocio). Son clases que persisten durante la aplicación. Representan información relevante para la aplicación.
  - De ***frontera*** (a.k.a. de contorno). Clases que crean la interfaz que el usuario ve y con la que interacciona.
  - De ***control***. Realizan una "unidad de trabajo": crean o actualizan objetos de entidad, validan datos, etc.

Diagrama de clases de análisis

Caso de uso Procesar Venta



Método de Clase-Responsabilidad-Colaborador (CRC)

- Clases/Responsabilidades/Colaboradores.
- Facilitan la colaboración entre desarrolladores.
- Una ficha por cada clase relevante.
- Se identifican sus responsabilidades.
- División de responsabilidades, relaciones de colaboración.
- Jerarquías de generalización/especialización.

Método de Clase-Responsabilidad-Colaborador (CRC)

Clase: **PlanoDePlanta**	
Descripción:	
Responsabilidad	**Colaborador**
Define el nombre/tipo de plano de planta	
Maneja la posición del plano de planta	
Escala el plano de planta	
Incorpora paredes, puertas y ventanas	Pared
Muestra la posición de las cámaras de vídeo	Camara

Del Análisis al Diseño

- El modelo de análisis describe el sistema desde el punto de vista de los actores.
- **No** contiene información de la estructura interna del sistema, esto es del ***cómo***.
- Diseño del sistema:
  - Objetivos de diseño que se deben optimizar (derivados de los requisitos no funcionales).
  - Una *arquitectura software*: descomposición en subsistemas, dependencias entre ellos, etc.
- Diseño detallado (de objetos).
  - Refinamiento del diseño del sistema.
  - Diseño de las clases de la solución, interfaces.

Diseño Orientado a Objetos

- Conceptos básicos de DOO:
  - Encapsulamiento.
  - Ocultación de información.
  - Herencia.
  - Interfaces.
  - Polimorfismo.

Diseño Orientado a Objetos

Encapsulamiento

- Desarrollador
  - Objetivo: crear clase con interfaz clara y comprensible
  - Manera: ocultar detalles de implementación
  - Beneficios: cambio de estructuras/algoritmos sin afectar
  - Coste: clases reutilizables más caras a corto plazo

Diseño Orientado a Objetos

Encapsulamiento

- Usuario de las clases
  - Objetivo: usar la clase con el mínimo esfuerzo
  - Manera: usar sólo las operaciones provistas
  - Beneficios: interfaz comprensible, bajo coste de programación
  - Coste: pérdida de eficiencia de ejecución

Descomposición Funcional

- Módulos construidos alrededor de las operaciones
- Datos globales o distribuidos entre módulos
- Entrada/Proceso/Salida
- Organigramas de trabajo y flujo de datos

OOD

- Módulos construidos alrededor de las clases
- Clases escasamente acopladas, sin datos globales
- Encapsulamiento y mensajes
- Diagramas jerárquicos de clases

Definición de una clase

- Identificar y nombrar la clase
- Identificar sus componentes
- Identificar sus atributos
- Identificar los errores
- Identificar las conexiones funcionales (qué clases sirve/exige)
- Definir conexiones con superclase y subclases
- Identificar propiedades especiales (persistencia, concurrencia)
- Probar la clase en un prototipo

Identificar atributos

- El conjunto de atributos de una clase debe ser:
  - Completo (contienen toda la información pertinente)
  - General (se aplican a todos los objetos de la clase)
  - Diferenciado (cada atributo representa un aspecto diferente de la clase)

Definir atributos

- Tipos de atributos
  - Atómicos predefinidos (entero, real, carácter, pixel...)
  - Atómico enumerativo (color, día de la semana...)
  - Colección
  - Composición (referencias objetos)
- Valor del atributo
  - Común a muchos objetos (variable de clase)
  - Propio de un objeto (variable de objeto)

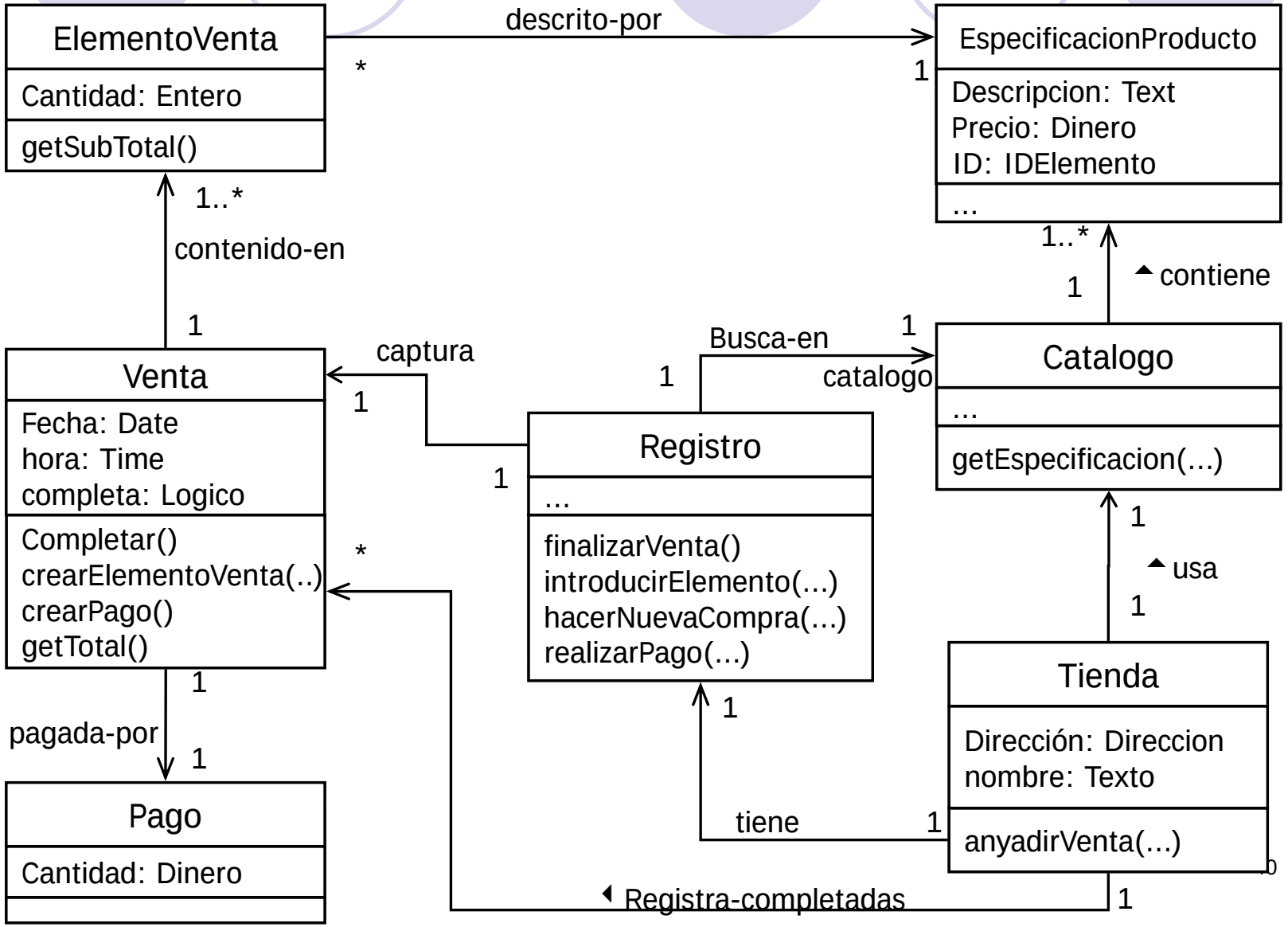
Identificar Métodos

- Método: algoritmo que utiliza y modifica los atributos de una clase
- Un método es desencadenado por un mensaje
- Funcionalidad de la clase: conjunto de sus métodos
- El conjunto de métodos debe ser:
  - Completo (realizan toda la funcionalidad de la clase)
  - General (se aplican a todos los objetos de la clase)
  - Diferenciado (cada método debe ser simple y realizar una sola función)

Definir Métodos

- Tipos de métodos
  - Modificador (asigna valor a un atributo)
  - Selector (devuelve el valor de un atributo)
  - Aplicable a la clase (constructor)
  - Aplicable al objeto
- Parámetros del método
  - ¿Qué información necesita? (argumentos de entrada)
  - ¿Qué debe devolver? (resultado y argumentos de salida)

Ejemplo


ElementoVenta
Cantidad: Entero
getSubTotal()
descrito-por
*
1
EspecificacionProducto
Descripcion: Text
Precio: Dinero
ID: IDElemento
...
1..*
contenido-en
1
1..*
1
contiene
Venta
Fecha: Date
hora: Time
completa: Logico
Completar()
crearElementoVenta(..)
crearPago()
getTotal()
captura
1
1
Registro
...
finalizarVenta()
introducirElemento(...)
hacerNuevaCompra(...)
realizarPago(...)
Busca-en
1
catalogo
1
Catalogo
...
getEspecificacion(...)
1
usa
1
*
1
pagada-por
1
Pago
Cantidad: Dinero
1
tiene
1
Tienda
Dirección: Direccion
nombre: Texto
anyadirVenta(...)
0..
Registra-completadas
1

Identificar Errores

- ¿Qué puede salir mal durante la ejecución de un método?
- ¿Qué comprobaciones debe hacer cada método?
- ¿Cómo interceptar y corregir las condiciones de error?

Patrones de Diseño

- Catálogo de soluciones que han probado ser buenas para ciertos problemas comunes de diseño.
- Evita “reinventar la rueda” continuamente.
- Reutilización de buenas prácticas, común en otras ingenierías.
- Un patrón es una descripción del problema y la esencia de su solución, que se puede reutilizar en casos distintos.
- Los estudiaremos en el Tema 8.

Indice

- Modelos de Ciclo de Vida.
- Análisis.
- Diseño.
- **Notaciones.**
  - **UML**
- Metodologías.

UML

- *Unified Modeling Language.*

- Combinar y estandarizar una notación para la descripción de sistemas orientados a objetos a partir de los lenguajes de modelado más conocidos:
  - Booch - OOD.
  - Rumbaugh - OMT.
  - Jacobson - OOSE y Objectory.

- Combina las mejores propiedades de:
  - Conceptos de modelos de datos (ERD)
  - Modelos de negocios (workflow)
  - Modelos de Objetos
  - Modelos de Componentes

UML

- Es un lenguaje gráfico para visualizar, especificar, construir y documentar las partes de un sistema de software desde distintos puntos de vista.

- Ofrece una forma estándar de modelar sistemas software, pudiendo utilizarse:
  - Con cualquier proceso de desarrollo.
  - A lo largo de todo el ciclo de vida.
  - Con distintas tecnologías de implementación

- Puede usarse también en otras áreas, como la ingeniería de negocio, modelado de procesos, etc.

