# Tema 12 – Análisis y Diseño

## Ingeniería del Software

Rubén Fuentes Fernández
Dep. Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense Madrid

Trabajando con Antonio Navarro, Juan Pavón y Pablo Gervás



# Contenido

- Requisitos, Análisis, y Diseño
- Artefactos
- Trabajadores
- Análisis
- Diseño

# Requisitos vs. Análisis

- Casos de uso independientes
- En términos del dominio
- Cada caso de uso es una funcionalidad completa

- Identificar clases comunes a varios casos de uso
- En términos de clases, paquetes e interfaces
- Cada clase se ocupa de una parte concreta de alguna funcionalidad



# Cuestiones que surgen

- Interferencias entre casos de uso
- Concurrencia
- Conflictos
- Redundancias
- Recursos compartidos internos



# Objetivos

- Repartir la funcionalidad del sistema en una estructura (clases y paquetes) que facilite:
  - comprensión
  - preparación
  - modificación (flexibilidad ante los cambios)
  - mantenimiento
  - reutilización



# Trazabilidad

- Cada caso de uso se corresponde con una realización de caso de uso
- Esto permite una propagación rápida de cambios en los requisitos a la estructura que se está diseñando



# Análisis vs. Diseño

- Trata sólo requisitos funcionales
- Independiente de lenguaje, plataforma,...
- Número bajo de clases
- Clases provisionales que agrupan funcionalidad amplia

- Añade requisitos no funcionales
- Específico para un lenguaje, una plataforma
- Número elevado de clases
- Clases específicas para tareas concretas (~5 por cada clase de análisis)



# Objetivos del diseño I

- Adquirir comprensión en profundidad de
  - requisitos no funcionales, restricciones relativas a lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia, tecnologías de interfaz de usuario, tecnologías de gestión de transacciones, ...
- Crear entrada apropiada y punto de partida para implementación
  - Subsistemas, interfaces y clases



# Objetivos del diseño II

- Ser capaces de descomponer los trabajos de implementación en partes más manejables
  - Teniendo en cuenta posibles concurrencias
- Capturar interfaces entre subsistemas
  - A estas alturas del ciclo de vida
- Visualizar y reflexionar sobre el diseño
- Crear abstracción “sin costuras” de la implementación



# Contenido

- Requisitos, Análisis, y Diseño
- Artefactos
- Trabajadores
- Análisis
- Diseño



# Artefactos generados

- M1: modelo UML
- M2: clases
- M3: realización de caso de uso
  - Flujo de sucesos
  - Diagramas de clases
  - Diagramas de interacción
  - Requisitos
- M4: paquete
- M5: descripción de la arquitectura - subsistema
- M6: interfaz
- M7: modelo de despliegue



# M1: Modelo UML

- Modelo en UML que contiene el análisis/diseño de la aplicación
- Describe la realización física de los casos de uso
  - cómo afectan requisitos (funcionales y no funcionales) y otras restricciones
- Contiene todos los diagramas necesarios para representar la información obtenida sobre la aplicación (al nivel de detalle pertinente)
  - subsistemas, clases, realizaciones, interfaces



# M2: Clase de análisis

- Es una abstracción de una o varias clases de diseño
  - Trata sólo los requisitos funcionales (los no funcionales se anotan como “requisitos especiales”)
  - Más conceptual que las de diseño, y con más granularidad
  - No tienen interfaz en términos de operaciones y sus signaturas, sino responsabilidades
  - Tiene atributos, pero de alto nivel
  - Tiene relaciones, pero más conceptuales (ni la navegabilidad ni las generalizaciones son críticas)



# M2: Clase de diseño I

- Es una abstracción de una clase o construcción similar en la implementación del sistema
  - especificadas en el mismo lenguaje en que se van a implementar (misma sintaxis)
  - se especifica la visibilidad
    - public, protected, private
  - las relaciones entre clases tienen significados directos en el lenguaje de programación que se usa
    - herencia, variables de otras clases…



# M2: Clase de diseño II

- los métodos (o realizaciones de operaciones) tienen correspondencia directa con métodos en la implementación, pero se especifican en lenguaje natural o pseudocódigo (comentarios al código de la implementación)
- se puede todavía postponer tratamiento de algunos requisitos a la implementación (requisitos de implementación de clase)



