Arquitectura de Software

Vista de módulos (Estilos)

Índice

- Estilo de descomposición
- Estilo de uso
- Arquitecturas por capas
- Estilo de aspectos
- Relación de generalización
- Modelo de datos. *No lo discutiremos*

Estilo de descomposición

- Lo utilizamos para representar la relación "Es parte de"
- Descomponemos el sistema en unidades de implementación
- Describe la organización del código en módulos y sub-módulos, mostrando cómo se distribuyen las responsabilidades

Estilo de descomposición ...

- Es un buen punto para comenzar a diseñar la arquitectura del sistema: divide y vencerás
- Identifica los módulos que aparecerán en los otros estilos/vista (uso y "capas")

Estilo de descomposición ...

- Criterios para descomponer en módulos
  - En IS de acoplamiento/cohesión
  - Distribuir el trabajo
    - Equipos trabajando en paralelo
    - Habilidades de los equipos
  - Decisiones de comprar frente a construir
  - Conseguir atributos de calidad
    - P.e. Modificabilidad: ocultación de información
  - *Documentar el criterio utilizado*

Estilo de descomposición ...

- Heurística
  - Un módulo es suficientemente pequeño si puede ser desechado e implementarlo de nuevo en caso de que el programador(es) abandone(n) el proyecto

Estilo de descomposición ...

- ¿Para qué sirve?
  - Es uno de los primeros documentos que daremos a quien se incorpore al proyecto
    - Servirá para entender cómo se distribuyen las responsabilidades del proyecto
  - Para navegar por la estructura del proyecto
    - Localizar cambios
  - Para distribuir el trabajo

Estilo de descomposición ...

- Restricciones
  - Un módulo como máximo pertenece a un agregado
  - No puede haber bucles
- Notación informal
  - Gráfica o textual
  - Ejemplo Figura 2.4
- Notación UML
  - El símbolo de "paquete" lo podemos usar para representar módulos
  - Ejemplos de las Figuras 2.2. y 2.3

Estilo de descomposición ...

- Relación con otras vistas
  - Componentes y Conectores
    - Debemos conocer la transformación entre la vista de descomposición y al menos una de C&C
    - La transformación es muchos a muchos
  - Está completamente relacionado con el estilo de asignación de trabajo de la vista de distribución

Estilo de uso

- Lo utilizamos para representar una forma de "dependencia", la relación de "uso"
- Simplemente nos dice para un módulo qué otros módulos usa
  - A usa B si para funcionar correctamente depende de una implementación correcta de B

Estilo de uso ...

- No hay restricciones
  - Pero si existen bucles, cadenas largas de dependencia, muchos módulos que dependen de muchos otros q será difícil entregar el sistema de manera incremental

Estilo de uso ...

- Notación UML
  - Figura 2.7
- Relación con otros estilos
  - Va de la mano con el de "capas" que debe precederlo

Estilo de uso ...

- ¿Para qué sirve?
  - Planificar el desarrollo por incremento gradual. Ejemplo Figura 2.6
  - Pruebas
  - Estimar los efectos de los cambios
- Notación informal
  - Matriz de dependencias
    - Figuras 2.8 y 2.9

Arquitectura por capas

- Se vio en Ingeniería del Software
  - Técnica para obtener módulos (subsistemas) con poco ***acoplamiento***
- Agrupamos en una capa los módulos que proporcionan un conjunto de servicios relacionados (***cohesión***)
- Las capas dividen completamente el software

Arquitectura por capas ...

- Los servicios se hacen públicos a través de las **interfaces** de la capa ę "ocultar información"
- Propiedad fundamental (ver Figura 2.17)
  - Relación de orden estricta
    - (A,B): A "puede usar" cualquier servicio de B
    - (A,B): B no puede usar ningún servicio de A
  - Está permitido "saltarse capas" q arquitecturas abiertas
    - Estos sistemas son menos portables y modificables

Arquitectura por capas ...

- Restricciones
  - Al menos dos capas
  - Cada módulo pertenece a una capa
  - No hay relaciones circulares
- M1 usa M2 ¿Cómo los organizamos en la misma capa o en diferentes?

Arquitectura por capas ...

- Las capas sirven para crear y comunicar la arquitectura pero no se hacen explícitas en el código
- Examinando el código podemos sacar la relación de "uso" pero no las "capas"

Arquitectura por capas ...