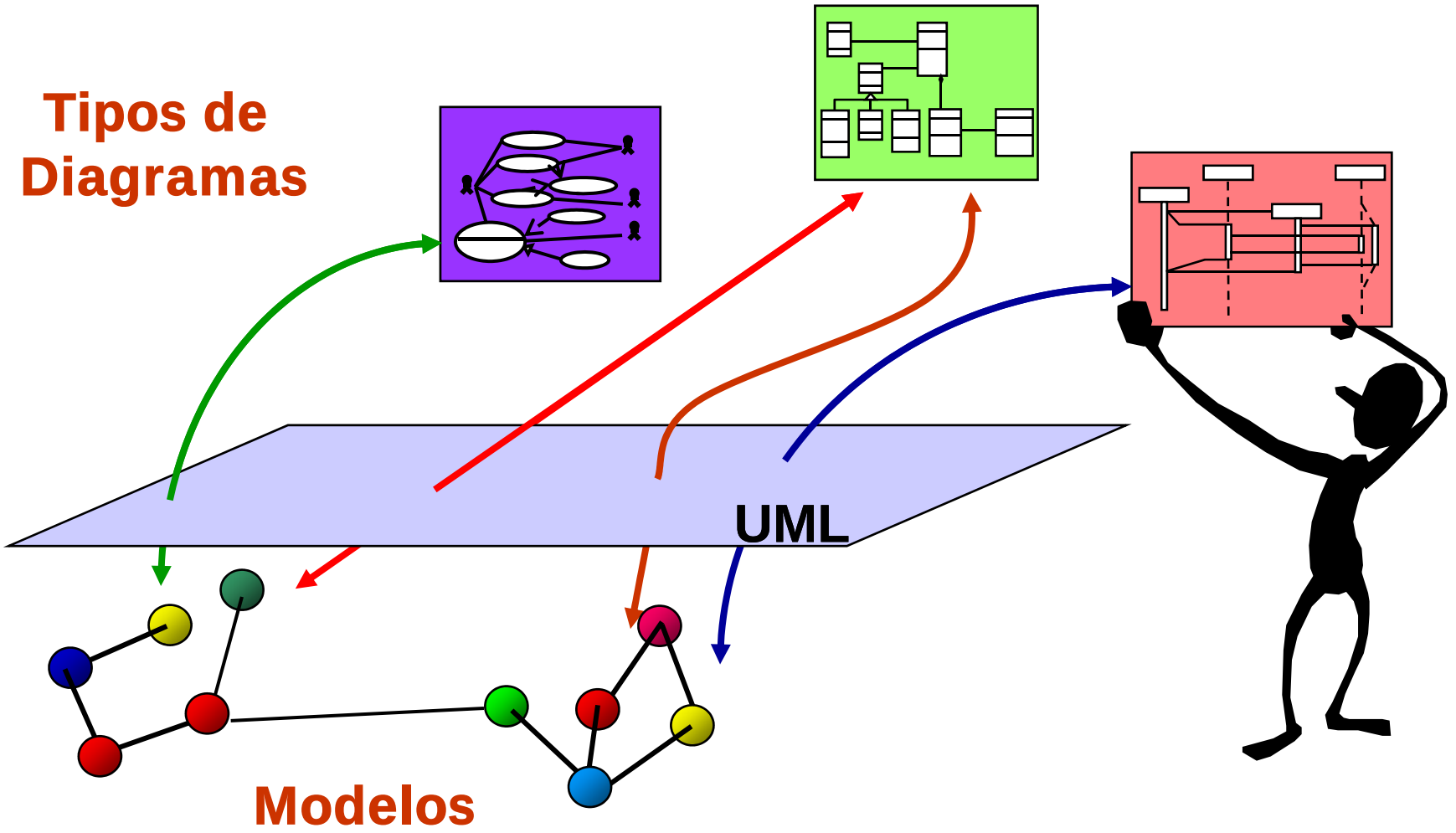
UML

- **No** es un método, ni un proceso ni una metodología.
- **No** establece qué modelos construir.
- Para un óptimo aprovechamiento, debe ser usado en un proceso:
  - guiado por casos de uso,
  - centrado en la arquitectura,
  - iterativo e
  - incremental.

UML

- UML es una familia de notaciones, útiles para describir distintos aspectos de un sistema:
  - ***Estático***. Describe los elementos del sistema y sus relaciones.
  - ***Dinámico***. Describe el comportamiento del sistema a lo largo del tiempo.
    - ***Casos de Uso***. Desde el punto de vista del usuario.

UML



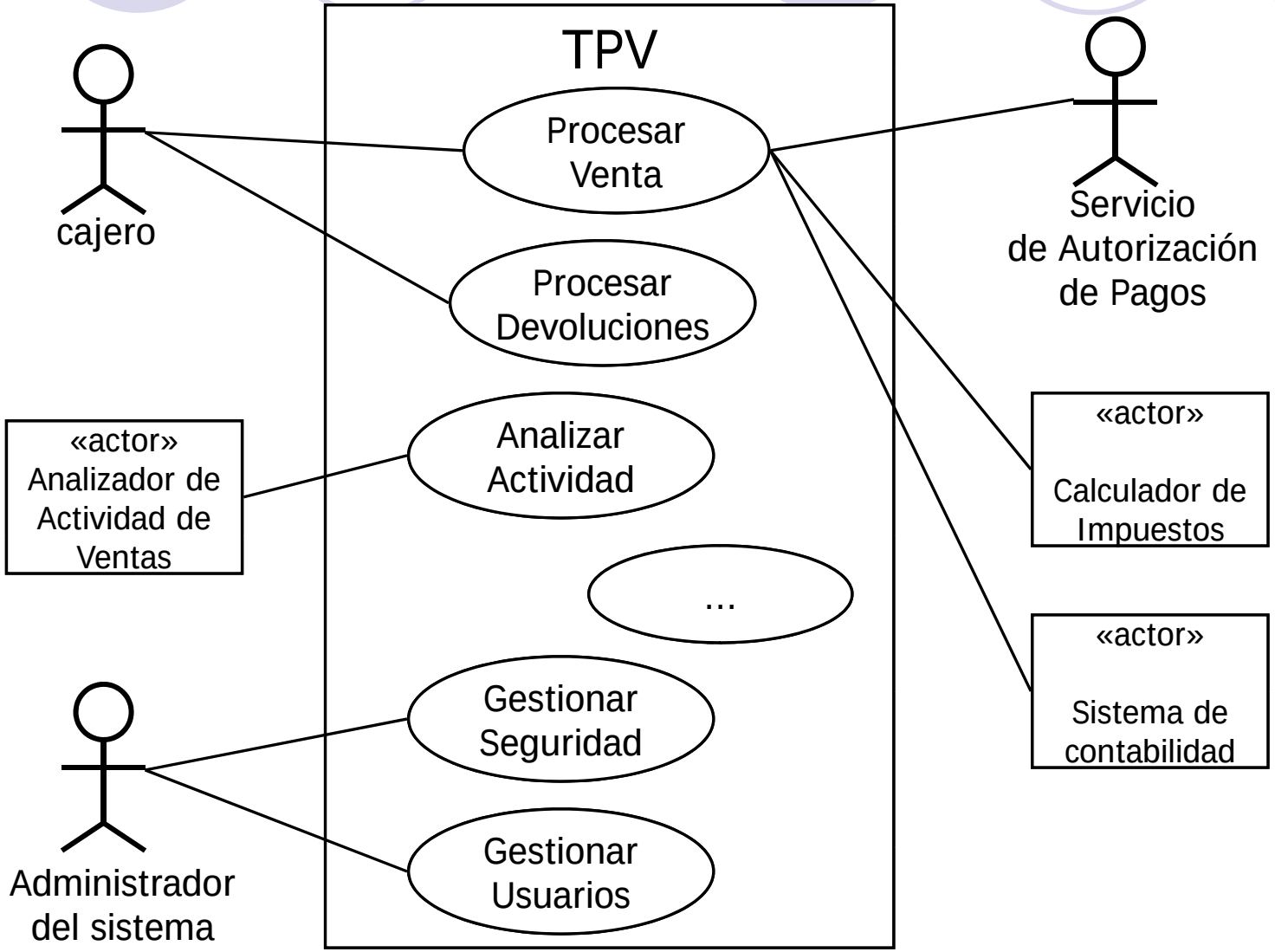
Vistas

- Vista de Casos de Uso
  - Funcionalidad externa del sistema
- Vista Lógica
  - Estructura estática y conducta dinámica del sistema
- Vista de Componentes
  - Organización de las componentes
- Vista de Concurrencia
  - Comunicaciones y sincronización
- Vista de Despliegue
  - Arquitectura física

Vista de Casos de Uso

- Dirigida al Análisis de Requisitos (lo que quiere hacer el usuario)
- Describe la funcionalidad del sistema, como la perciben los actores externos
- Dirige el desarrollo de las otras vistas
- Define los objetivos finales del sistema
- Permite validar el sistema
- Actor externo:
  - Usuario
  - Otro sistema
- Se plasma en diagramas
  - de Casos de Uso
  - de Actividad
  - de Secuencia

Vista de Casos de Uso



Vista Lógica

- Describe la funcionalidad interna
- Dirigida a diseñadores y desarrolladores
- Define la estructura estática
  - Clases, objetos y relaciones
- Define las colaboraciones dinámicas
  - Mensajes y funciones
- Propiedades adicionales
  - Persistencia y concurrencia
  - Interfaces y estructura interna de las clases

Vista Lógica

- Se plasma en diagramas
  - Estáticos
    - de Clases
    - de Objetos
  - Dinámicos
    - de Estado
    - de Secuencia
    - de Colaboración
    - de Actividad

