APUNTES INGENIERÍA DEL SOFTWARE
CURSO 2016-2017

TEMA 3: INTRODUCCIÓN AL DISEÑO ORIENTADO A OBJETOS

INTRODUCCIÓN AL DISEÑO DE SOFTWARE

Tenemos las siguientes etapas en el desarrollo del Software:

- Análisis de Requisitos → ¿Qué sistema hay que construir?
- Especificación → ¿Que ha de hacer el sistema?
- Diseño → ¿Cómo lo hace el sistema?
- Implementación

Las dos primeras etapas son independientes de la tecnología.

Punto de partida:

- Resultado de la especificación: ¿Qué ha de hacer el sistema?
- Tecnología: ¿Con qué recursos? (Recursos hardware y software disponibles)

A partir del punto de partida tenemos el proceso de diseño. Actividad de aplicar distintas técnicas y principios con la finalidad de definir un sistema con suficiente detalle para que se pueda implementar.

Resultado: ¿Cómo o hace el sistema?

- Arquitectura del sistema software:
  Descripción de los subsistemas y componentes de un sistema software y las relaciones entre ellos.
- Diseño detallado del sistema software
  Diseño de los componentes del sistema.

DISEÑO EN UML

ESPECIFICACIÓN	→	DISEÑO
El sistema software se ve como una sola clase de objetos que engloba toda la información y todas las operaciones.		Cada clase de objetos tiene sus propias operaciones de manipulación de información. Los objetos interactúan para Satisfacer las operaciones del sistema.

Resultado de la especificación:

- Modelo de casos de usos. ¿Qué interacción hay entre los actores y las funciones del sistema software?
- Esquema conceptual de datos. ¿Cuáles son los conceptos relevantes del mundo real de referencia?
- Diagrama de secuencia del sistema. ¿Qué respuestas da el sistema a los eventos externos? ¿Qué operaciones ha de tener el sistema?
- Contrato de las operaciones. ¿Qué hacen las operaciones del sistema?

Resultado del diseño:

- Modelo de casos de uso. Define la interacción real, con una interfaz concreta.
- Diagrama de clases de diseño. Describe las clases del software y sus operaciones.
- Diagramas de secuencia. Define la interacción entre las clases de objetos para responder a un evento externo.
- Contratos de las operaciones. Definen qué hacen las operaciones de las clases de objetos.

El enfoque de los modelos es muy diferente en las dos etapas.

Especificación

- Los modelos definen los conceptos del mundo real (dominio del problema).
- No tiene en cuenta las propiedades a lograr ni la tecnología que se usará para implementar el sistema software.

Diseño

- Los modelos definen los conceptos que se desarrollarán para proporcionar una solución a las necesidades del mundo real (dominio de la solución).
- Puede ser necesario cambiar los modelos de especificación para conseguir determinadas propiedades.

ARQUITECTURA DEL SOFTWARE

Dependiendo de:

- Propiedades que se quiere que cumpla el sistema(requisitos no funcionales).
- Recursos tecnológicos disponibles
  - Lenguajes de programación.
  - Sistemas de gestor de base de datos / Sistema de ficheros.
  - Etc…

Se determina

- La arquitectura del sistema software.
- Patrones que se usarán en el diseño del sistema.

PATRONES DE DISEÑO

Patrón arquitectónico

- Expresa un esquema de organización estructural para sistemas software.
- Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas guías para organizar las relaciones entre ellas.

Patrón de diseño

- Una vez determinado el patrón arquitectónico, un patrón de diseño da un esquema para refinar sus subsistemas o componentes, o las relaciones entre ellos.
- Describe la estructura de una solución a un problema que aparece repetidamente.
- Resuelve un problema de diseño general en un contexto particular.

Modismo

- Describe la estructura de la solución dependiendo del lenguaje de programación.

PATRÓN ARQUITECTÓNICO: ARQUITECTURA EN CAPAS

Contexto

- Un sistema grande que requiere su descomposición en grupos de subtareas (componentes) tales que cada grupo de subtareas está en un nivel determinado de abstracción.

Problema

- Hay que diseñar un sistema con la característica dominante de incluir aspectos de alto y bajo nivel(de abstracción), donde las tareas de alto nivel se basan en las de bajo nivel.
- Las tareas de alto nivel no se pueden implementar utilizando directamente los servicios de la plataforma, a causa de su complejidad. Se necesitan servicios intermedios.
- Fuerzas a equilibrar.
  - Cambios en el código no deberían propagarse en todo el sistema (mantenible).
  - Los componentes se deberían poder reutilizar y reemplazar por implementaciones alternativas (reusabilidad, separación de interfaz e implementación).
  - Responsabilidades similares se deberían agrupar para favoreces la comprensibilidad y mantenibilidad (cohesión).
  - Se desea portabilidad a otras plataformas.