- Cuando creamos capas esperamos que
  - Equipos con diferentes capacidades trabajarán en ellas
  - Re-utilización de las capas
  - Módulos que evolucionarán juntos en el tiempo

Arquitectura por capas ...
¿Para qué sirve?

- El objetivo de conseguir una arquitectura por capas es:
  - ***Modificabilidad***
  - ***Portabilidad***
  - ***Mantenibilidad***

Arquitectura por capas ...
¿Para qué sirve?

- Cada capa aplica el principio de "ocultar información"
  - Un cambio en la implementación de una capa inferior queda oculto por sus interfaces y no tendrá consecuencias en las capas superiores
  - Es lo que promueve la *portabilidad*
  - Sin embargo los cambios en el rendimiento sí se propagan
- Las interfaces portables no son dependientes de la plataforma q son abstractas

Arquitectura por capas ...
¿Para qué sirve?

- Para crear y comunicar la arquitectura
  - Las capas y sus interfaces sirven para manejar la complejidad de grandes sistemas q promueve la *mantenibilidad*
  - Junto a la vista de descomposición es una forma para entender el código -*mantenibilidad*- y para asignar trabajo

Arquitectura por capas ...

- Documentar propiedades de las capas
  - Hay que describir para qué sirve la capa, la lista de módulos y cómo se implementan
  - Para cada capa las capas que puede usar
  - Argumentar el *criterio* que se ha utilizado para crear las capas

Arquitectura por capas ...

- Notación UML
  - No proporciona un constructor explícito para capas. Las representaremos como un paquete estereotipado con <<layer>>
  - Figura 2.24
- Notación informal
  - Figuras 2.18, 2.19, 2.21, 2.22, 2.23

Relación con otros estilos ...

- Vista de descomposición
  - Establecer correspondencia entre módulos de vistas (capas y descomposición)
  - Dos submódulos de un mismo módulo pueden pertenecer a capas diferentes
  - Un módulo puede aparecer en más de una capa. Figura 2.25

Aspectos

- Se utiliza para "aislar" en la arquitectura los módulos que implementan aspectos *transversales*
- Algunos módulos mezclan la "lógica de negocio" con lo "transversal"
  - Ejemplo sistema para banca: Módulos para Clientes, Cuentas, ATM. Todos ellos tienen control de acceso, gestión de transacciones, …
  - Figuras 2.36 y 2.37

Aspectos ...

- En programación orientada a aspectos esta solución es natural
  - Con otros paradigmas podemos aislar estos módulos e implementarlos como librerías, interfaces, herencia, …
- ¿Para qué sirve?
  - El sistema es más fácil de modificar

Aspectos ...

- Notación UML
  - Clases estereotipadas <<aspect>>
  - Es válido no identificar cada clase que se ve afectada por un aspecto q No detallemos más la arquitectura que la implementación!
  - Figuras 2.33 y 2.34
  - Figura 2.35

Generalización

- Es el mismo concepto que hemos manejado en orientación a objetos
  - Relación is-a
- En OO hablamos de generalización entre clases, ahora hablamos de generalización entre módulos
  - En la vista de módulo una clase la podemos considerar como un módulo

Generalización ...

- Podemos representar herencia de implementación y herencia de interfaces
  - En OO era "más importante" la herencia de comportamiento -Análisis-, de implementación -Implementación-
  - Al describir la arquitectura ponemos más énfasis en compartir y reutilizar interfaces

Generalización ...

- Notación UML
  - Los módulos se suelen expresar como clases o interfaces
  - Figuras 2.12, 2.13

Generalización ...

- Podemos usar generalización en cualquier vista/estilo de módulos
- Podemos combinar cualquier estilo para presentar una vista de módulos
  - Evitar sobrecargar los diagramas
  - Cuidado con el estilo de capas q no se aplica a todos los sistemas
  - Figura 2.15

Puntos a discutir

1. ¿Crearías un modelo de datos utilizando el diagrama ER para un sistema que no tenga BBDD? ¿En qué situaciones y por qué?
2. ¿Cuál es la relación de un diagrama de clases UML con los estilos vistos?¿Muestra descomposición,uso,generalización,…?
3. Si nuestro sistema tiene módulos COTS ¿En qué vista los mostramos y por qué?
4. Discutir generalización y herencia
5. Pensad un sistema que no pueda ser descrito por capas. Si no es por capas ¿qué nos dice esto sobre la relación “puede usar”?
6. Los diagramas de la Figura 2.46 son de sistemas reales. ¿Cómo representan la relación “puede usar”?Crear una leyenda para que no haya ambigüedad