

Arquitectura del Software

Arquitectura del Software

- Segons Wikipedia:
 - Indica l'estructura, funcionament i interacció entre les diverses parts del software.
 - És un nivell de disseny que se centra en l'estructura global del problema.
- De manera informal
 - Organització general del codi.

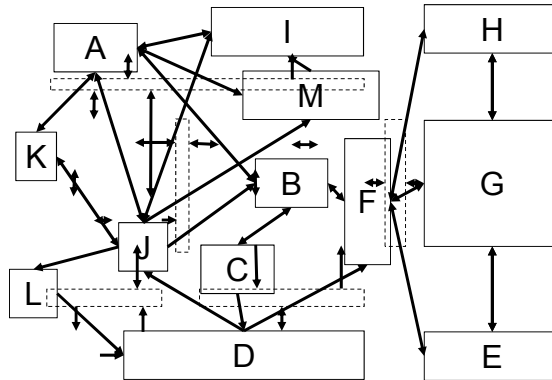
Exemple (informal)

Sens Arquitectura:

- Ombres de les classes
- Organització del Codi
- Descontrolades entre elles
- Allà on hi ha un problema
- Difícil d'entendre
- Difícil de mantenir
- Difícil de debuggar

Conclusió

- Facilita pel programador
- Donar un exemple



3

Característiques Desitjables d'una Arquitectura

- L'Objectiu del disseny és tenir una arquitectura de qualitat. Per a fer-ho cal considerar com distribuïm les classes i operacions creades i quina responsabilitat té cada element de la nostra arquitectura.
- Hi ha dues propietats que són un bon indicador de una arquitectura correcta
 - La quantitat d'acoblament (que interessa sigui baix)
 - El nivell de Cohesió (que interessa que sigui alt)
- Existeixen múltiples principis que ens poden ajudar a fer una arquitectura bona
 - Principi obert-tancat
- No obstant, no hi ha una fórmula única que indiqui què fa que una arquitectura sigui bona i cada cas s'ha de considerar específicament.

4

Propietats i Principis

- Acoblament
- Cohesió
- Principi Obert-Tancat

5

Acoblament

- **Acoblament** d'una classe és una mesura del grau de connexió, coneixement i dependència d'aquesta classe respecte d'altres classes.
- Direm que una classe **A** està acoblada amb una classe **B**, si la classe **A** sap de la existència de **B**
- L'acoblament és inevitable però s'ha d'intentar reduir al mínim necessari perquè pot portar problemes...

6

Acoblament

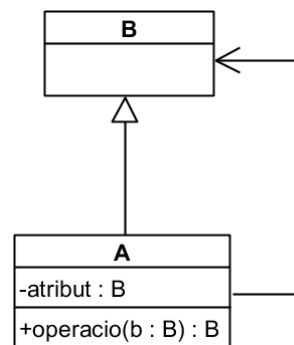
- Convé que l'acoblament sigui baix:
 - Si hi ha un acoblament de A a B, un canvi en B pot implicar canviar A
 - Quan més acoblament té una classe, més difícil resulta comprendre-la aïlladament
 - Quan més acoblament té una classe, és més difícil reutilitzar-la, perquè requereix la presència d'altres classes
- Excepcions
 - L'acoblament amb classes estables ben conegudes no acostuma a ser un problema (tipus de dades, classes biblioteques ofertes pel llenguatge de programació, ...)

7

Acoblament

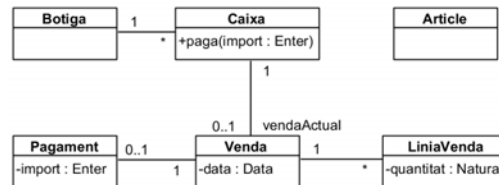
Direm que hi ha un acoblament entre la classe A i la classe B si:

- A té un atribut de tipus B
- A té una associació navegable amb B
- B és un paràmetre o el retorn d'una operació de A
- Una operació de A fa referència a un objecte de B
- A és una subclasse directa o indirecta de B
- ...