Solución

- Estructurar el sistema en un número apropiado de capas.
- Colocar las capas verticalmente.
- Todos los componentes de una misma capa han de trabajar en el mismo nivel de abstracción.
- Los servicios que proporciona la capa j utilizan servicios proporcionados por la capa j-1. Al mismo tiempo, los servicios de la capa j pueden depender de otros servicios en la misma capa.

Tenemos diferentes capas:

- Presentación. Sabe cómo presentar los datos al usuario, pero no sabe qué transformaciones hay que hacer para dar respuesta a las peticiones del usuario.
- Dominio. La capa de dominio sabe cómo satisfacer las peticiones del usuario, pero ignora dónde se guardan los datos y cómo se presenta al usuario.
- Gestión de datos. La capa de gestión de datos sabe dónde y cómo están almacenados los datos, pero desconoce cómo tratarlos. Transforma operaciones conceptuales en operaciones físicas.
- Sistema de gestión bases de datos / ficheros. Se encarga de mantener una representación persistente y concreta del estado de dominio.

CAPA DE DOMINIO (Patrones GRASP)

Patrones de principios generales para asignar responsabilidades.

- Principios fundamentales para guiar la asignación de responsabilidades a objetos.
- La asignación de responsabilidades a objetos se hace durante el diseño, al definir las operaciones de cada clase de objetos.

- La asignación de responsabilidades se muestra en los diagramas de interacción mediante mensajes que se envían a las clases de objetos.

Hay algunos patrones GRASP básicos: Experto en información, Creador, Controlador, Alta cohesión, bajo acoplamiento.

Patrón Experto en Información

- Contexto: Asignación de responsabilidades a objetos.
- Problema: ¿Cuál es el principio general para asignar responsabilidades a los objetos? Decidir a qué clase hemos de asignar una responsabilidad concreta.
- Solución:
  - Asignar una responsabilidad a la clase que tiene la información necesaria para realizarla.
  - La aplicación de patrón requiere tener claramente definidas las responsabilidades que se quieren asignar (postcondiciones de las operaciones).
  - No siempre existe un único experto, sino que puede existir diversos expertos parciales que tendrán que colaborar.

Patrón Creador

- Contexto: Asignación de responsabilidades a objetos.
- Problema: ¿Quién ha de tener la responsabilidad de crear una instancia de una clase?
- Solución: Asignar a la clase B la responsabilidad de crear una instancia de una clase A si satisface una de las siguientes condiciones:
  - B es un agregado de objetos de A.
  - B contiene objetos de A.
  - B tiene los datos necesarios para iniciar un objeto de A (B tiene los valores de los parámetros del constructor de A).
  - B usa muchos objetos de A.
  - B guarda instancias de objetos de A.

Patrón de diseño controlador

- Contexto: los sistemas software reciben eventos externos. Una vez recibido un evento en la capa de presentación, algún objeto de la capa de dominio ha de recibir este evento y ejecutar las acciones correspondientes.
- Problema: ¿Qué objeto es el responsable de tratar un evento externo?
- Solución:
  - Asignar esta responsabilidad a un controlador.
  - Un controlador es un objeto de una clase (no pertenece a la interfaz usuario).
  - Posibles controlador:
    - Controlador de Fachada: Una clase que representa todo el sistema.
    - Controlador de Caso de uso: Una clase que representa una instancia de un caso de uso.

Acoplamiento

- Contexto: Asignación de responsabilidades a objetos.
- Problema: ¿Cómo soportar bajas dependencias, bajo impacto del cambio e incremento de reutilización?
  - Hay un acoplamiento de la clase A a la clase B si:
    - A tiene un atributo de tipo B
    - A tiene una asociación navegable con B.

        - B es un parámetro o el retorno de una operación de A.
        - Una operación de A hace referencia a un objeto de B.
        - A es una subclase directa o indirecta de B.
- Solución: El acoplamiento entre clase ha de ser bajo para:
    - Evitar que cambios en una clase tengan efectos sobre otras clases.
    - Facilitar l comprensión de las clases (sin recurrir a otras).
    - Facilitar la reutilización.