Vista Lógica

Diagramas estáticos

Diagrama de clases

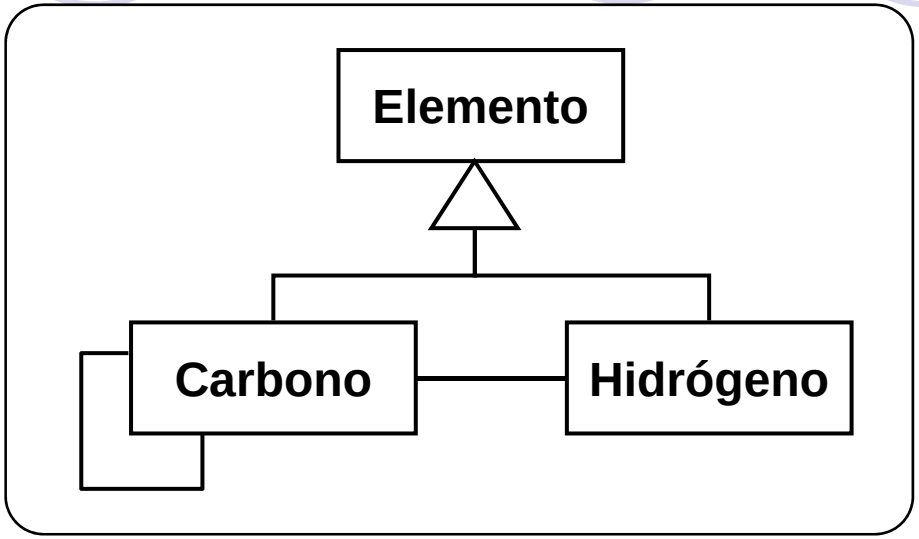
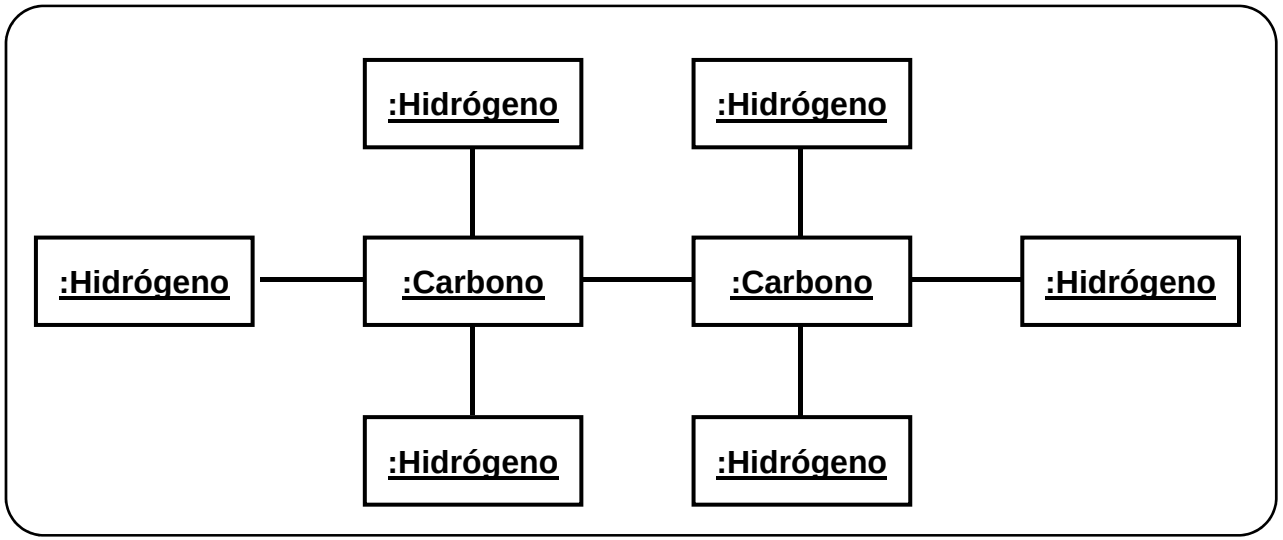
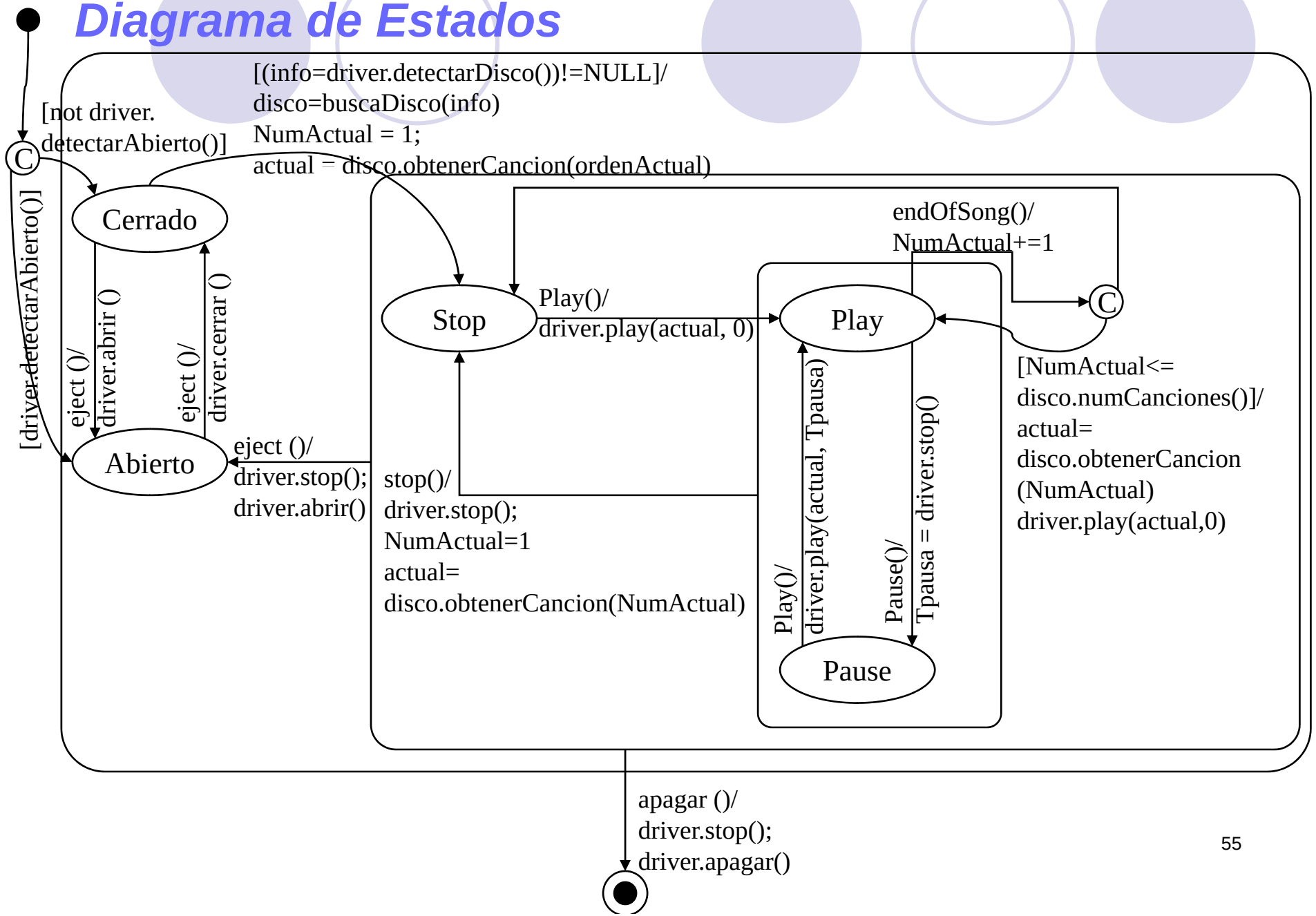


Diagrama de objetos



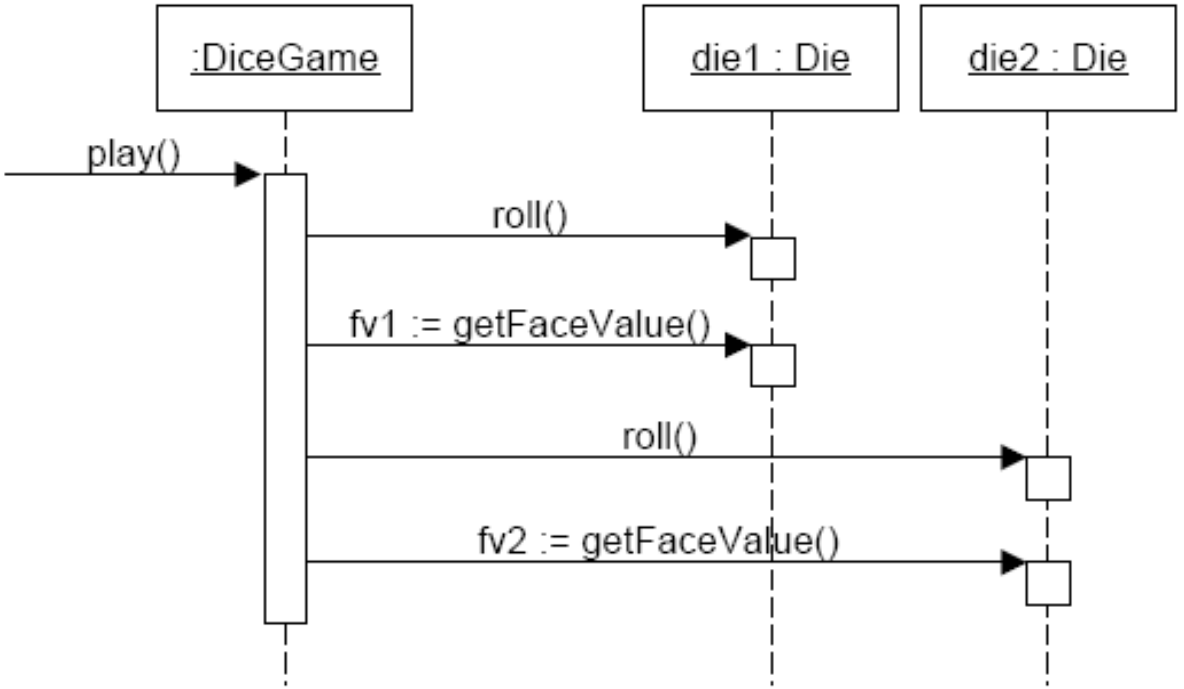
Vista Lógica

Diagrama de Estados



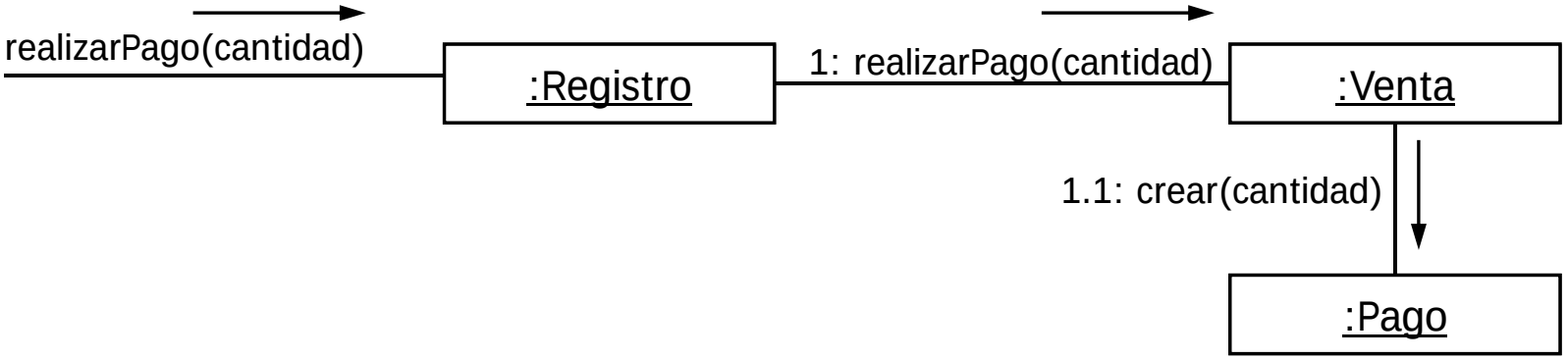
Vista Lógica

Diagrama de Secuencia



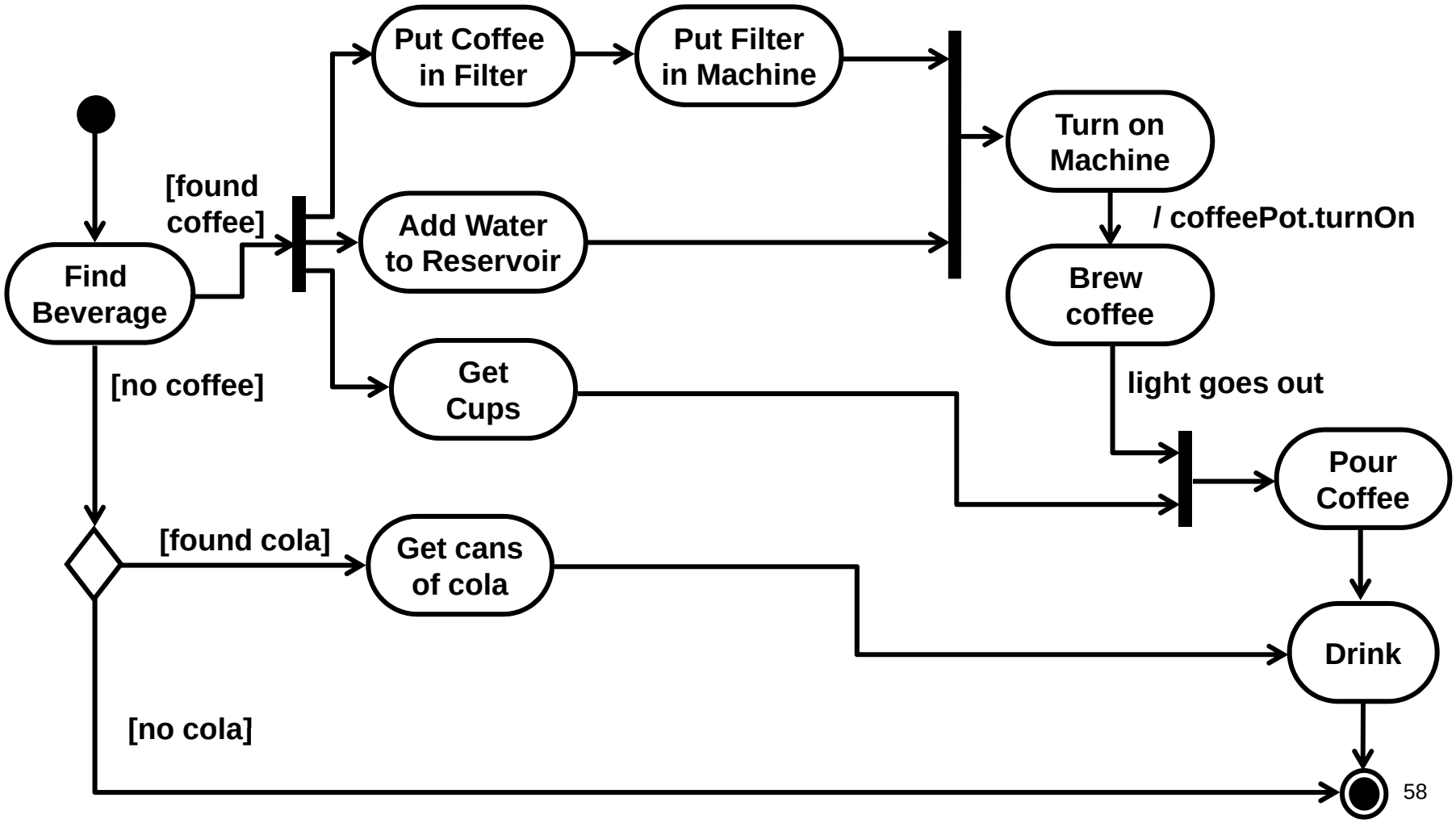
Vista Lógica

Diagrama de Colaboración (comunicación)



