Arquitectura del Software

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona
FIB
Departament d'Enginyeria de Serveis i Sistemes d'Informació
UNIVERSITAT POLITÈCNICA DE CATALUNYA

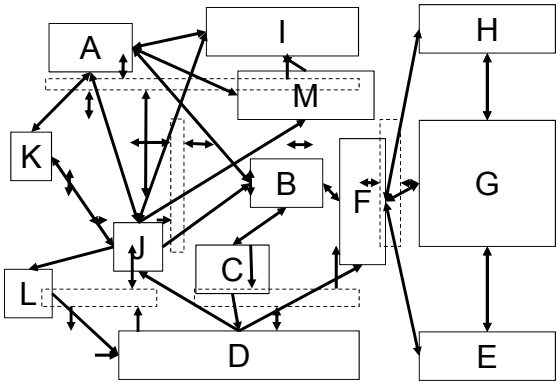
Arquitectura del Software

- 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)

Sense Arquitectura:

- Organització de les classes
- Organització del Codi
- Sabem on mirar quan hi ha un problema
- Sabem com extendre el sistema
- Difícil debuggar



Conclusió

- Facilitat pel programador

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.

Propietats i Principis

- Acoblament
- Cohesió
- Principi Obert-Tancat

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...

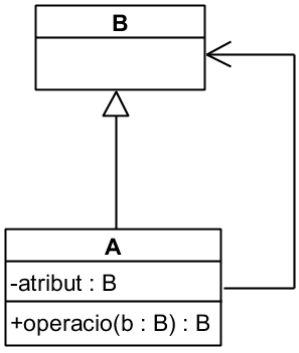
Acoblament

- 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, ...)

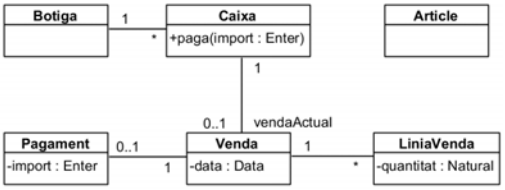
Acoblament

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
- ...



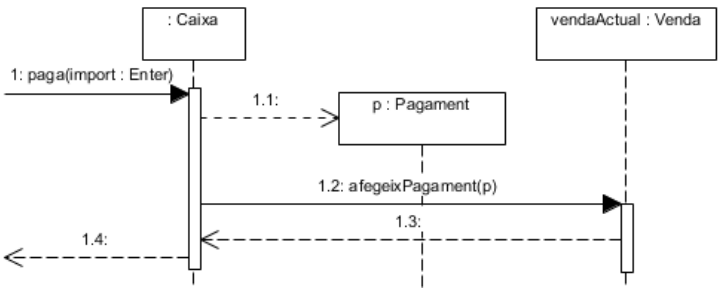
Acoblament - Exemple




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

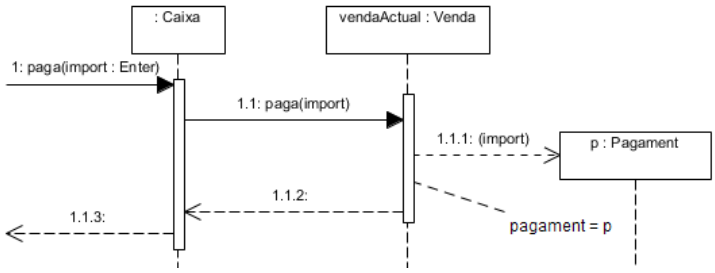

Acoblament - Exemple

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



Acoblament - Exemple

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



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 "estranys". Per això, la llei també es coneix com a "No parleu amb estranys".

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)

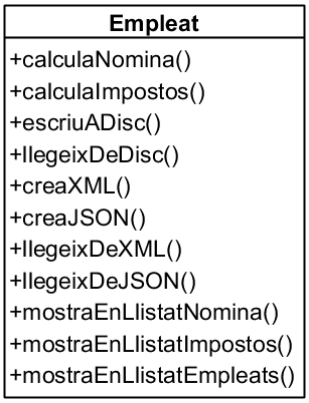
Informalment direm que una classe està ben cohesionada si:

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

Cohesió

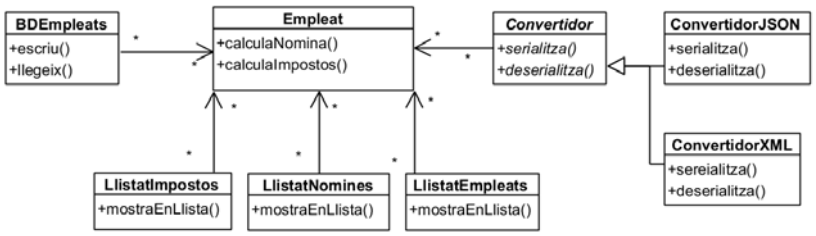
- Convé que la cohesió sigui alta
- Una classe amb cohesió alta:
  - Té poques responsabilitats en una àrea funcional
  - Col·labora (delega) amb d'altres classes per a fer les tasques
  - Acostuma a tenir poques operacions. Aquestes operacions
  - Estan molt relacionades funcionalment
- Avantatges: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

Cohesió - Exemple



- Aquesta classe és un exemple de Cohesió baixa

Cohesió - Exemple



- Aquest sistema és un exemple de cohesió alta

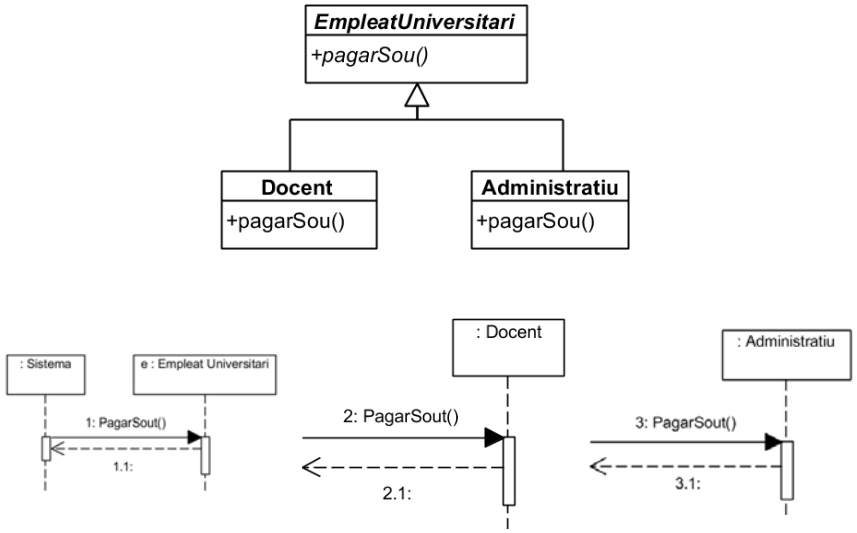
El principi Obert-Tancat (OCP)

- 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

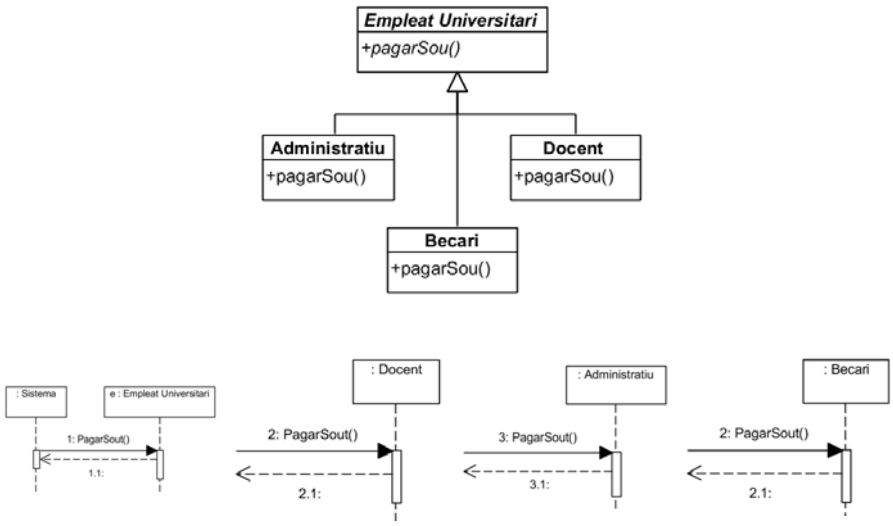
El principi Obert-Tancat (OCP)