Cohesión

- Contexto: Asignación de responsabilidades a objetos.
- Problema: ¿Cómo mantener la complejidad controlable?
- Solución: Asignar responsabilidades de manera que la cohesión permanezca alta.
    - Una clase con cohesión alta:
        - Tiene pocas responsabilidades en un área funcional.
        - Colabora con otras clases para hacer las tareas.
        - Suele tener pocas operaciones muy relacionadas funcionalmente.
        - Ventajas: fácil comprensión y fácil reutilización y mantenimiento.
    - Una clase con cohesión baja:
        - Tiene muchas responsabilidades de distintas áreas funcionales.
        - Tiene muchas operaciones poco relacionadas funcionalmente.
        - Inconvenientes: difícil de comprender, difícil de reutilizar y de mantener, sensibles a los cambios.

DIAGRAMAS DE INTERACCIÓN

- Interacción: Comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos dentro de un contexto para lograr un propósito.
- Mensaje: Especificación de una comunicación entre objetos que transmite información, con la expectativa de desencadenar una actividad.
- Se describe cómo los objetos colaboran entre sí para realizar cierta actividad.
- Dos tipos de de Diagramas de Interacción:
    - Diagramas de Secuencias.
      Destacan la ordenación temporal de los mensajes.
    - Diagramas de Colaboración.
      Destacan la organización estructural de los objetos participantes.
- Equivalencia semántica(prácticamente isomorfos).
- Notación general diagramas interacción: Instancia de una clase, mismo símbolo con texto subrayado.

DIAGRAMAS DE SECUENCIA

- Representan escenarios.
- Incluye:
    - Objetos y su linea de vida.
    - Mensajes.
    - Información de control: condiciones y marcas de iteración.
    - Indicar el objeto devuelto por el mensaje: return.
- Focos de control y cajas de activación. Se pueden mostrar los focos de control (llamada de rutina ordinaria, la operación está en la pila de llamadas) utilizando cajas de activación.
- Mensajes a “self” o “this”. Mensaje que se envía de un objeto a él mismo utilizando caja de activación anidad.

- Creación de objetos. Mensaje de creación apunta al símbolo del objeto.
- Destrucción de objetos. Fin de la linea de vida de un objeto.
- Mensajes condicionales.
- Mensajes condicionales mutuamente exclusivos.
- Iteración para un único mensaje.
- Iteración de una serie de mensajes.
- Iteración sobre una colección(multiobjeto).
- Mensajes a objetos de clase.

DIAGRAMAS DE COLABORACIÓN

- Representa escenarios.
- Incluye:
  - Objetos y ENLACES (relaciones estructurales objetos).
  - Mensajes.
  - Información de control: condiciones y marcas de iteración.
  - Indicar el objeto devuelto por el mensaje: return.
- Enlaces. Camino de conexión entre dos objetos. Indica que es posible alguna forma de navegación entre los objetos.
- Mensajes. Expresión del mensaje y flecha que indica la dirección. Añadir número de secuencia para mostrar el orden secuencial.
- Mensajes a “self” o “this”. Mensaje desde un objeto a él mismo.
- Creación de instancias. Convenio: mensaje create. Si se utiliza otro nombre, añadir estereotipo “create”. Mensaje create puede incluir parámetros(valores iniciales).
- Secuencia de números de mensajes. El orden de los mensajes se representa mediante número de secuencia. No se numera el primer mensaje.
- Mensajes condicionales.
- Caminos condicionales.
- Iteración o bucle.
- Iteración sobre una colección.
- Mensaje a un objeto clase.

DIAGRAMA SECUENCIA VS DIAGRAMA COLABORACIÓN

- Equivalencia semántica.
- Simples para comportamientos simples.
- Si hay mucho comportamiento condicional, usar diferentes escenarios.
- Diagrama de secuencia muestra mejor el orden en que se ejecutan los mensajes.
- Diagrama de colaboración muestra mas claramente los objetos con los que interactúa un determinado objeto.

DIAGRAMA DE CLASES DE DISEÑO

- A partir de los Diagramas de Interacción realizados durante el Diseño, obtener el Diagrama de Clases de Diseño.
  - Clases que participan en las interacciones.
  - Relaciones entre las clases necesarias para que las clases colaboren.
  - Atributos de las clases de diseño.
  - Operaciones de las clases de diseño: mensajes que reciben las clases.
- El Diagrama de Clases de Diseño != Diagrama de Clases Conceptual.