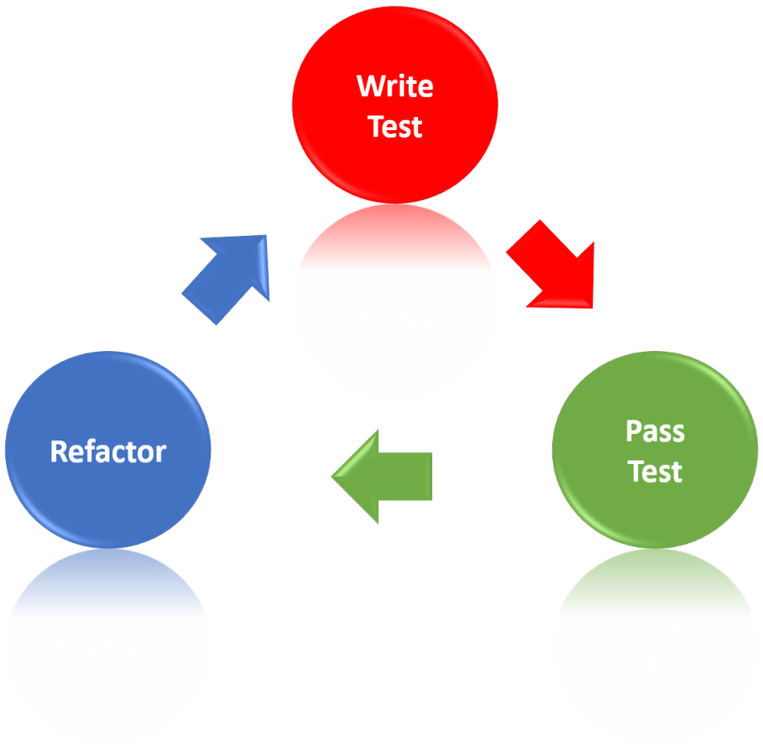


Tema 3: Disseny

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2020/2021

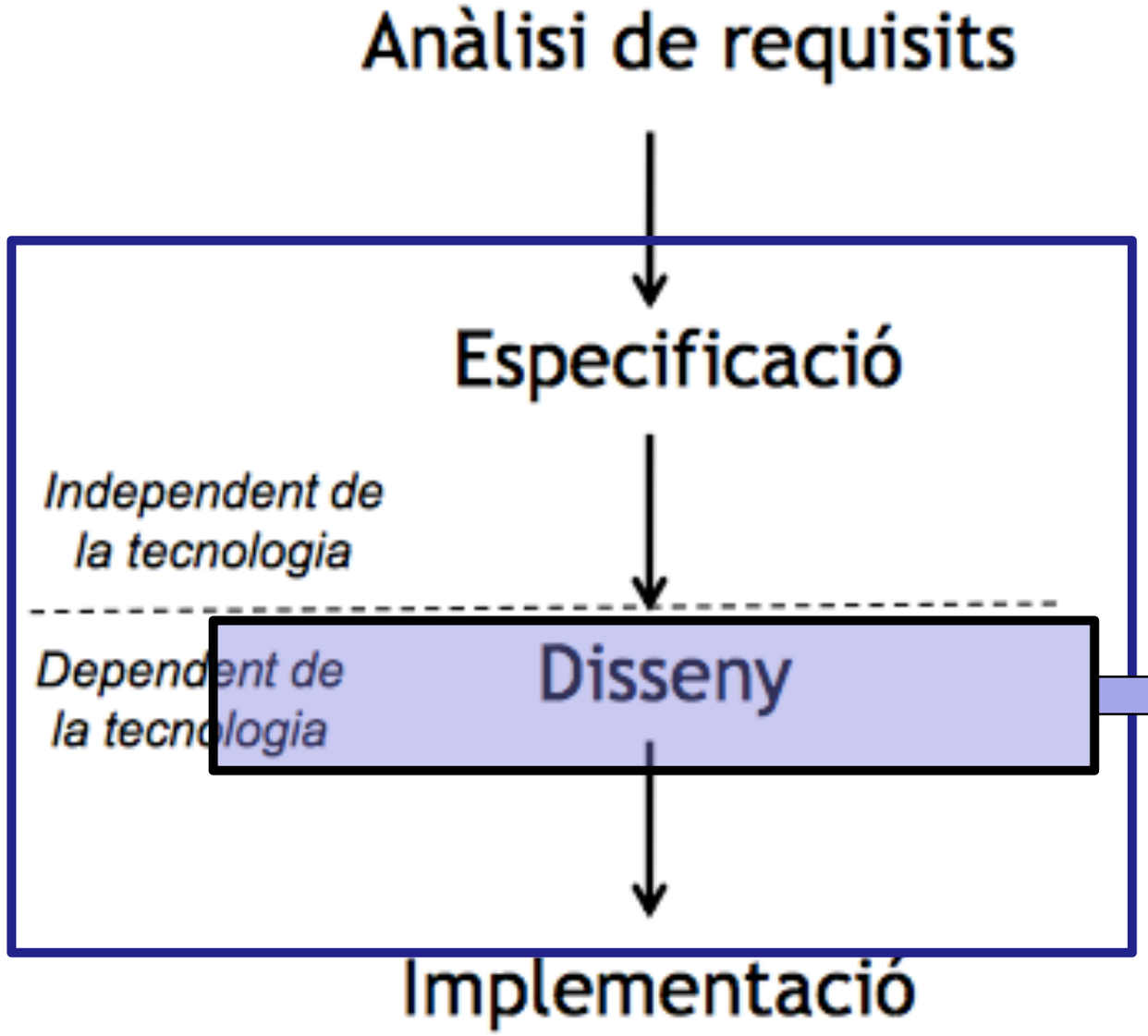
UNIVERSITAT DE BARCELONA

Temari

1. Introducció al procés de desenvolupament del software
2. Anàlisi de requisits i especificació
3. Disseny
   - 3.1 **Introducció**
   - 3.2 Patrons arquitectònics
   - 3.3 Criteris de Disseny: G.R.A.S.P.
   - 3.4 Principis de Disseny: S.O.L.I.D.
   - 3.5 Patrons de disseny
4. Del disseny a la implementació
5. Ús de frameworks de testing

3.1. Introducció

Procés sistemàtic (Tema 3):