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

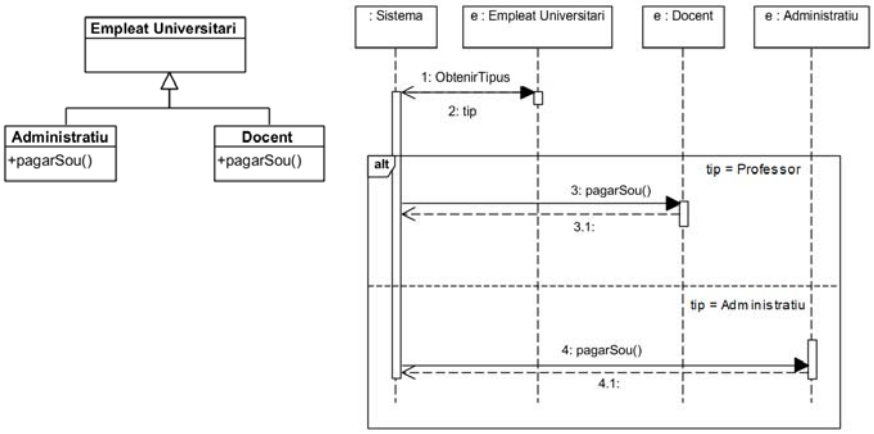
Principi Obert-Tancat - Exemple



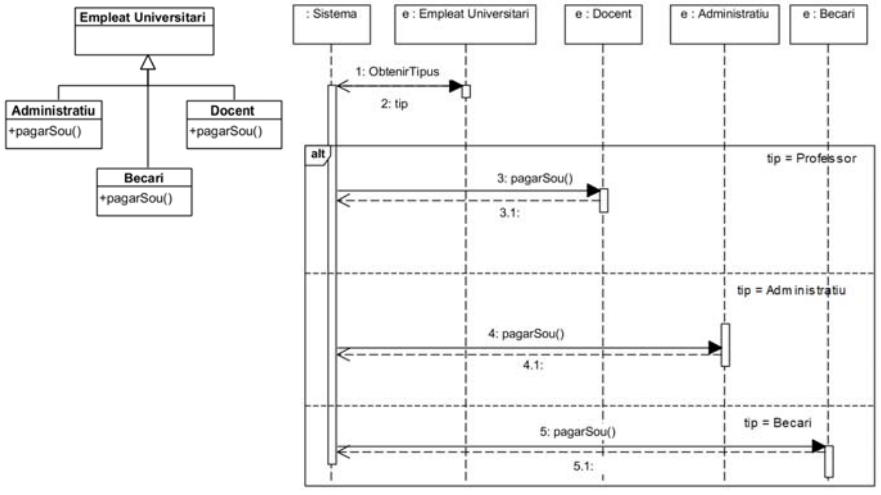
Principi Obert-Tancat - Exemple



Principi Obert-Tancat – Exemple on no se satisfà



Principi Obert-Tancat - Exemple on no se satisfà



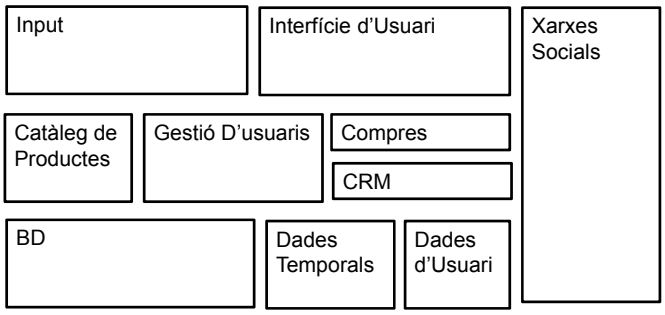
Arquitectura

- Arquitectura en “components”
- Arquitectura en Capes

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 arquitectures 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 composen.
- Els “components” poden rebre molts noms:
  - Capes
  - SDKs
  - Llibreries
  - …

Arquitectura en “components” - exemple



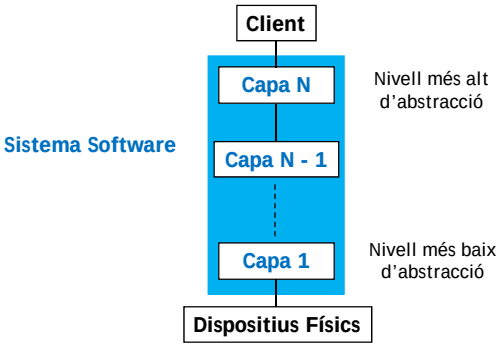
Arquitectura en Capes

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

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.



