Introducción al Diseño con Patrones

Diseño de Sistemas Informáticos

AYER: Programación estructurada

- pero sigue en uso...
- Separación entre diseño y codificación
- Parte de la idea de que la mayoría de los errores se introducen durante el diseño.
- Supone que los requerimientos se conocen de antemano y permanecen inalterados durante el ciclo de vida.

  (Horowitz 1993) Cambios o nuevos requerimientos:

30% del coste de desarrollo
70% del coste del mantenimiento

- Alternativa: *Desarrollo en espiral*
  - El diseño e implementación procede de forma incremental con gran interacción con el usuario a fin de mitigar riesgos.
  - Ejemplo: *Rapid Structured Prototyping*
- *Inconveniente*: Separación del modelo de datos del modelo del proceso.
  - Cambios en uno pueden ocasionar efectos imprevisibles en el otro.

HOY: Análisis, Diseño e Implementación OO

- Orientación a Objetos:
  - Análisis, Diseño: UML
  - Implementación: Lenguaje OO (Java, C++, Delphi,...)
- *Suposición fundamental*: los objetos del dominio son las entidades más estables del sistema.
  - Un diseño basado en estos objetos del dominio será más resistente al cambio.
  - Fusiona los modelos de proceso y de datos
  - Ventaja adicional: reduce el vacío entre el software y los modelos de negocio.
- Tres elementos fundamentales:
  - *Análisis OO*, analizar el dominio del problema para construir un modelo del mundo real utilizando objetos
  - *Diseño OO*, refinamiento de los modelos de análisis para crear especificaciones adicionales que enriquecen el modelo de análisis con detalles próximos a la implementación
  - *Implementación*, codificación del diseño *posiblemente* utilizando un lenguaje OO

Análisis vs Diseño OO

- Limites difusos entre análisis y diseño OO
  - *Análisis*: **investigación**
    (descripción del problema y requerimientos)
  - *Diseño*: **solución lógica**
    (forma en que se cumplen los requerimientos: asignación de responsabilidades, interacciones entre objetos, etc.)

- Soporte
  - Análisis de requerimientos: *casos de uso*
  - Análisis: *modelo conceptual*

    no es una descripción de componentes software, sino que representa el dominio del problema.
  - Diseño: *diagrama de clases, colaboración, estado*

- Cuestiones claves en el diseño OO:

  ¿cómo se asignan responsabilidades a las clases?
  ¿cómo deben interactuar los objetos?
  ¿qué clases deberían hacer qué?

Diseño e Implementación OO

- Problema: Sigue habiendo interacciones molestas debidas al uso inadecuado de la OO, por ejemplo, mal empleo de la encapsulación.

- Problema: Vacío existente entre análisis, diseño e implementación
  - El programador debe tomar decisiones de diseño
  - Sobrespecificación

- Nuevo enfoque: **Diseño con Patrones**
  - Nuevo collar para un perro viejo: *Abstracción*
  - Idea de *Christopher Alexander* (arquitecto), que aplica el concepto a la construcción urbanística.

(1977) *A Pattern Language: Towns, Buildings, Construction*

(1979) *The Timeless Way of Building*

Diseño con Patrones

- *Objetivo*: Agrupar una colección de soluciones de diseño que son válidas en distintos contextos y que han sido aplicadas con éxito en el pasado.

- *Patrón de Diseño*: Solución a un problema de diseño no trivial que es *efectiva y reusable*.
  - Es *efectiva* en el sentido de que ya resolvió el problema con anterioridad satisfactoriamente.
  - Es *reusable* si se puede aplicar a un abanico de problemas de diseño en distintas circunstancias.

- Los patrones son soluciones de sentido común que ya deberían formar parte del conocimiento de un diseñador experto.

- Facilitan la comunicación entre diseñadores al establecer un marco de referencia: terminología, justificación.

- Facilitan el aprendizaje al diseñador inexperto

- *Esencia*: establecer parejas problema-solución.

Aportaciones de los Patrones de Diseño

- *Identificar los Objetos Apropiados*

  Parte complicada de la OO: Descomponer un sistema en objetos (encapsulación, granularidad, dependencias, flexibilidad, rendimiento, evolución, reusabilidad, ...)

- *Determinar la Granularidad de los Objetos*

- *Especificar las Interfaces*

  Identifican los elementos clave en las interfaces y las relaciones existentes entre distintos interfaces

  Facilita la distinción "subclase o subtipo"

- *Especificar las Implementaciones*

- *Reutilizar*

  Facilita la decisión "herencia o composición" (herencia de caja blanca frente a herencia de caja negra)

  Gamma et al.:

  - Favorecer composición sobre herencia
  - Uso de la delegación

- *Relacionar Estructuras en Tiempo de Compilación y en Tiempo de Ejecución*

- *Diseñar para el Cambio*

Componentes de un Patrón (Culinario)

- *Nombre*
- *Descripción*
- *Clasificación*
- *Ejemplo motivador*
- *Contexto/alcance*
- *Componentes involucrados*
- *Algoritmo de preparación*
- *Recetas relacionadas*

Componentes de un Patrón de Diseño (GoF)

- *Nombre y Clasificación*
  - Estructural, creación, comportamiento
- *Propósito* (Intent)
- *Otros nombres* (Also Known As)
- *Motivación*
  - Escenario que presenta un problema de diseño
  - Solución para el escenario utilizando el patrón
- *Aplicabilidad* (situaciones en las que se puede aplicar)
- *Estructura* (diagrama de clases, colaboración, estado)
- *Participantes*
  - Clases y objetos participantes
  - Responsabilidades
- *Colaboraciones entre participantes*
- *Consecuencias* (ventajas e inconvenientes de su aplicación)
- *Implementación* (técnicas y peligros en la implementación)
- *Código de Ejemplo*
- *Usos conocidos* (ejemplos del patrón en sistemas reales)
- *Patrones relacionados*