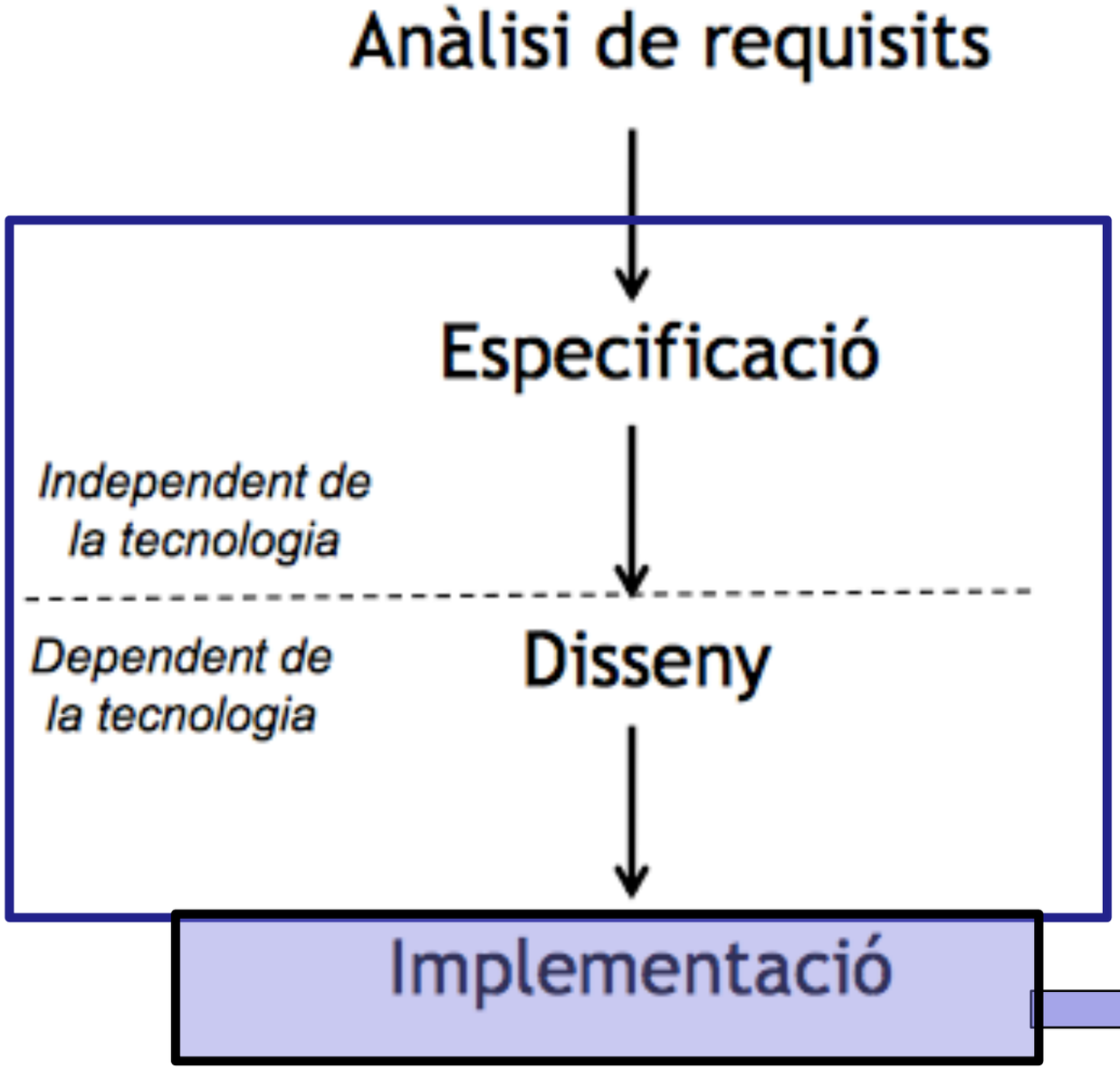


Com ho fa el sistema?

- **Tests d'acceptació**
- **Descripció** de l'**arquitectura** dels subsistemes i components del sistema software
- **Diagrama de classes**
- **Diagrames d'interacció**

3.1. Introducció

Procés sistemàtic (Laboratori):



Generació de Codi:

- **Tests unitaris**
- **Codi en llenguatge concret de les classes**

3.1. Introducció

Per a què es vol un **bon** disseny de software?

- Per manegar de forma **fiable** la **complexitat** del problema
- Per a **desenvolupar ràpid** i lliurar-lo a temps
- Per poder incloure **canvis** fàcilment

- Preservar els principis de disseny
- Adaptació de solucions genèriques a problemes coneguts de disseny (**patrons**)

3.1. Introducció

Els dissenyadors experts no resolen els problemes des de l'inici, reutilitzen solucions que han usat en el passat

- A més a més,

- El software canvia
- Per anticipar-se als canvis en els requisits s'ha de dissenyar pensant en quins aspectes poden canviar

- Els patrons de disseny estan orientats al canvi

3.1. Introducció

"*Each **pattern** (patró) describes a problem which occurs over and over again in our environment, and then describes the **core of the solution** to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice*"

Christopher Alexander, arquitecte (1977)

Nom del patró

Problema

- Descripció del problema a resoldre
- Enumeració de les forces a equilibrar

Solució

- ***Aspecte estàtic***: impacte en el diagrama de classes del disseny
- ***Aspecte dinàmic***: establiment del comportament de les noves operacions