8

Acoblament - Exemple

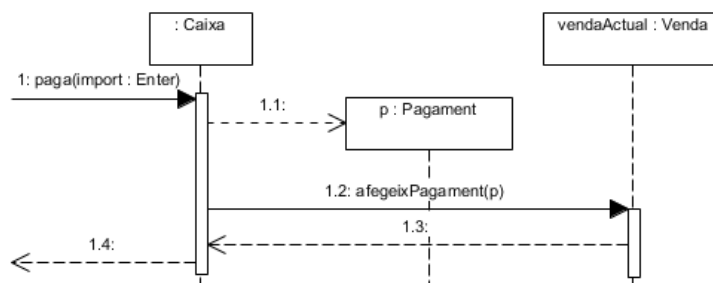


context Caixa::paga(import: Enter)
pre: existeix vendaActual
post: crea un pagament nou p amb p.import = import
 associa la vendaActual amb el pagament

9

Acoblament - Exemple

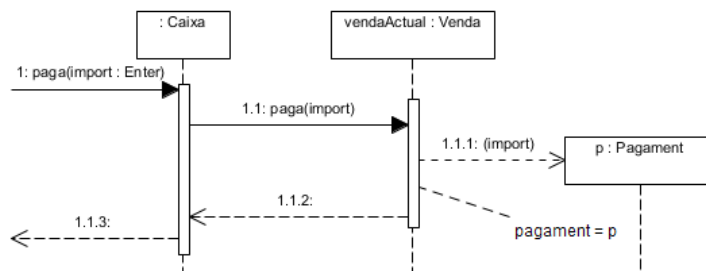
- Caixa crea un pagament i l'associa a Venda
- Introdueix un nou acoblament (dinàmic) entre Caixa i Pagament !



10

Acoblament - Exemple

- Caixa delega a Venda l'operació de creació del pagament
- No introdueix nous acoblaments dinàmics



11

Acoblament – Llei de Demèter

Una operació només hauria d'invocar operacions ("parlar") d'objectes accessibles des de *self* ("familiars"), que són:

- L'objecte que està executant l'operació (*self*)
- Un paràmetre rebut per l'operació
- Els valors dels atributs de l'objecte *self*
- Els objectes associats amb *self*
- Els objectes creats per la pròpia operació

Tots els altres objectes són "estranyes". Per això, la llei també es coneix com a "No parleu amb estranyes".

La llei de Demèter ajuda a mantenir l'acoblament baix



Cohesió

Cohesió d'una classe és una mesura del grau de relació i de concentració de les diverses responsabilitats (atributs, associacions i operacions)

Informalment direm que una classe està ben cohesionada si:

- Està clar què fa
- Fa una cosa única
- És responsable d'aquella cosa

13

Cohesió

- Convé que la cohesió sigui alta
- Una classe amb cohesió alta:
 - Té poques responsabilitats en una àrea funcional
 - Col·labora (delega) amb d'altres classes per a fer les tasques
 - Acostuma a tenir poques operacions. Aquestes operacions
 - Estan molt relacionades funcionalment
- Avantatges:
 - Fàcil comprensió
 - Fàcil reutilització i manteniment
- No existeix una mètrica quantitativa simple de la cohesió
 - Avaluació qualitativa

14

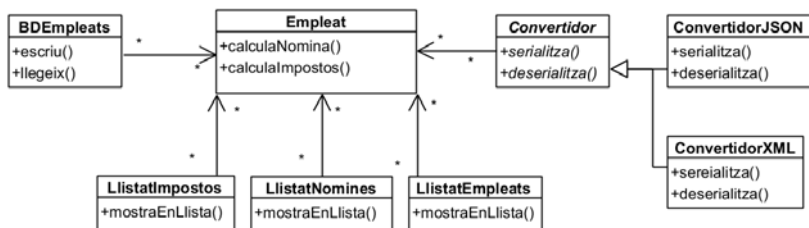
Cohesió - Exemple

Empleat
+calculaNomina() +calculaImpostos() +escriuADisc() +llegeixDeDisc() +creaXML() +creaJSON() +llegeixDeXML() +llegeixDeJSON() +mostraEnLlistatNomina() +mostraEnLlistatImpostos() +mostraEnLlistatEmpleats()

- Aquesta classe és un exemple de Cohesió baixa

15

Cohesió - Exemple



- Aquest sistema és un exemple de cohesió alta

16

El principi Obert-Tancat (OCP)

- Els mòduls (classes, funcions, etc.) haurien de ser:
 - *Oberts* per a l'extensió. El comportament del mòdul es pot estendre per tal de satisfer nous requisits.
 - *Tancats* per a la modificació. L'extensió no implica canvis en el codi del mòdul. No s'ha de tocar la versió executable del mòdul.
- El comportament dels mòduls que satisfan aquest principi es canvia afegint nou codi, i no pas canviant codi existent.
- L'ús correcte del polimorfisme afavoreix aquest principi