# M3: Realización de casos de uso

- Traza directa de un caso de uso del modelo de casos de uso
- Engloba:
  - (R1) una descripción textual del flujo de sucesos
  - (R2) diagramas de clases
    - Muestran las clases participantes
  - (R3) diagramas de colaboración / interacción
    - Muestran la realización de un flujo de sucesos o escenario particular del caso de uso
  - (R4) lista de requisitos especiales
    - No funcionales



# R1: Flujo de sucesos

- Texto adicional que explica los diagramas
- Escrito en términos de objetos o subsistemas
- No debe mencionar atributos concretos, responsabilidades ni asociaciones (porque cambiarán, y así no hay que retocarlos)
- Especialmente útil cuando hay realizaciones descritas por múltiples diagramas o diagramas que representan flujos complejos



# R2: Diagramas de clases

- Una clase puede participar en varias realizaciones
- Algunos atributos, operaciones, relaciones son relevantes sólo para una realización
- Importa para coordinar todos los requisitos de las diferentes realizaciones
- Para manejarlo, se crean diagramas de clases conectados a realizaciones de casos de uso concretas, mostrando solamente las clases participantes, subsistemas y sus relaciones



# R3: Diagramas de interacción

- Preferible representar con diagramas de secuencia (encontrar secuencias concretas, ordenadas en el tiempo)
- Se pueden incluir subsistemas para describir cuáles participan y qué interfaces intervienen
  - Sirve para diseñar los casos de uso a un nivel alto antes de entrar en detalles
  - Cuando decimos en estos diagramas que un subsistema envía o recibe un mensaje es un objeto de una clase del subsistema la que lo hace



# R4: Lista de requisitos

- Descripciones textuales que recogen los requisitos no funcionales sobre una realización de caso de uso
- Serán los mismos del caso de uso y a lo mejor alguno más que haya aparecido durante el análisis/diseño
- Puede haber alguno que surja durante el análisis/diseño pero que no se deba tratar hasta la implementación



# M4: Paquete

- Proporcionan un medio de organizar los artefactos del modelo de análisis/diseño en piezas manejables.
- Puede constar de:
  - Clases
  - Realizaciones de casos de uso
  - Otros paquetes (recursivamente)
- Deben ser cohesivos (contenidos relacionados) y débilmente acoplados (dependencias mínimas entre paquetes)



# M5 Descripción de la arquitectura - Subsistema

- Son una forma de organizar los artefactos del modelo en pieza más manejables
- Tiene:
  - clases
  - realizaciones de casos de uso
  - interfaces
  - otros subsistemas (recursivamente)
- Debe ser cohesivo (contenidos fuertemente asociados) y débilmente acoplado (pocas dependencias con otros)



# M6 Interfaz

- Se utilizan para especificar las operaciones que proporcionan las clases y subsistemas del diseño
- Sirven para separar la especificación de la funcionalidad en términos de operaciones de sus implementaciones en términos de métodos



# Programar para interfaces

- Una clase que implementa un interfaz debe proporcionar métodos que realicen las operaciones del interfaz
- Un subsistema que proporcione un interfaz debe contener clases u otros subsistemas que proporcionen las operaciones del interfaz
- La mayoría de las interfaces entre subsistemas se consideran relevantes para la arquitectura (definen las interacciones permitidas)



# Ventajas de usar interfaces

- Interesa diseñar interfaces estables antes de implementar los subsistemas correspondientes
- Al diseñar los subsistemas se toman esos interfaces:
  - como requisitos y
  - como mecanismos de sincronización entre equipos que realizan subsistemas distintos



# M7: Modelo de despliegue

- Representa una correspondencia entre la arquitectura hardware y la arquitectura software
- 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
- La distribución del sistema tiene una influencia principal en su diseño



# Diagrama de despliegue

- Cada nodo representa un recurso de cómputo
  - Procesador o dispositivo hardware similar
- Las relaciones entre nodos representan medios de comunicación entre ellos
  - Internet, intranet, bus...
- El modelo de despliegue puede describir diferentes configuraciones de red
  - Para pruebas, para simulación...
- La funcionalidad de un nodo viene dada por los componentes que se le asignan