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:

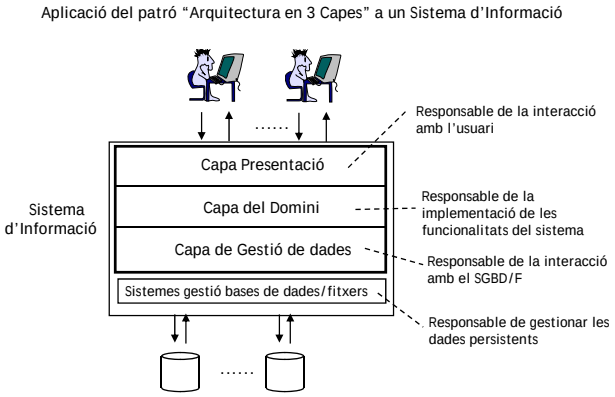
Possible guany en flexibilitat i eficiència

Possible pèrdua en la canviabilitat, reusabilitat

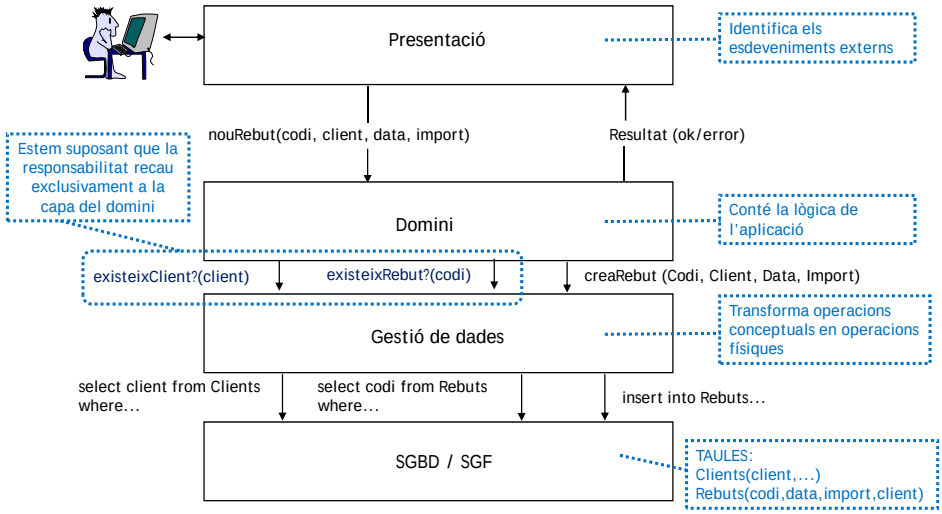
Arquitectura en 3 capes

- Una de les arquitectures 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

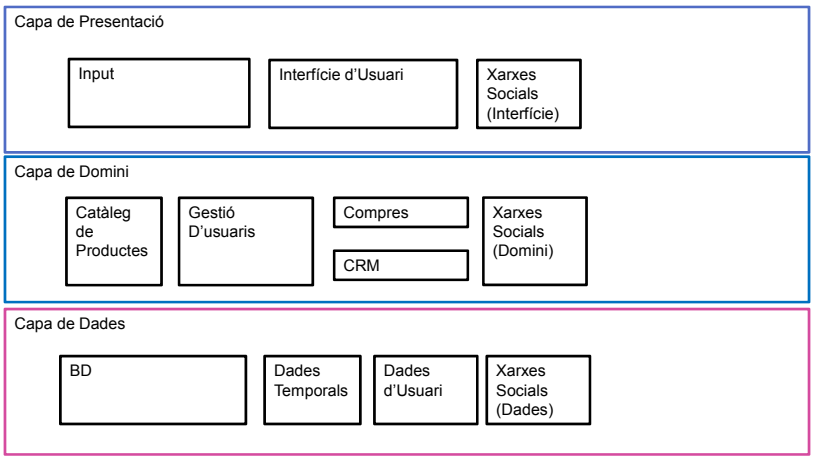
Arquitectura en capes: visió general



Exemple: comunicació entre capes



Arquitectura en "capes" - exemple



Bibliografia

- Larman, C. "*Applying UML and Patterns. An Introduction to Object-oriented Analysis and Design*", Prentice Hall, 2005, (3ª 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.