Vista Lógica

Diagrama de Actividad

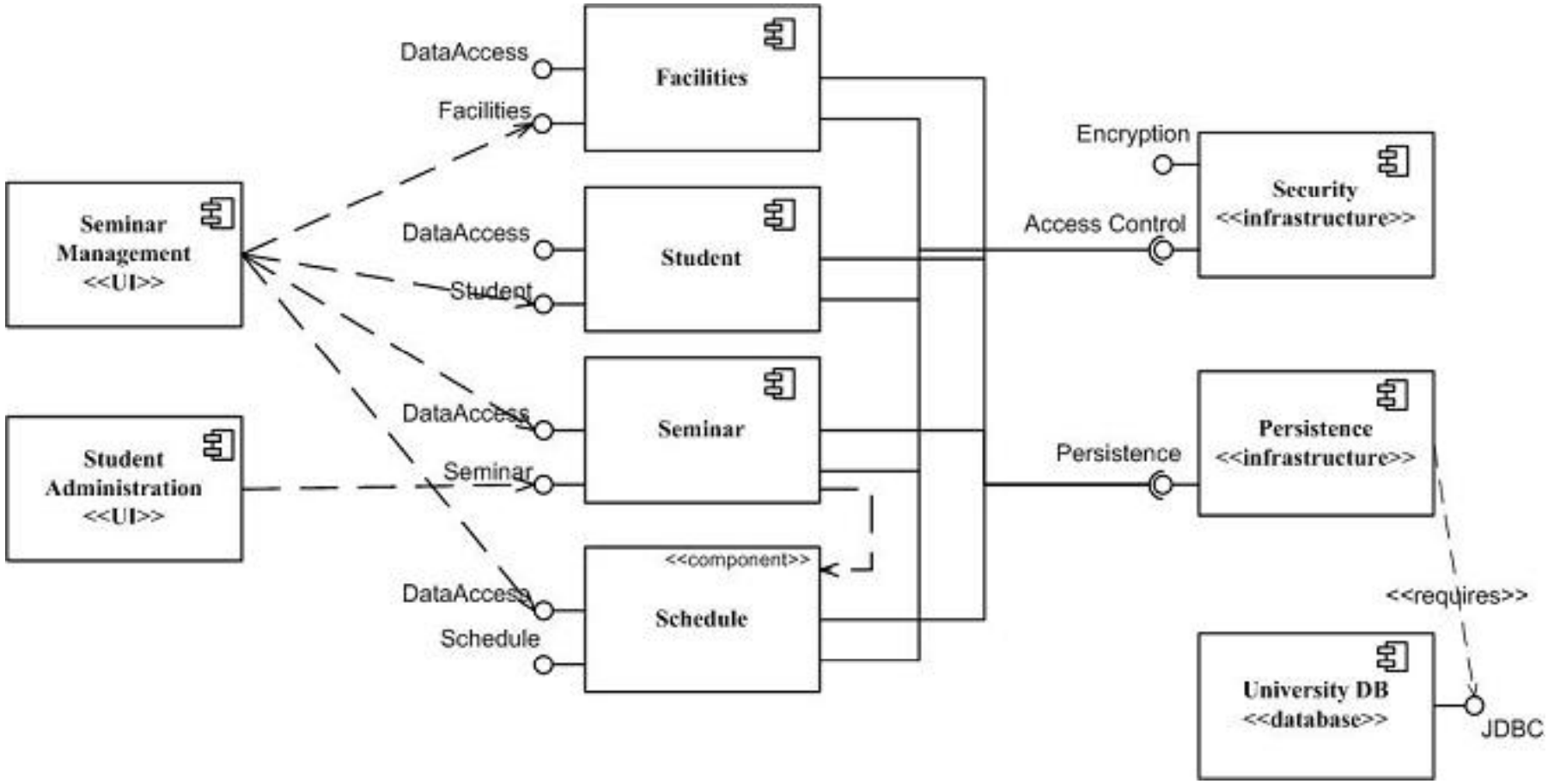


Vista de Componentes

- Describe los módulos del sistema y sus dependencias.
- Modelar la arquitectura software.
- Dirigida a desarrolladores.
- Se plasma en diagramas.
  - de Componentes

Vista de Componentes

Ejemplo



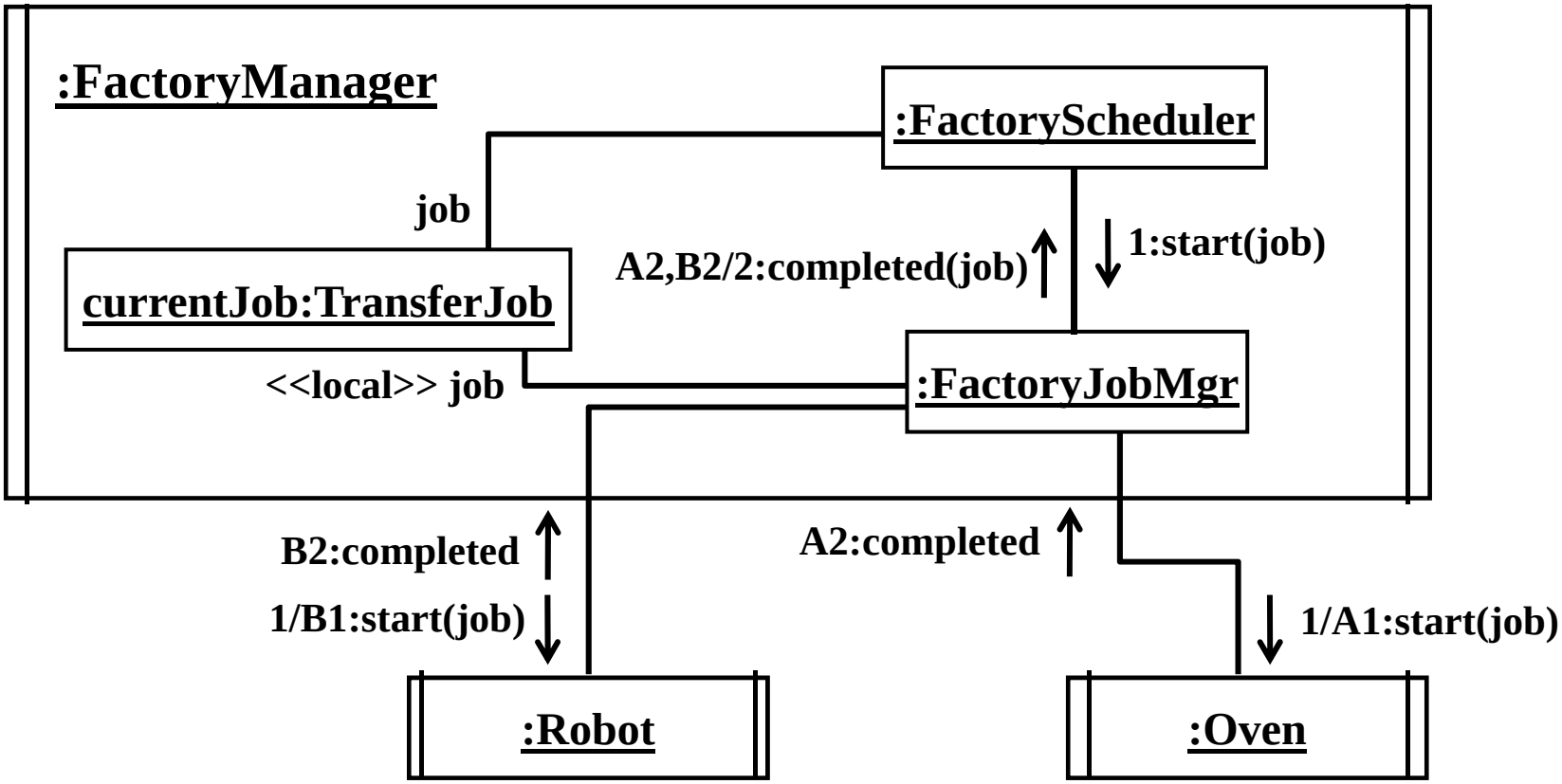
http://www.agilemodeling.com/artifacts/componentDiagram.htm

Vista de Concurrencia

- Describe la división del sistema en procesos y procesadores
- Dirigida a desarrolladores e integradores
- Resuelve problemas de
  - uso eficiente de los recursos
  - ejecución en paralelo (hilos)
  - comunicación y sincronización de hilos
- Se plasma en diagramas
  - dinámicos
  - de Componentes
  - de Despliegue