# Contenido

- Requisitos, Análisis, y Diseño
- Artefactos
- Trabajadores
- Análisis
- Diseño



# Trabajadores

- Arquitecto
- Ingeniero de casos de uso
- Ingeniero de componentes



# Arquitecto

- Responsable de:
  - Modelo de análisis
  - Modelo de diseño
  - Modelo de despliegue
- Debe:
  - Diseñar la arquitectura
  - Supervisar el reparto de tareas
  - Vigilar el tratamiento correcto de las clases comunes a varios casos de uso
  - Garantizar la integridad de los modelos
- Los modelos deben ser correctos, consistentes y legibles como un todo



# Integridad de un artefacto

- Contenido y descripción correctos
- Dependencias de otros subsistemas o interfaces son mínimas
- Realizan correctamente las interfaces que ofrecen



# Ingeniero de casos de uso

- Responsable de:
  - La integridad de una o más realizaciones de casos de uso, garantizando que cumplen los requisitos que recaen sobre ellas
  - Hacer legibles y adecuados los diagramas y descripciones textuales
  - No es responsable de las clases, subsistemas e interfaces
- Debe :
  - Realizar los casos de uso en términos de clases y/o subsistemas participantes y sus interfaces
  - Generar descripciones textuales
  - Generar diagramas de interacción



# Ingeniero de componentes

- Define y mantiene:
  - Las responsabilidades, atributos, relaciones y requisitos especiales de una o varias clases (cada clase cumple los requisitos de cada realización en que participa)
  - La integridad de los paquetes (contenido correcto y dependencias con otros paquetes mínimas)
  - La integridad de uno o más subsistemas
- Debe:
  - Integrar los requisitos resultantes dentro de cada clase mediante la creación de operaciones consistentes en cada interfaz
  - Generar diagramas de clases



# Asignación de responsabilidad

- El arquitecto, al identificar clases necesarias, las va asignando a un trabajador
- Si todavía falta algo, el ingeniero de casos de uso lo comunica al arquitecto e ingenieros de componentes, se añade al modelo y se asigna a un trabajador
- El responsable de un subsistema suele ser también responsable de los elementos que lo componen (y suele seguir con todo ello en la implementación)



# Contenido

- Requisitos, Análisis, y Diseño
- Artefactos
- Trabajadores
- Análisis
- Diseño



# Tareas básicas

- Analizar y diseñar la arquitectura (arquitecto)
- Analizar y diseñar los casos de uso en términos de clases y/o subsistemas participantes y sus interfaces (ingeniero de casos de uso)
- Analizar las clases e integrar en ellas los requisitos encontrados mediante operaciones consistentes en cada interfaz (ingeniero de componentes)
- Analizar los paquetes (ingenieros de componentes)
- Diseñar cada subsistema (ingenieros de componentes)



# Restricciones

- Las realizaciones de casos de uso realizadas establecen los requisitos de comportamiento para cada clase o subsistema
- A cada paso se identifican nuevos candidatos a subsistemas, interfaces, clases y mecanismos de diseño genéricos
- Los ingenieros responsables de los subsistemas individuales deberán refinarlos y mantenerlos



# Tareas

A. Arquitectura
   - Análisis (paquetes, clases, requisitos)
   - Diseño (nodos, subsistemas, clases, mecanismos)
B. Casos de uso
C. Clases
D. Paquetes
E. Subsistemas



# A: Análisis de la arquitectura

- Identificar paquetes
- Identificar clases evidentes
- Identificar requisitos especiales comunes



# Identificar paquetes

- Opciones para agrupar:
  - casos de uso
  - funcionalidad común
  - organizar los paquetes por capas



# Agrupar por casos de uso

- Paquetes con:
  - Casos de uso requeridos para dar soporte a un proceso de negocio
  - Casos de uso requeridos para dar soporte a un actor
  - Casos de uso relacionados mediante generalización o extensión



# Agrupar por funcionalidad común

- Paquetes con:
  - Clases que gestionan las interfaces gráficas
  - Clases que gestionan la conexión con la base de datos
  - Clases que gestionan el envío y recuperación de mensajes
  - Clases que gestionan la seguridad