Conseqüències: Avantatges i desavantatges

Com es descriu un patró

- **Nom del patró**: com es diu el patró? És una forma general de descriure un problema de disseny, les seves solucions i les conseqüències en una o dues paraules (Comunicació/Vocabulari tècnic)

- **Descripció del problema:** a resoldre (o context) Una descripció de quan (en quins casos) aplicar el patró i quan no s'ha d'aplicar. Ha d'explicar amb més detall el problema i el seu context

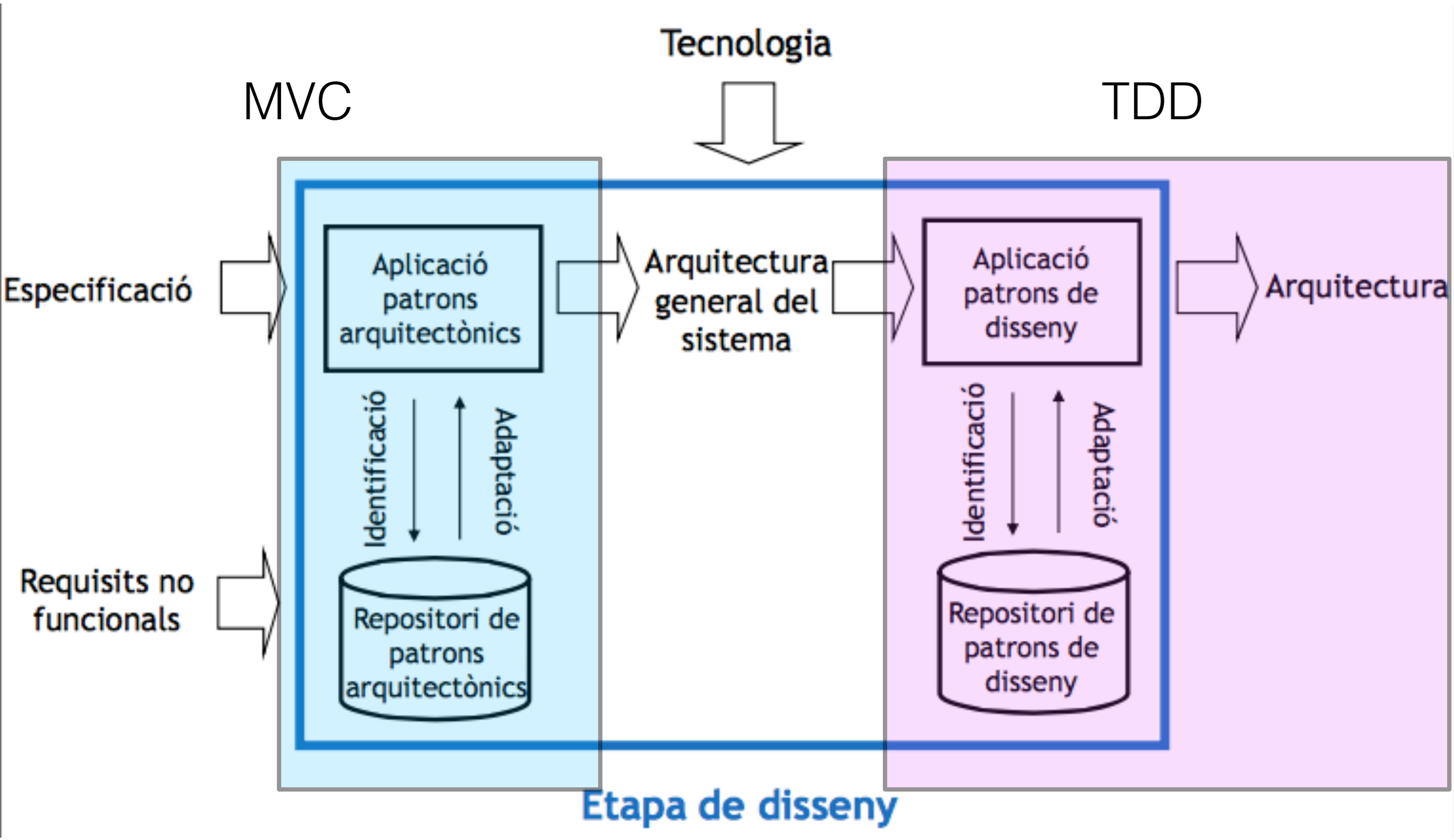
- **Solució:** Descriu els elements de disseny que constitueixen la solució (no és un disseny o implementació concreta per un cas particular, és com una plantilla que es pot aplicar a molts casos particulars)