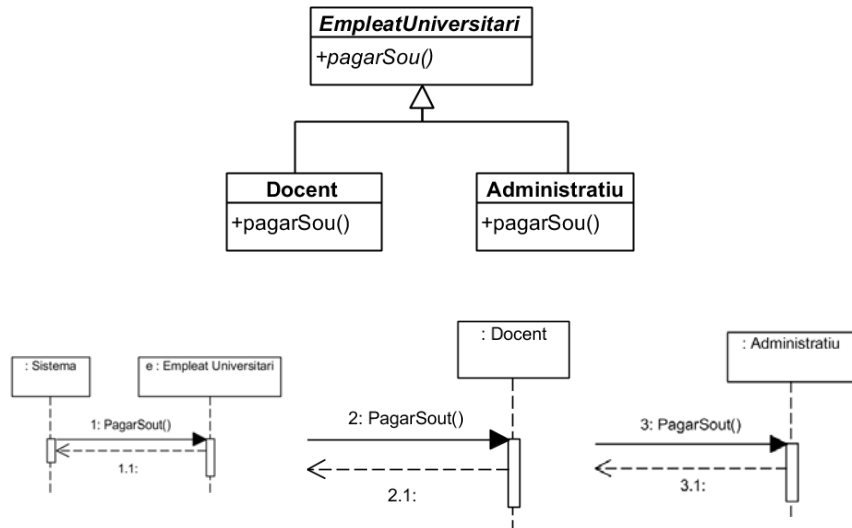
17

El principi Obert-Tancat (OCP)

- Totes les entitats software (classes, mòduls, funcions, etc.) haurien d'estar obertes per extensió, però tancades per modificació