# Agrupar paquetes por capas

- ¡Tarea de diseño!
  - Capa específica de la aplicación (Interfaces, detalles específicos)
  - Capa general de la aplicación (Lógica de la aplicación)
  - Capa intermedia (Java.applet, Java.awt, Java.rmi)
  - Capa de software de sistema (TCP/IP)



# Aspectos comunes de los paquetes

- Qué hacer con clases que aparecen en varios paquetes
  - colocarla en su propio paquete o sin paquete
  - hacer que los otros paquetes sean dependientes de éste



# Identificar clases evidentes

- Suele ser adecuado preparar una propuesta preliminar de clases de entidad (10 o 20)
- A partir de clases del dominio o del negocio
- La mayoría aparecerán al analizar los casos de uso, al crear las realizaciones de los casos de uso
- Aquí, no identificar muchas
- Identificar conjunto provisional de asociaciones



# Identificar requisitos especiales comunes

- Por ejemplo, limitaciones o restricciones sobre:
  - Distribución o concurrencia
  - Características de seguridad
  - Tolerancia a fallos
  - Gestión de transacciones
- Son responsabilidad del arquitecto
- No siempre se pueden encontrar al principio, sino que aparecen al explorar las realizaciones
- Una clase o una realización puede tener varios requisitos especiales



# Localizar datos concretos

- Ejemplo: requisito de persistencia
  - Rango de tamaño: de los objetos que hay que hacer persistentes
  - Volumen: Número de objetos que hay...
  - Periodo de persistencia: periodo de tiempo habitual que un objeto debe ser persistente
  - Frecuencia de actualización: de los objetos
  - Fiabilidad: si deben sobrevivir en caso de caída del software o el hardware
- Estas características se califican luego para cada clase o realización que haga referencia al requisito



# A: Diseñar la arquitectura

- Identificar los siguientes elementos:
  - nodos y sus configuraciones de red
  - susbsistemas e interfaces
  - clases de diseño significativas
  - mecanismos de diseño genéricos que tratan requisitos comunes
    - como los requisitos especiales sobre persistencia, distribución, rendimiento y demás (los capturados durante el análisis)



# Reglas generales

- Considerar reutilización de partes de sistemas parecidos, o productos software generales
- Los subsistemas, interfaces... que resulten del diseño se añadirán después



# A1: Identificar nodos

- Qué nodos se necesitan y cuál debe ser su capacidad en términos de potencia de procesamiento
- Qué tipo de conexiones debe haber entre ellos y qué protocolos de comunicación
- Qué características deben tener
  - Ancho de banda, disponibilidad, calidad..
- Si se necesita capacidad de proceso redundante, modos de fallo, migración de procesos, mantenimiento de copias de seguridad de los datos...



# A2: Subsistemas e interfaces

- Dos opciones:
  - identificar al principio para dividir el trabajo de diseño, o
  - ir encontrándolos al alcanzar un tamaño que necesite descomponerse
- Algunos subsistemas representan productos reutilizados o recursos existentes en la empresa
- Su inclusión permite analizar y evaluar alternativas de reutilización



# A2.1: Subsistemas de la aplicación

- Partir de los paquetes del análisis
- Refinar si:
  - parte de un paquete de análisis puede ser compartida y utilizada por varios otros subsistemas
  - parte de un paquete de análisis se realiza mediante productos software reutilizados (capas intermedias o subsistemas software del sistema)
  - no representan reparto adecuado del trabajo
  - no representan la incorporación de un sistema heredado
  - no están preparados para una distribución directa sobre los nodos



# A2.2 Subsistemas intermedios y de software

- El software de sistema y la capa intermedia constituyen los cimientos del sistema
- Selección e integración de productos software que se compran
- Mantener libertad de acción y no hacerse dependiente: considerar cada producto que se compra como un subsistema independiente con interfaces explícitos



# A2.3 Dependencias entre subsistemas

- Se definen dependencias si sus contenidos tienen relaciones unos con otros
- La dirección de la dependencia debe ser la misma que la navegabilidad de la relación
- Si hay interfaces, las dependencias deben ir a los interfaces



