Desarrollo de software en sistemas distribuidos

Curso 2015

OOAD Vs SOAD

- Filosofía de diseño de software identificando unidades o entidades del mundo real.
- En ambos subyace la idea de reutilización de código
- En ambos se busca minimizar el impacto de los cambios en el software en produccion.

- OOAD
  - Reusabilidad basada en polimorfismo y herencia
  - Los objetos se asocian a la resolución de problemas vinculados a una aplicación particular
- SOAD
  - No existe el concepto de herencia
  - Requieren estabilidad y accesibilidad

Cuando usar servicios?

Algunas de las pautas a tener en cuenta son:

- La cantidad de trabajo realizada por el servicio debe ser acorde al trabajo adicional provocado por su diseño más el *overhead* de ejecución que provoca.
- La funcionalidad del servicio debe ser utilizada por dos o más clientes.
- La implementación del servicio evoluciona mientras que la funcionalidad que aporta se mantiene relativamente estable.

Objetivos del OOAD Vs Objetivos SOAD

- *Cumplimiento con los requerimientos del negocio* incrementado (alineación del negocio con IT)
- *Robustez* incrementada (interoperabilidad)
- *Extensibilidad* incrementada (carga de IT reducida y diversificación de vendedores)
- *Flexibilidad* incrementada (agilidad organizacional)
- *Reusabilidad y productividad* incrementada (ROI o retorno de inversión)

Comparación de conceptos

- Clases y objetos

  La clase define tanto interfaces publicas como privadas, los servicios solo publicas

- Métodos y atributos

  Las clases definen métodos (comportamiento) y atributos (propiedades o estados de la clase) y ambos pueden ser tanto públicos como privados.

  Los contratos de servicios sólo definen capacidades abstractas que serán métodos si el servicio se implementa con componentes y serán operaciones si la implementación es por Web Services.

Comparación de conceptos

- Mensajes

  En la OO los mensajes tienen un formato binario y son sincrónicos

  En los servicios se intercambian unidades de información basados en texto y lo hacen tanto sincrónica como asincrónicamente

- Interfaces

  Una interface reúne un conjunto de métodos relacionados que pueden ser implementados por una clase.

  La orientación a servicios define tanto el contrato de servicio como su solución lógica subyacente.

Comparación de principios de diseño

- Encapsulamiento

  Encapsular significa encerrar algo en un contenedor. En la OO esto se ve reflejado en el concepto de "ocultamiento de la información". En la OS el encapsulamiento se refleja en la idea de abstracción de servicio con un deliberado ocultamiento de la información. Refiere el "que" de la implementación del servicio.

- Herencia

  Es un concepto descartado en la OS para enfatizar la autonomía individual de los servicios y reducir el acoplamiento entre ellos.

  En la OO constituye un patrón de diseño fundamental a la hora de reutilización de comportamiento.

Comparación de principios de diseño

- Generalización/Especialización

  La generalización en la OO representa la relacion "es una clase de" mientras que la especialización representa la relacion "es un".

  En la OS el concepto es similar pero no existe la herencia. En el SOAD, la especialización o generalización se vinculan con la granularidad del servicio. Cuanto más especializado es un servicio, mayor es su nivel de granularidad

Comparación de principios de diseño

- Abstracción

  La abstracción en la OO consiste en crear clases simplificadas que ocultan detalles de complejidad

  En OS también se persigue el mismo objetivo. La diferencia es que en la OS no se hace presente la herencia ni la noción de clase abstracta.

- Polimorfismo

  Es la capacidad de responder al mismo mensaje con diferentes resultados en función de la sub-clase que lo ejecute.

  Dado que no existe la herencia en la OS, no existe esta clase de polimorfismo.

Comparación de principios de diseño

- Single Responsibility Principle (SRP) o cohesión

  En el diseño OO la lógica se circunscribe en torno a un propósito único. En la OS cada servicio establece un contexto funcional desde el momento que su propósito es automatizar un proceso de negocio específico.

Comparación de principios de diseño

- Delegación

  Este principio de la OO requiere que la lógica que ya existe en otro objeto sea delegada para que este último la lleve a cabo. Este concepto aplica perfectamente a la idea de composición de servicios en la OS

- Asociación

  En la OO la asociación es una relación. Esta relación puede ser "used-by" o "has-a". La interacción se servicios equivale a la primera relación ya que esta no requiere vinculación entre las clases, simplemente usan funcionalidad una de otra.

Comparación de principios de diseño

- Composición
  - La idea de composición es similar en la OO y la OS, pero al igual que el encapsulamiento, se utiliza de manera diferente.
  - En la OO la idea de componer objetos o clases tiene directa vinculación con la relación "has-a "
  - En la OS, componer servicios equivale a ensamblarlos o agregarlos, sin una estructura propia predefinida
- Agregación
  - La agregación en la OO es similar a la composición pero con reglas diferentes.

Comparación de principios de diseño

- En resumen ….

Las principales diferencias entre OO y OS se sustentan en que los servicios no respetan una estructura jerárquica y no existe el concepto de herencia, mientras que la mayoría de las similitudes se vinculan al hecho que ambos persiguen los mismos objetivos.

Guía para el diseño clases orientadas a servicios

- **Implementar interfaces de las clases**

  Para poder concebir las clases como servicios, estas deben siempre implementar interfaces, de modo de mantener el contrato publico separado de la clase.

- **Limitar el acceso a las clases a través de las interfaces**

  Esta forma positiva de acoplamiento protege los detalles de clase que subyacen la aplicación, de la misma forma que previene las formas negativas de acoplamiento dentro de los servicios que existen como Web
  Services.

Guía para el diseño clases orientadas a servicios

- **No definir atributos públicos en las interfaces**
  Este principio ya existe en el OOAD. Remover los atributos públicos para acceder a los servicios fuerza la comunicación a través de métodos (mensajes)
- **Utilizar la herencia con cuidado**
  El uso de herencia se contrapone con la libertad y autonomía de los servicios.
  Según los patrones de diseño de servicio, podemos preparar para la descomposición de servicios por la forma en que diseñamos un contrato de servicios y la lógica. La lógica de servicio compuesto por componentes
  fuertemente unido a través de estructuras herencia será más difícil
  de descomponer que si las estructuras de clase subyacentes son
  menos interdependientes.

Guía para el diseño clases orientadas a servicios

- **Evitar relaciones "has-a" entre servicios**
  - La composición de servicios requiere libertad para permitir que la composición de sus miembros actúe separados de su ancestro iniciador o controlador.
- **Usar clases abstractas para modelar. No para diseñar.**
  - Las clases abstractas no se requieren en un OS dado que no existe una relación de herencia formalmente definida.
  - Sin embargo, pueden ser útiles en la etapa de análisis como base de colecciones de clases/servicios relacionados.

Guía para el diseño clases orientadas a servicios

- **Usar facade de clases**
  - El uso de facade es un patrón de diseño, no un principio de diseño. Desde la perspectiva de diseño de servicios crear clases facade es un practica útil e importante.

Guía para el diseño clases orientadas a servicios

- **En resumen …**

  Las clases orientadas a servicios pueden ser diseñadas usando principios de diseño OS y regulando el uso de ciertos principios OO, como por ejemplo el uso de la herencia.

Lecturas recomendadas

- "SOA Principles of Service Design" by Erl, Thomas. Prentice Hall. 2007 ISBN-13: 9780132344821. Capitulo 14