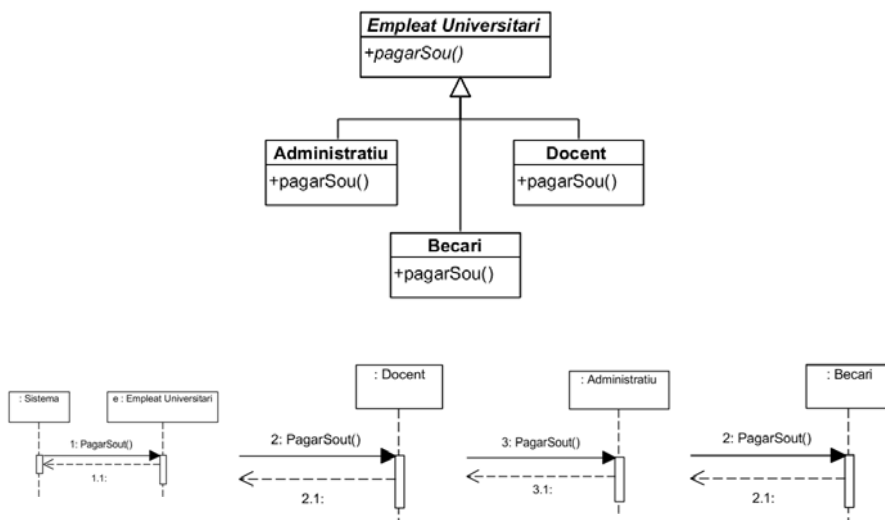
18

Principi Obert-Tancat - Exemple



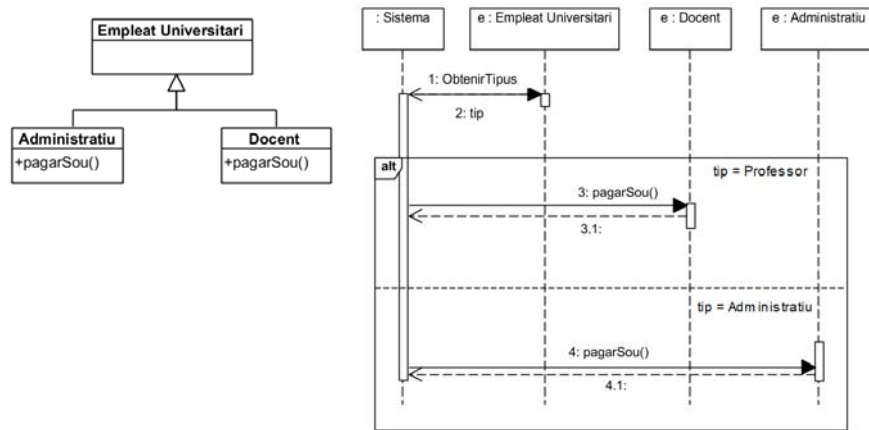
19

Principi Obert-Tancat - Exemple



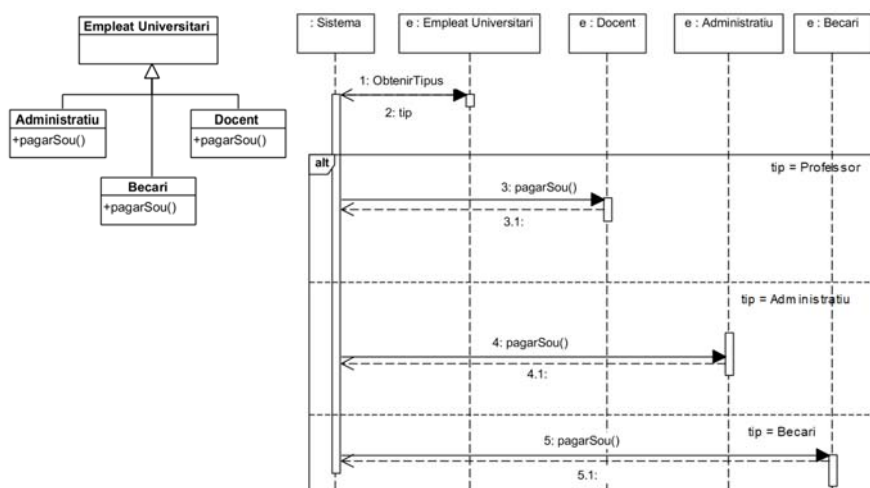
20

Principi Obert-Tancat – Exemple on no se satisfà



21

Principi Obert-Tancat - Exemple on no se satisfà



22

Arquitectura

- Arquitectura en “components”
- Arquitectura en Capes

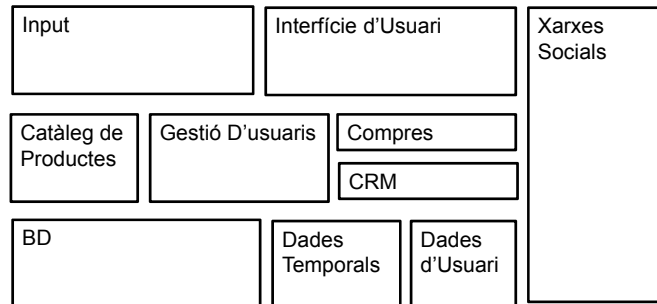
23

Arquitectura en “components”

- A nivell global, una arquitectura ha de mantenir els principis d'acoblament baix i cohesió alta no només entre classes, sinó també entre grups de classes.
- A arrel d'aquesta situació apareix la necessitat de dividir les architectures en “components” i establir una comunicació entre ells.
- Cada “component” ha de mantenir un nivell d'acoblament i cohesió amb els altres blocs i alhora un nivell d'acoblament i cohesió entre les diverses classes que el componen.
- Els “components” poden rebre mols noms:
 - Capes
 - SDKs
 - Lliberies
 - ...

24

Arquitectura en “components” - exemple



25

Arquitectura en Capes

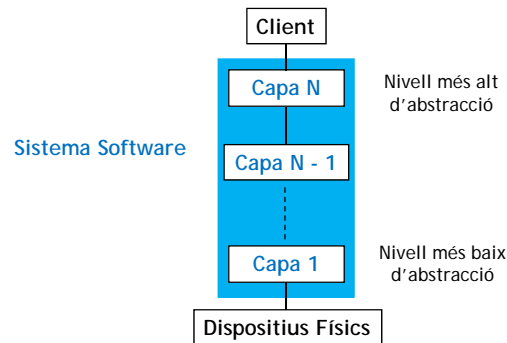
- Una de les formes més comunes i utilitzades per a dividir el codi en components és fer servir una Arquitectura en Capes...

26

Arquitectura en Capes

Context:

- Un sistema gran que requereix ser descompost en grups de tasques (components), tals que cada grup de tasques està a un nivell determinat d'abstracció.



27

Arquitectura en capes: beneficis i inconvenients

Beneficis:

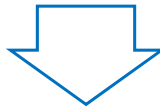
Canviable, Reusable, Portable, Provable

Inconvenients:

Eficiència

Feina innecessària o redundant

Dificultat en establir la granularitat i el nombre de capes



Arquitectura en capes relaxat

Una capa pot usar els serveis de qualsevol capa inferior

Tots o només part dels serveis de la capa (opacitat parcial)

Conseqüències:

Possible guany en flexibilitat i eficiència

Possible pèrdua en la canvialitat, reusabilitat

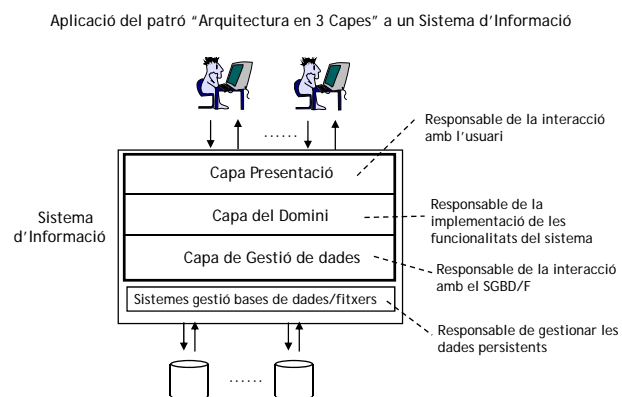
28

Arquitectura en 3 capes

- Una de les architectures en capes més utilitzada és la arquitectura en 3 capes.
 - Capa de Presentació: Responsable de la interacció amb l'usuari
 - Capa de Domini: Responsable de la implementació del sistema
 - Capa de gestió de Dades: Responsable de la interacció amb el SGBD/F

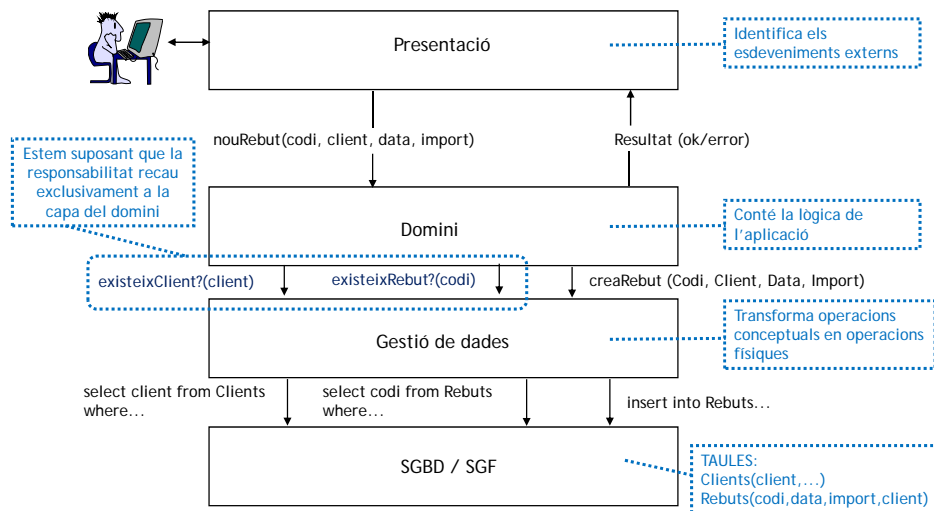
29

Arquitectura en capes: visió general



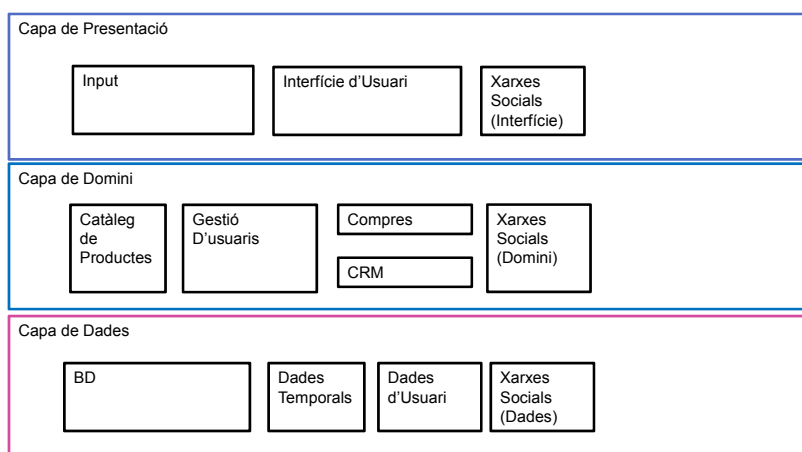
30

Exemple: comunicació entre capes



31

Arquitectura en “capes” - exemple



32

Bibliografia

- Larman, C. "*Applying UML and Patterns. An Introduction to Object-oriented Analysis and Design*", Prentice Hall, 2005, (3^a edició).
- Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. "*Pattern-oriented software architecture. A system of patterns*", John Wiley & Sons, 1996.
- Fowler, M. "*Patterns of Enterprise Application Architecture*", Addison-Wesley, 2002.
- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. "*Design Patterns*", Addison-Wesley, 1995
- Martin, R.C., "*Agile Software Development: Principles, Patterns and Practices*", Prentice Hall, 2003.