# A2.4 Interfaces entre subsistemas

- Las interfaces proporcionadas por un subsistema definen las operaciones que son accesibles desde fuera del subsistema
  - Las proporcionan las clases o subsistemas de orden inferior
- Para esbozar interfaces, considerar dependencias
  - Si un subsistema tiene una dependencia que le apunta, probablemente deba proporcionar una interfaz
- Para capas inferiores, las interfaces vendrán definidas



# A3: Clases relevantes

- Acuerdo provisional para mantener homogeneidad
- Identificar clases activas



# A4: Mecanismos de diseño

- Estudiar requisitos comunes (requisitos especiales) y decidir qué tecnologías usar para tratarlos
- El resultado es un conjunto de *mecanismos de diseño*:
  - clases, colaboraciones, subsistemas cuya misión común es tratar un requisito (normalmente no funcional)



# Ejemplos de mecanismos de diseño

- La distribución transparente de objetos en una aplicación se consigue heredando de la clase java.rmi.UniCastRemoteObject (RMI)
- La persistencia puede conseguirse mediante ficheros, base de datos orientadas a objetos o bases de datos relacionales



# Tareas

A. Arquitectura
B. Casos de uso
   - Análisis (clases, interacciones, requisitos)
   - Diseño (clases, interacciones, subsistemas, requisitos)
C. Clases
D. Paquetes
E. Subsistemas



# B: Análisis de casos de uso

- Refinamiento de casos de uso como colaboración de clases de análisis
  - B1: Identificar clases que participan en el flujo de sucesos de un caso de uso
  - B2: Distribuir el comportamiento del caso de uso entre los objetos que interactúan
  - B3: Capturar requisitos especiales sobre la realización



# B1: Identificar clases

- Identificar clases de control, entidad e interfaz necesarias para realizar los casos de uso
- Esbozamos los nombres, responsabilidades, atributos y relaciones
- Puede hacer falta refinar descripciones de flujo de sucesos del caso de uso (versión requisitos)
- Tener en cuenta las dadas por el arquitecto
- Representarlas en un diagrama de clases



# B2: Describir interacciones

- Diagramas de colaboración en que aparecen:
  - actores participantes
  - objetos del análisis
  - enlaces entre ellos
- Un diagrama de colaboración por cada flujo o subflujo diferenciado
- Un diagrama de colaboración se crea:
  - comenzando por el principio del flujo de sucesos de una realización
  - identificando a cada paso qué interacciones de objetos e instancias de actor son necesarias



# B3: Identificar requisitos especiales

- Se recogen todos los requisitos sobre una realización de caso de uso
- Se identifican aquí, pero se tratan durante el diseño y la implementación
- Hacer referencia, si es posible, a los requisitos especiales comunes identificados por el arquitecto



# B: Diseñar un caso de uso

- Identificar clases de diseño y subsistemas cuyas instancias son necesarias para llevar a cabo el flujo de sucesos del caso de uso
- Distribuir el comportamiento entre los objetos (y subsistemas)
- Definir requisitos sobre las operaciones de las clases (subsistemas) y sus interfaces
- Capturar requisitos de implementación



# B1: Clases de diseño

- Estudiar clases del análisis, traza de las clases de diseño hacia las de análisis
- Estudiar requisitos especiales de la correspondiente realización de caso de uso – análisis
- Identificar clases necesarias



# B2: Descripción de interacciones

- Diagramas de secuencia, uno por cada flujo o subflujo
- Se puede partir de la realización de caso de uso – análisis
  - Pero normalmente se han añadido muchos objetos nuevos
- Transformar un diagrama de colaboración en diagrama de secuencia



# Diagramas de secuencia

- El causante de una invocación es un mensaje de una instancia del actor hacia un objeto del diseño
- Cada clase debe tener al menos un objeto que participa en un diagrama de secuencia
- Los mensajes se envían entre líneas de vida
  - Mensaje nombre temporal, el ingeniero de componentes lo asigna a un método



# B3: Subsistemas e interfaces

- Empezar por diseñar un caso de uso en función de subsistemas participantes
  - Estudiar clases del análisis que participan y su agrupación en paquetes
  - Estudiar los requisitos especiales