Vista de Concurrencia

Ejemplo

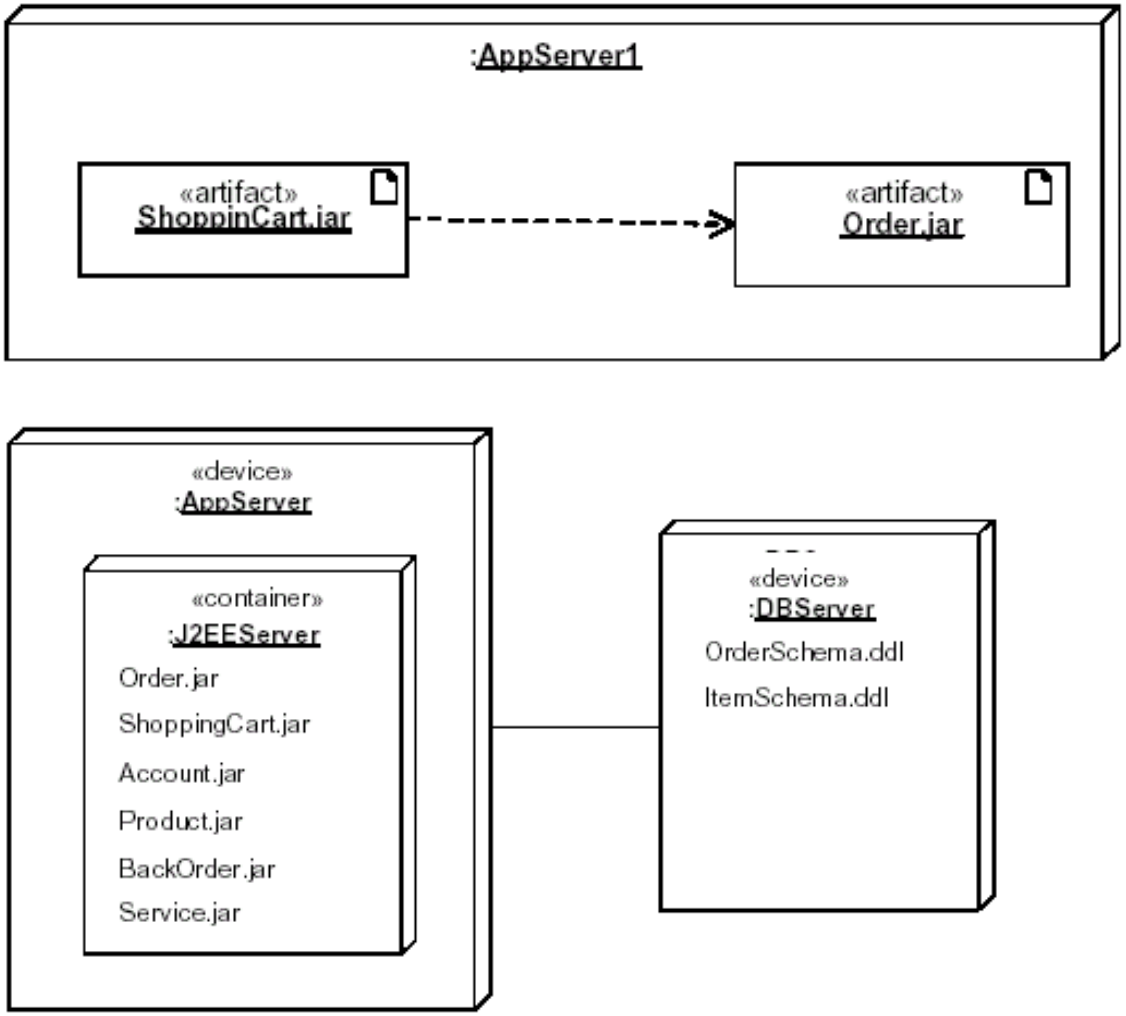


Vista de Despliegue

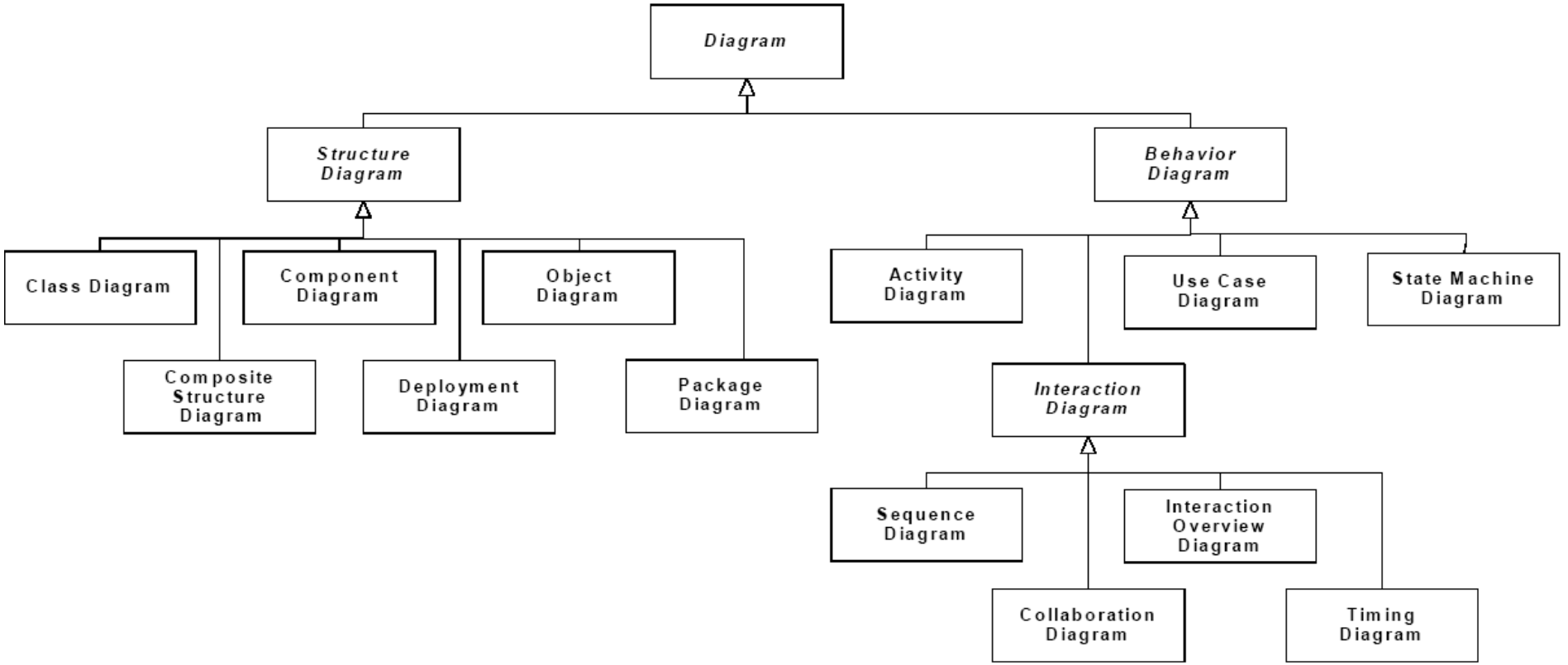
- Muestra la distribución física del sistema (ordenadores, dispositivos) y sus conexiones
- Dirigida a desarrolladores, integradores y probadores
- Se plasma en
  - el diagrama de Despliegue
  - el mapa de asignación de componentes a la arquitectura física

Vista de Despliegue

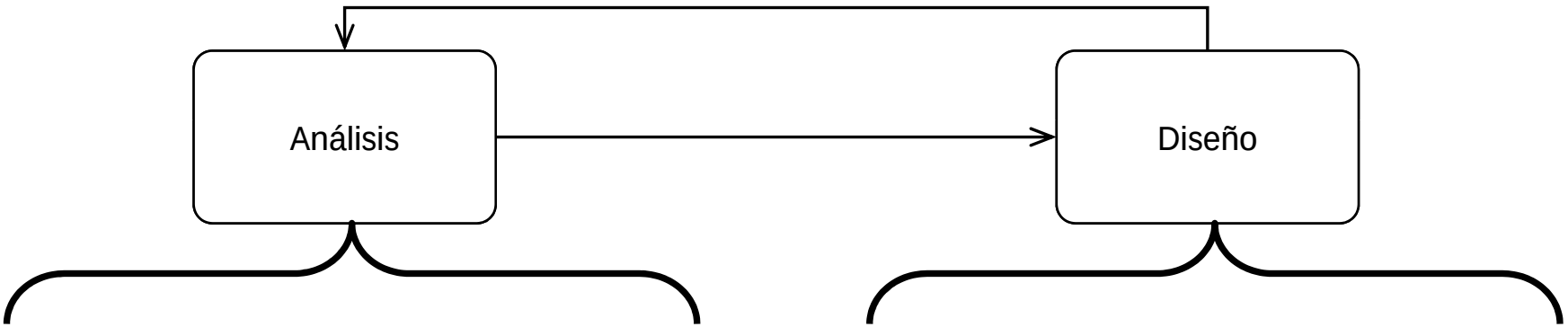
Diagrama de Despliegue



Tipos de Diagramas



Tipos de Diagramas



- D. Casos de Uso.
- D. Secuencia del Sistema.
- D. Clases Conceptuales.
- D. Clases Análisis.

- D. Clases y Objetos.
- D. Colaboración.
- D. Secuencia.
- D. Estados.

Indice

- Modelos de Ciclo de Vida.
- Análisis.
- Diseño.
- Notaciones.
- **Metodologías.**

Metodologías

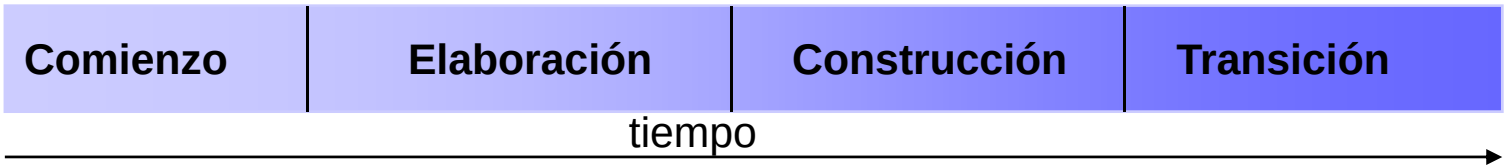
- Una notación no es suficiente.
- ¿Cómo se organiza el proyecto? (MCV)
- ¿Qué documentación se genera?
- ¿Qué técnicas se utilizan en cada fase?
- ¿Quién las realiza?
- ¿Herramientas de soporte?

Metodologías

- Booch (OOAD)
- CASEIode (CCM)
- Coad-Yourdon-Nicola (OOA,OOD)
- NE University (Demeter)
- Object Engin. (Fresco)
- Hewlett-Packard (Fusion)
- Graham (SOMA)
- Texas Instruments (IE\O)
- ICL (MTD)
- ParcPlace (OBA)
- Jacobson (OOSE)
- Olivetti (OGROUP)
- Martin-Odell (OOIE)
- TASKON (OORAM)
- Winter (OSMOSYS)
- Rumbaugh (OMT)
- LBMS (SE/OT)
- Shlaer/Mellor (OOSA)
- CCTA (SSADM)
- Wirfs-Brock (RDD)
- Lloyds Register (Z++)

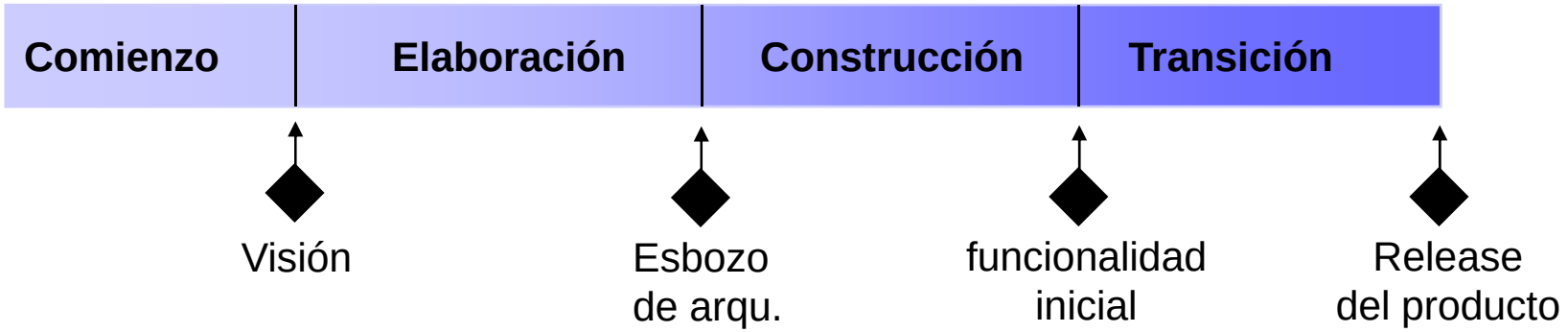
Rational Unified Process (RUP)

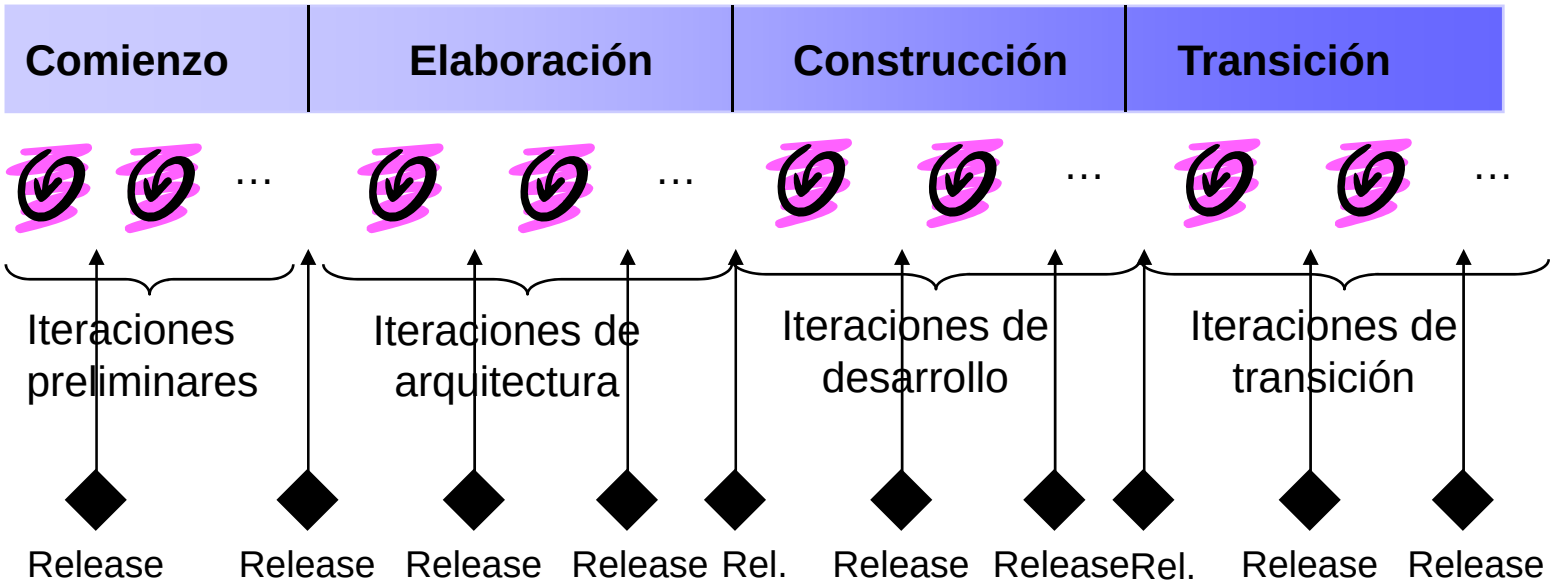
- Es un proceso iterativo e incremental.
- Dirigido por los casos de uso.
- Centrado en la arquitectura.
- Fases:
  - Comienzo. Definir el alcance del proyecto.
  - Elaboración. Plan de proyecto, especificar características, esbozar la arquitectura.
  - Construcción. Construir el producto.
  - Transición. Entrega a los usuarios.



Rational Unified Process (RUP)

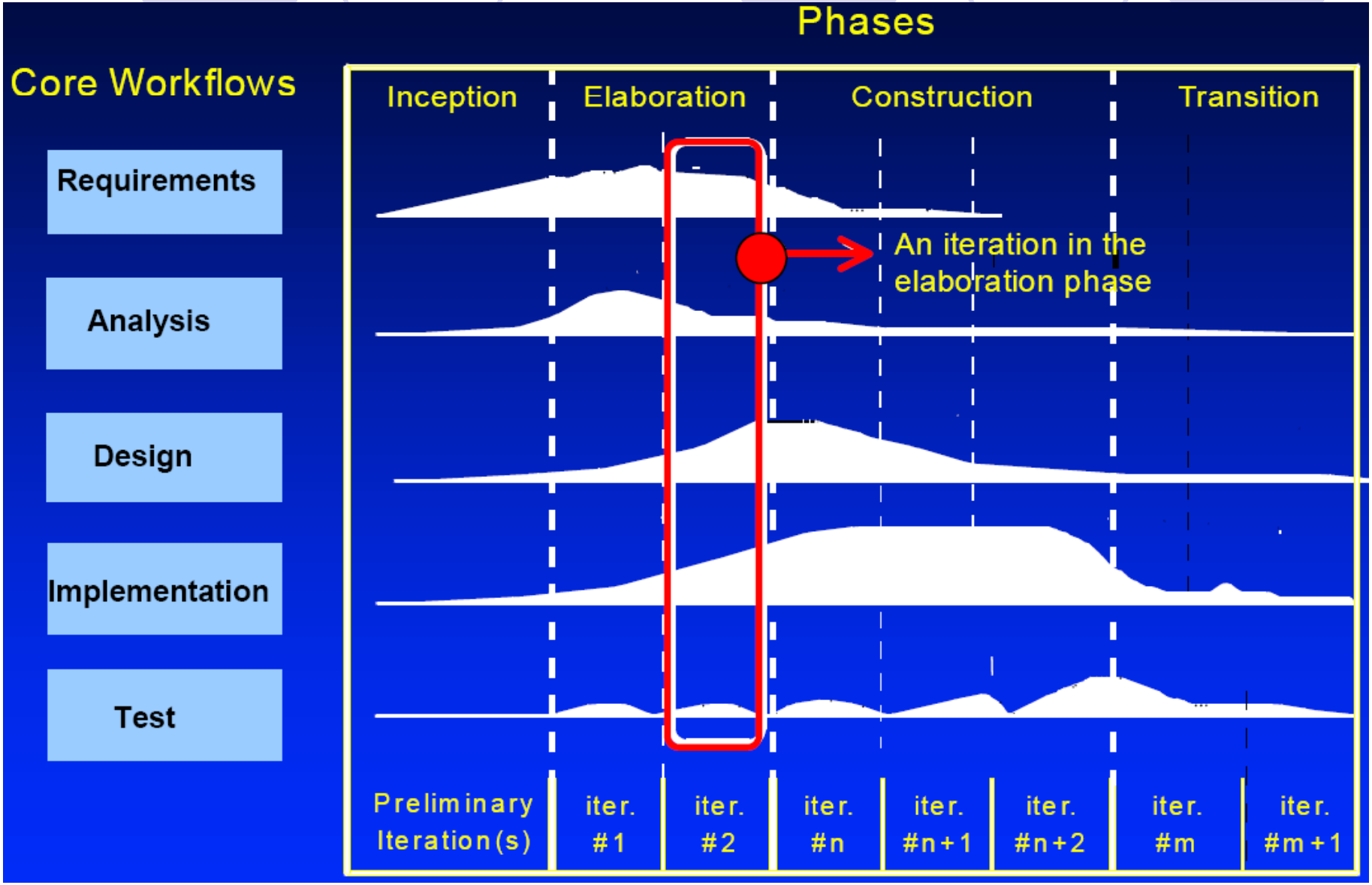
Hitos e Iteraciones





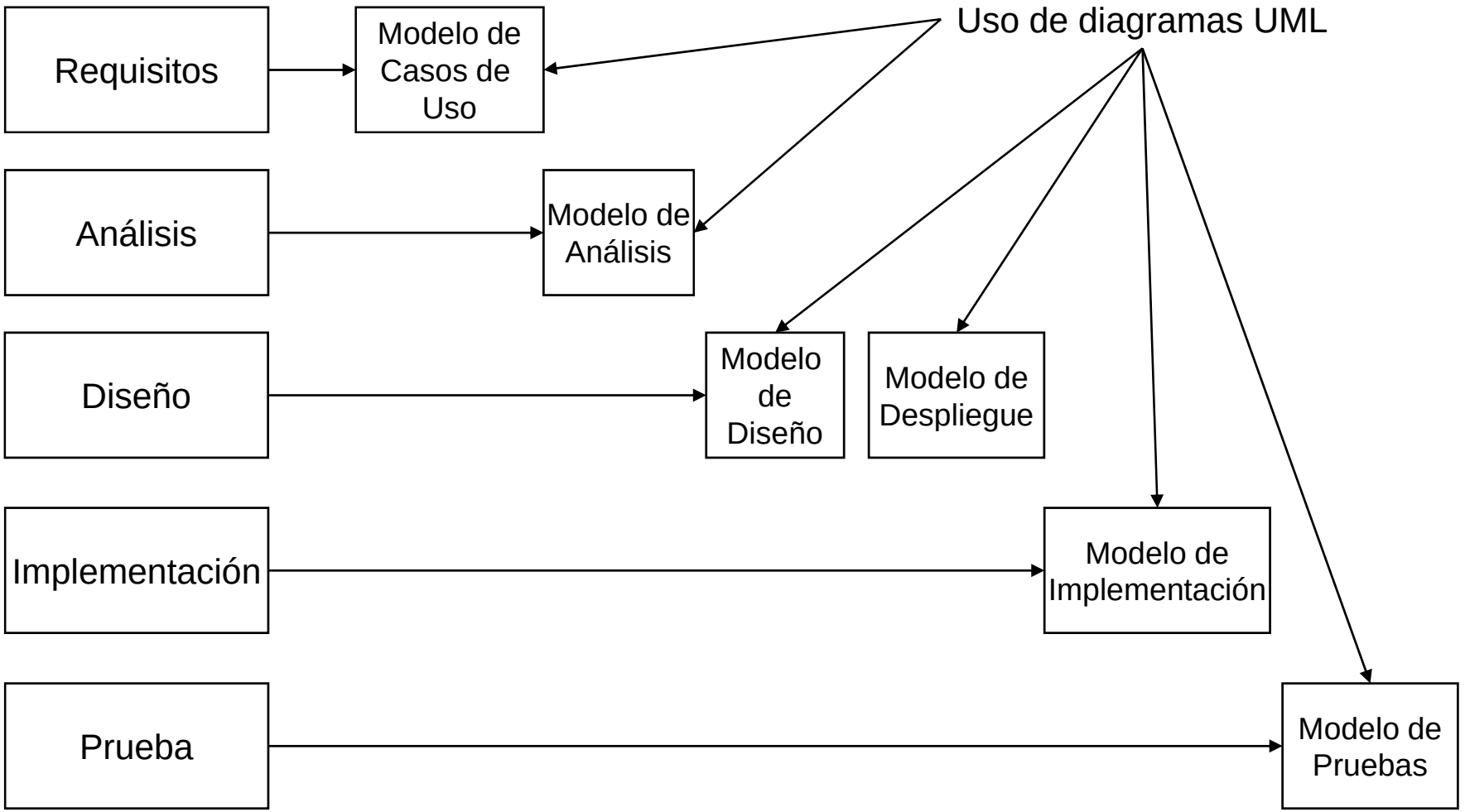
Rational Unified Process (RUP)