# B4: Interacciones entre subsistemas

- Diagramas de secuencia que contengan las instancias de los actores, subsistemas y transmisiones de mensajes entre ellos
- Diferencias:
  - Las líneas de vida denotan subsistemas en lugar de objetos
  - Cada subsistema tiene al menos una línea de vida
  - Si asignamos un mensaje a una operación de un interfaz, cualificar el mensaje con el interfaz
    - Cuando un subsistema proporciona varias interfaces, distinguir qué mensaje corresponde a cuál



# B5: Requisitos de implementación

- Requisitos no funcionales identificados durante el diseño que han de tratarse con la implementación



# Tareas

- Arquitectura
- Casos de uso
- Clases
- Paquetes
- Subsistemas



# C: Análisis de las clases

- Identificar y mantener responsabilidades (basadas en su papel en las realizaciones)
- Identificar y mantener los atributos
- Identificar y mantener las relaciones
- Capturar los requisitos especiales sobre la realización



# Criterios

- Recopilar todos los roles que cumple en diferentes realizaciones de caso de uso (mirando los diagramas de clase y de interacción)
- Extraer las responsabilidades de un rol detrás de otro, añadiendo responsabilidades adicionales o modificando las existentes (pasando de una realización a otra)



# Requisitos especiales

- Requisitos de una clase identificados pero no tratados
- Tener en cuenta los requisitos especiales de la realización del caso de uso, que pueden tener requisitos sobre la clase
- Ejemplo de requisito de persistencia:
  - Rango de tamaño: 2 a 24 Kbytes por objeto
  - Volumen: hasta 100.000
  - Frecuencia de actualización:
    - creación/borrado : 1.000 al día
    - actualización: 30 actualizaciones a la hora
    - lectura: 1 acceso a la hora



# C: Integrar requisitos en clases

- Crear una clase de diseño que cumpla su papel en las realizaciones de casos de uso y los requisitos no funcionales aplicables
- Tener en cuenta:
  - sus operaciones
  - sus atributos
  - relaciones en que participa
  - métodos (que realizan las operaciones)
  - estados impuestos
  - dependencias con cualquier mecanismo de diseño genérico
  - requisitos relevantes a su implementación
  - correcta realización de cualquier interfaz requerida



# Tareas

- Arquitectura
- Casos de uso
- Clases
- Paquetes
- Subsistemas



# D: Análisis de paquetes

- Garantizar que el paquete es lo más independiente posible de otros
- Garantizar que el paquete cumple su objetivo de realizar algunas clases del dominio o casos de uso
- Describir las dependencias (para estimar el efecto de cambios futuros)



# Criterios

- Definir y mantener las dependencias del paquete con otros
- Asegurarse de que el paquete contiene las clases correctas (que sea cohesivo: sólo contiene objetos relacionados funcionalmente)
- Limitar las dependencias con otros paquetes (reubicar aquellas clases que hacen este paquete muy dependiente de otros)



# Tareas

- Arquitectura
- Casos de uso
- Clases
- Paquetes
- Subsistemas



# D: Diseño de subsistemas

- Se debe garantizar que
  - el subsistema es tan independiente como sea posible de los demás y/o de sus interfaces
  - proporciona las interfaces correctos
  - el subsistema ofrece una realización correcta de las operaciones tal y como se definen en las interfaces



# D1: Mantenimiento de dependencias

- Si un subsistema proporciona interfaces, las dependencias deben declararse hacia las interfaces en vez de hacia el subsistema
- En cualquier caso, minimizar dependencias



# D2: Mantenimiento de interfaces

- Debe soportar todos los roles que cumple el subsistema en las diferentes realizaciones de caso de uso
- Puede ser necesario refinar las interfaces que ya estuvieran definidas



# D3: Mantenimiento de contenidos

- Por cada interfaz del subsistema, clases u otros subsistemas que lo proporcionen
- Se puede estructurar el subsistema en forma de colaboraciones de sus elementos



# Fuente

- I. Jacobson, G. Booch, J. Rumbaugh "El Proceso Unificado de Desarrollo de Software", Addison Wesley, 2000