Fases, workflows e iteraciones

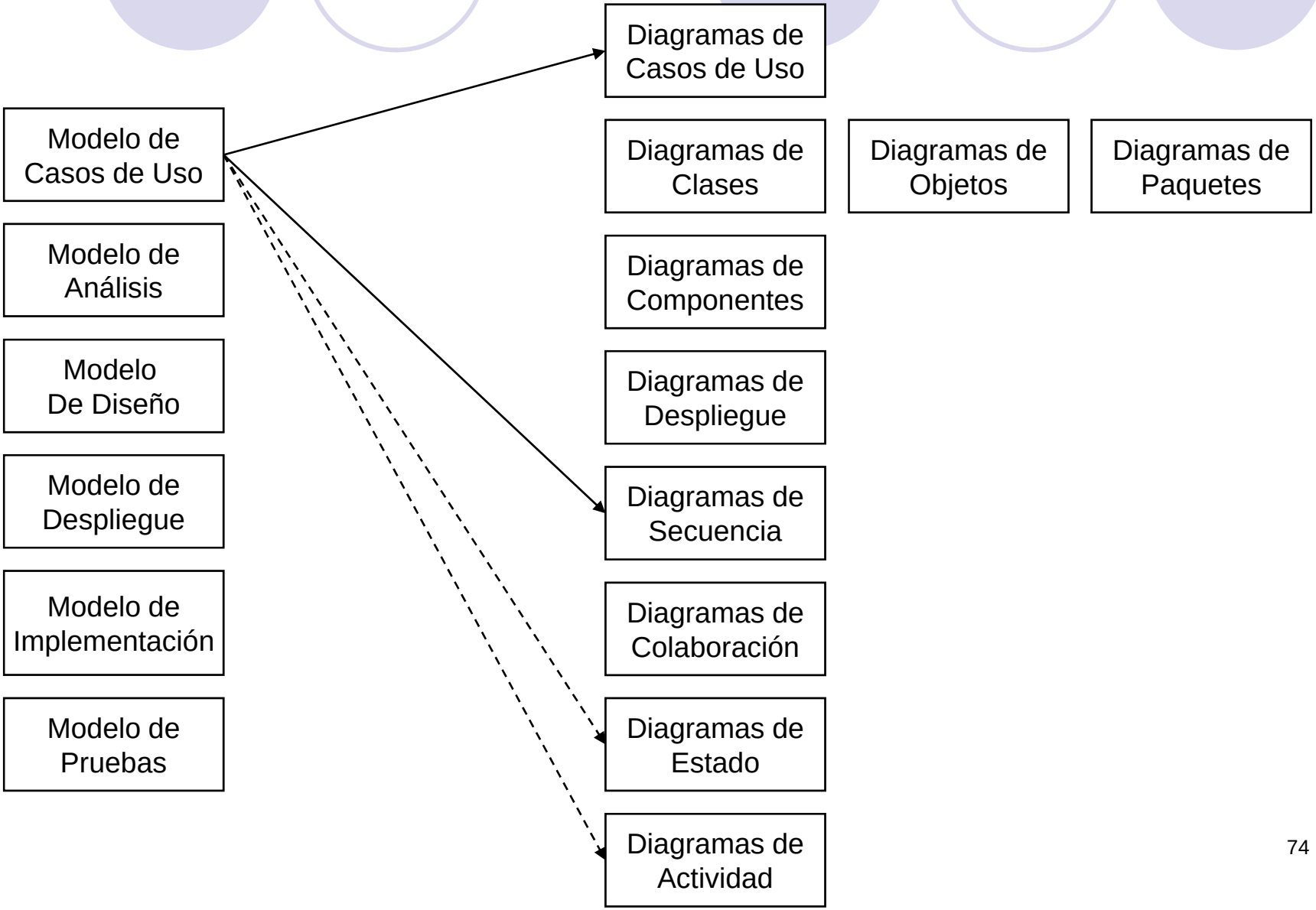


Rational Unified Process (RUP)

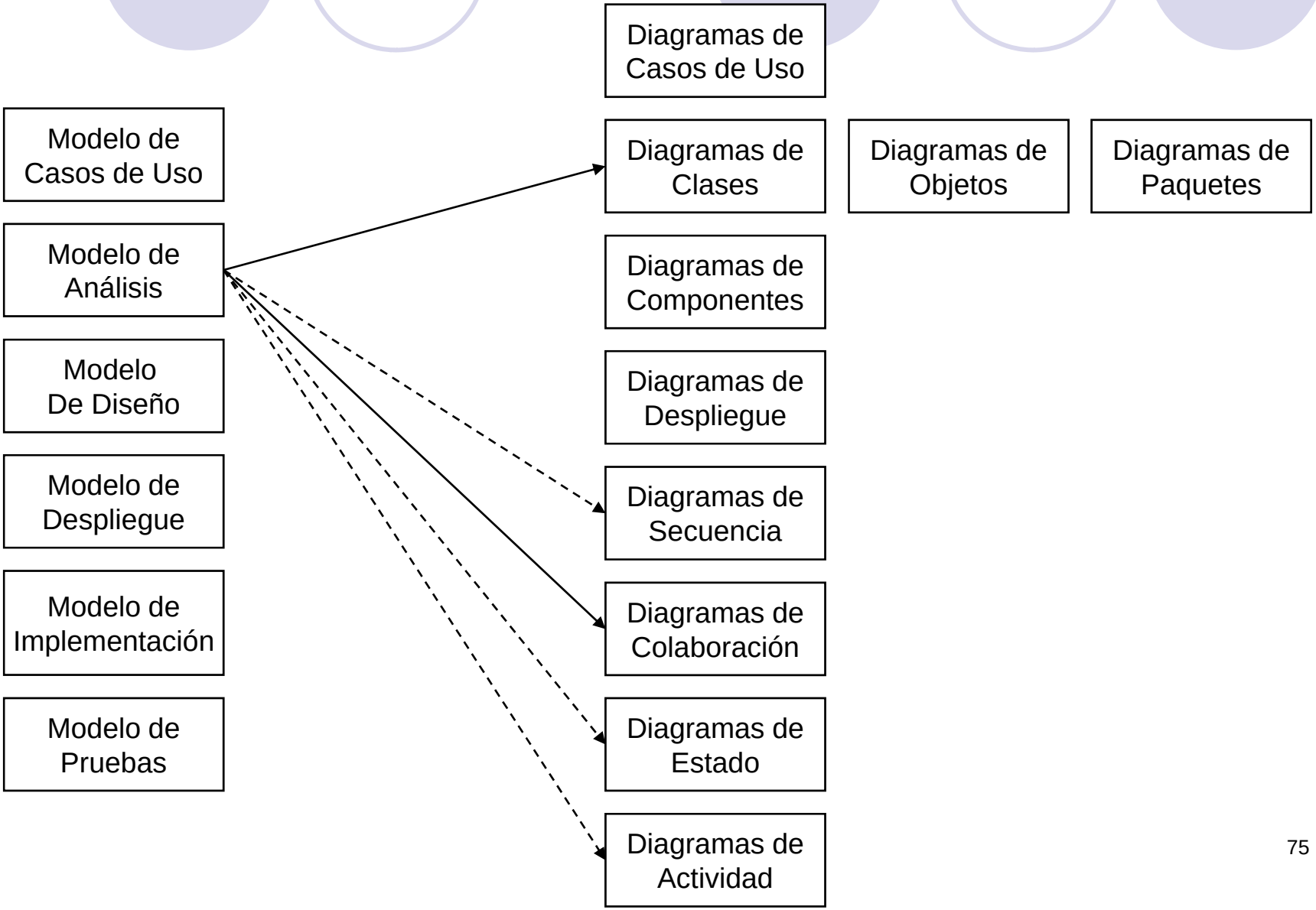
workflows y modelos



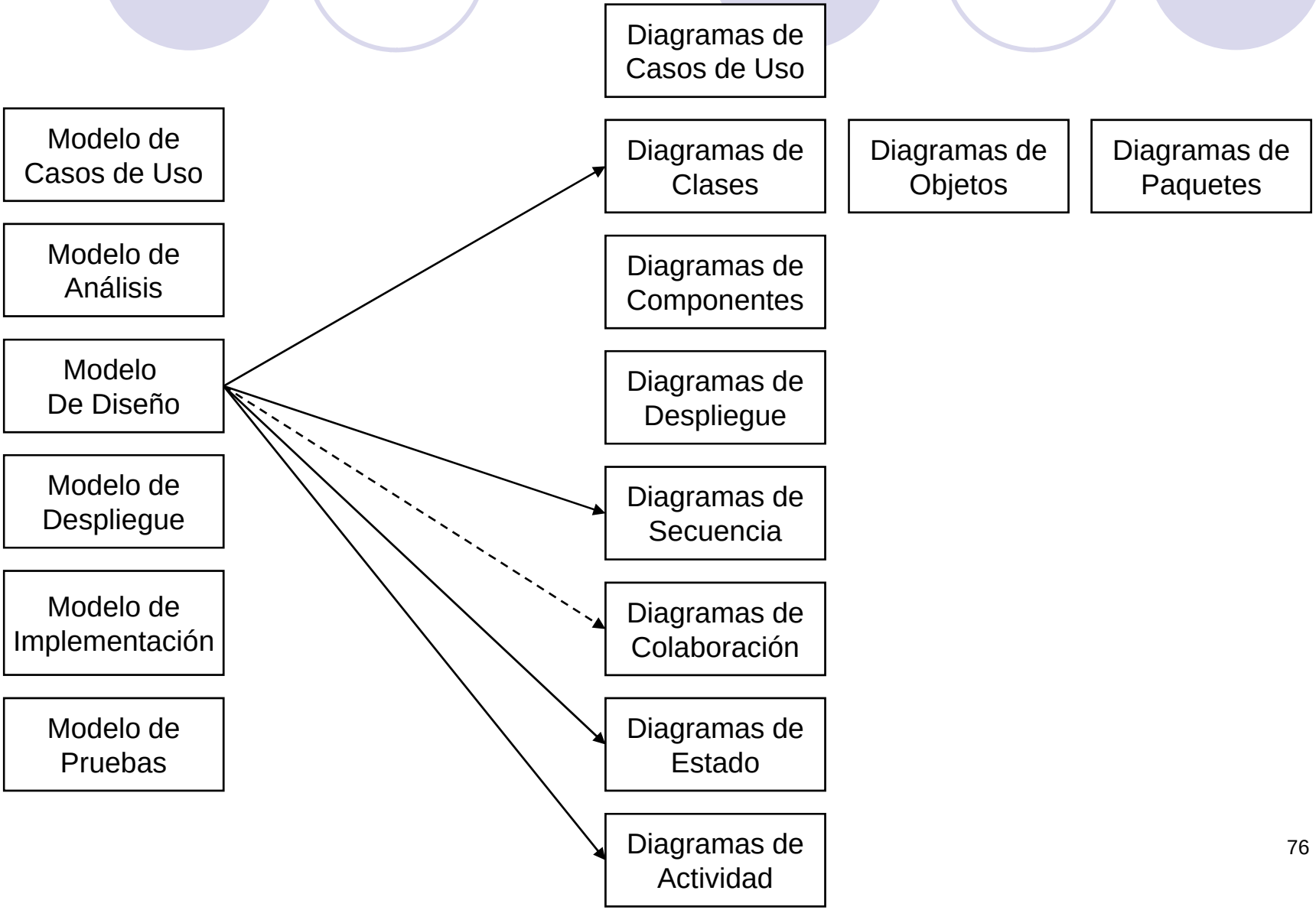
Modelo de Casos de Uso



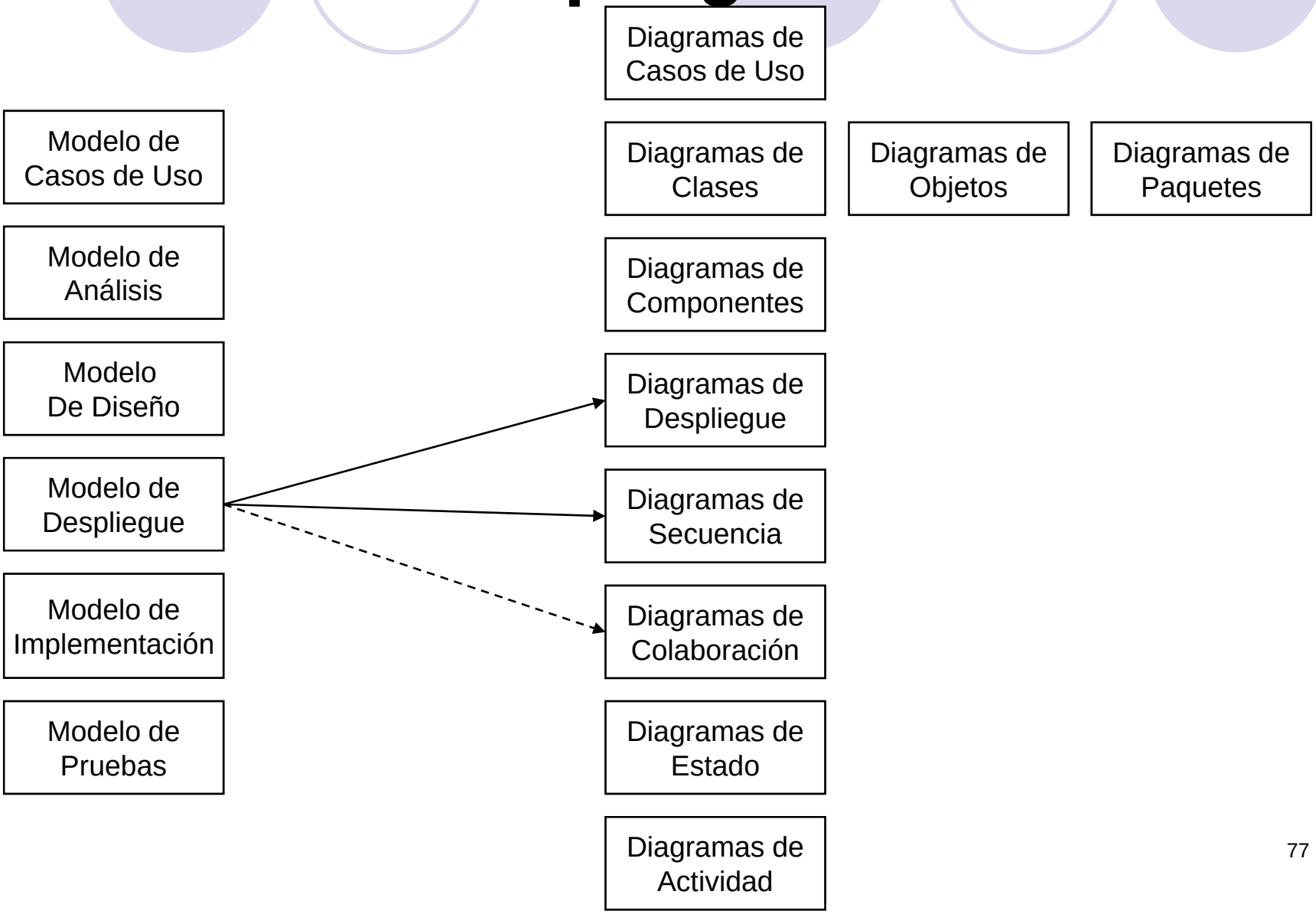
Modelo de Análisis



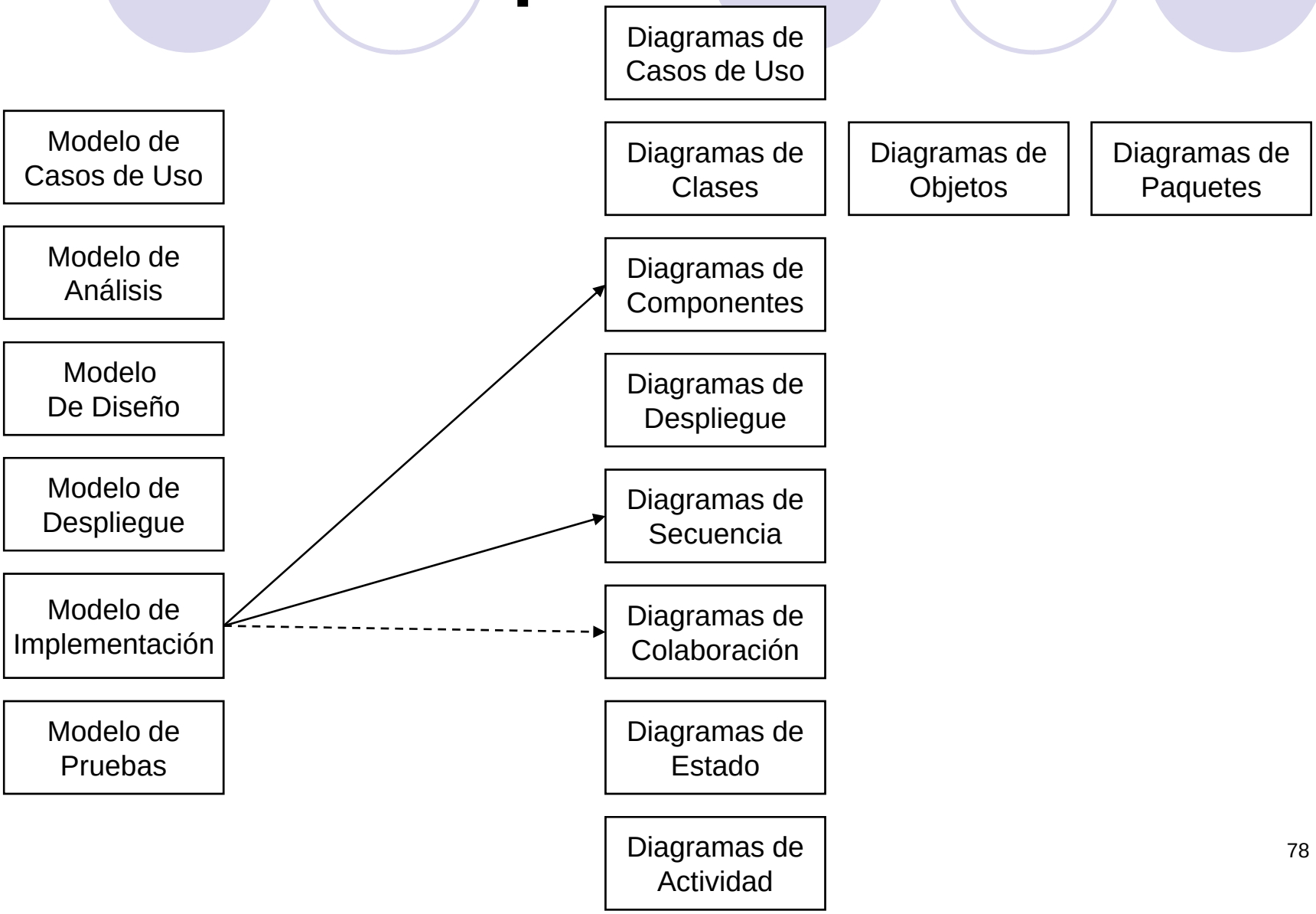
Modelo de Diseño



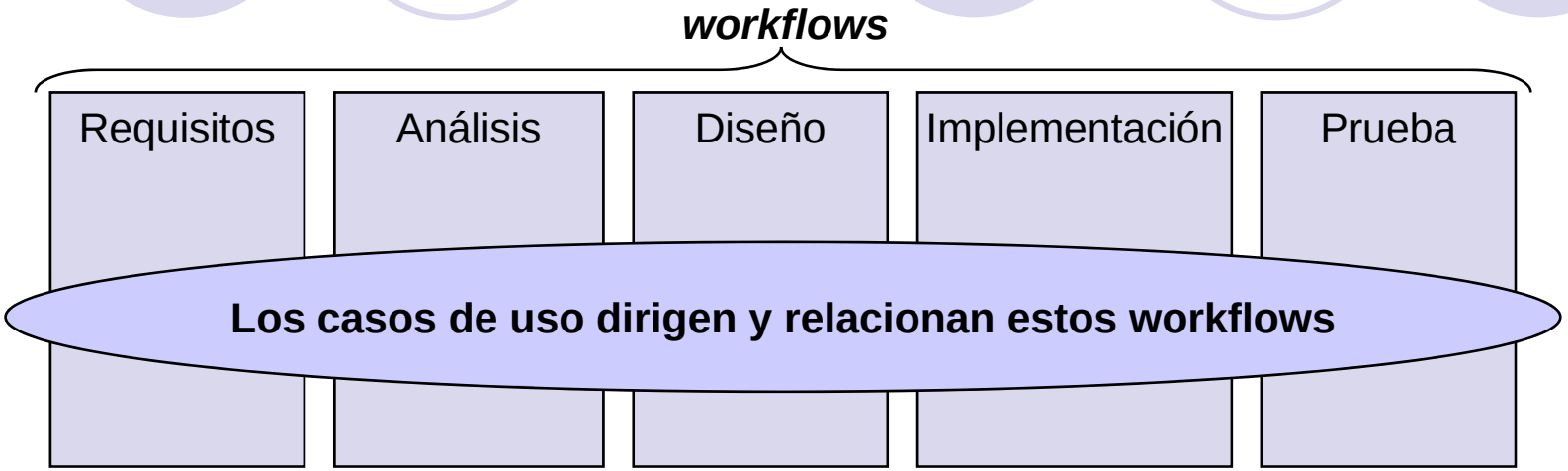
Modelo de Despliegue



Modelo de Implementación



Proceso dirigido por casos de uso

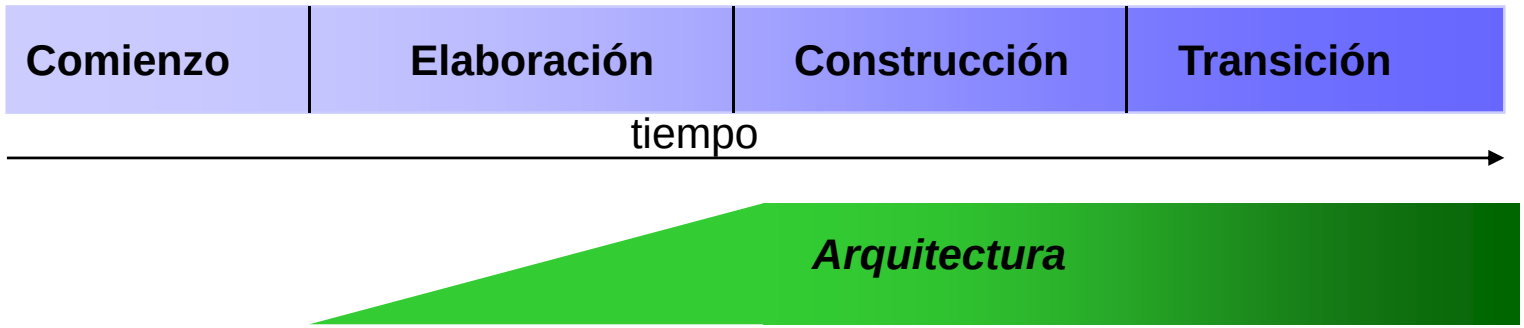


- Los casos de uso dirigen las iteraciones:
  - Creación y validación de la arquitectura.
  - Definición de los casos y procedimientos de prueba.
  - Planificación de las iteraciones.
  - Creación de la documentación de usuario.
  - Entrega del sistema
- Sincronización del contenido de los distintos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

Proceso centrado en la arquitectura

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

- RUP prescribe el refinamiento sucesivo de la arquitectura.



Bibliografía

- “Applying UML and Patterns. 2nd Edition ”. Craig Larman, Prentice Hall, 2002.
- “Applying UML in the Unified Process”. Ivar Jacobson, Rational Software.
- “Ingeniería del Software. Un enfoque práctico 6ª Edición”. R.S. Pressman, McGraw Hill. 2005.
- “Ingeniería del Software Orientado a Objetos”, Bruegge, Dutoit, Prentice Hall. 2002.
- “Análisis y Diseño Orientado a Objetos con UML y el Proceso Unificado”. Schach. McGraw Hill. 2005.