- **Conseqüències:** Són els resultats d'aplicar el patró. Descriu avantatges i desavantatges

3.1. Introducció

- **Patrons d'arquitectura**: usats en el disseny a gran escala i de gra guixut.
  - *Per exemple:* patró de **Capes**
- **Patrons de disseny**: utilitzats en el disseny d'objectes i frameworks de petita i mitjana escala. (micro-arquitectura)
  - *Per exemple*, patró **Façana** (*Facade*) per connectar les capes o el patró **Estratègia** per permetre algorismes connectables
- **Patrons d'estils**: solucions de baix nivell orientades a la implementació o al llenguatge.
  - *Per exemple*, patró **Singleton** per fer una única instància d'una classe.

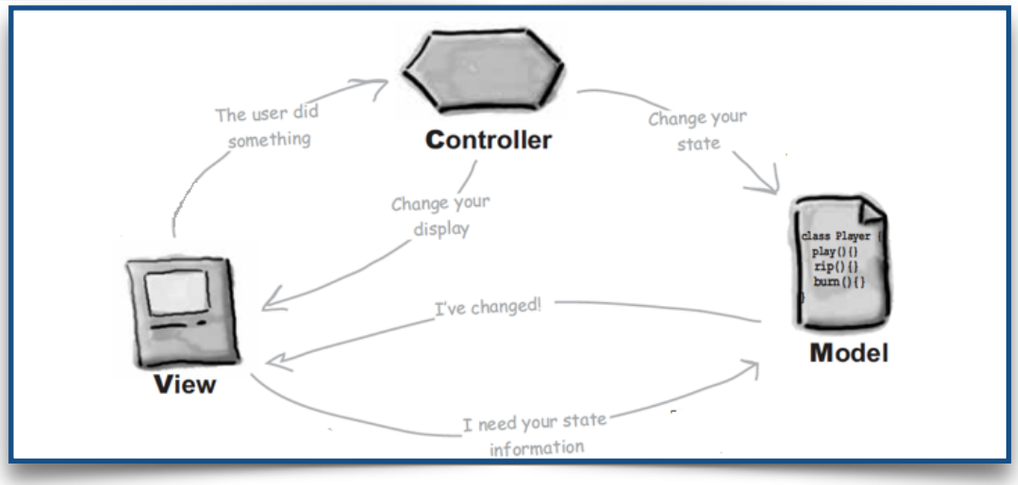
3.1. Introducció



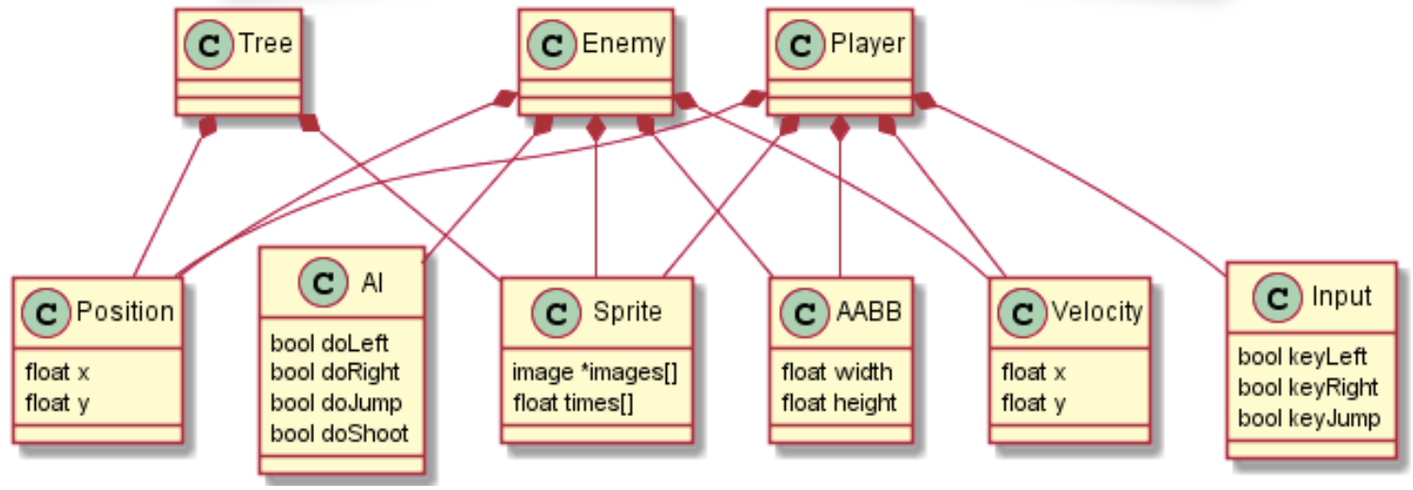
3.1. Introducció

Patrons arquitectònics
(entre d'altres):

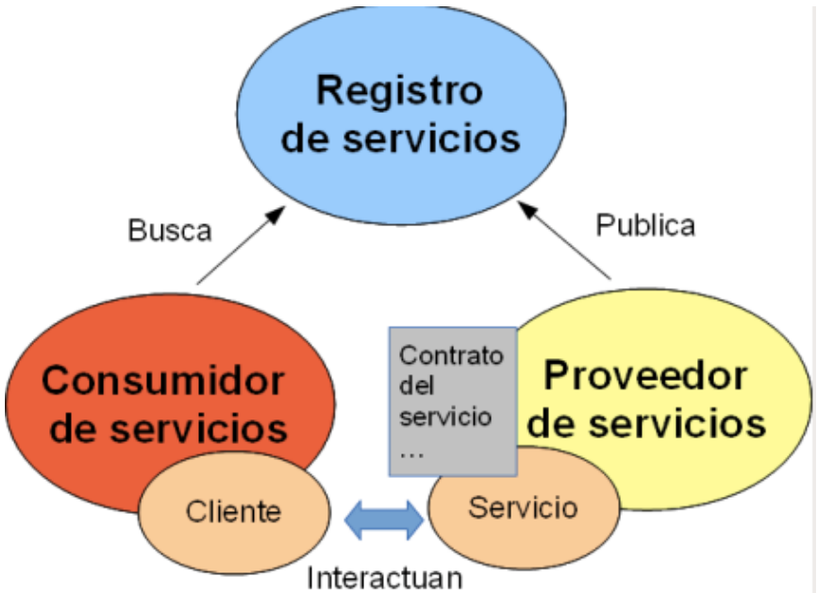
- Model-Vista-Controlador

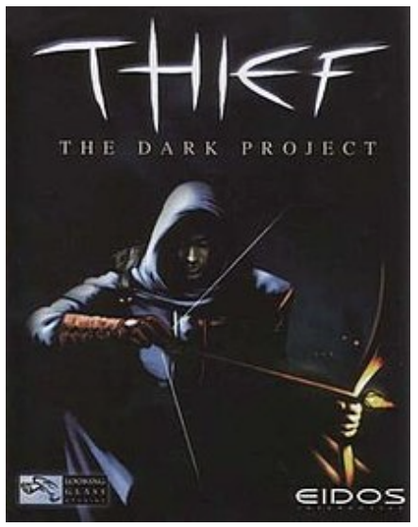


- Entity-Component System



- Service Oriented Architecture (SOA)



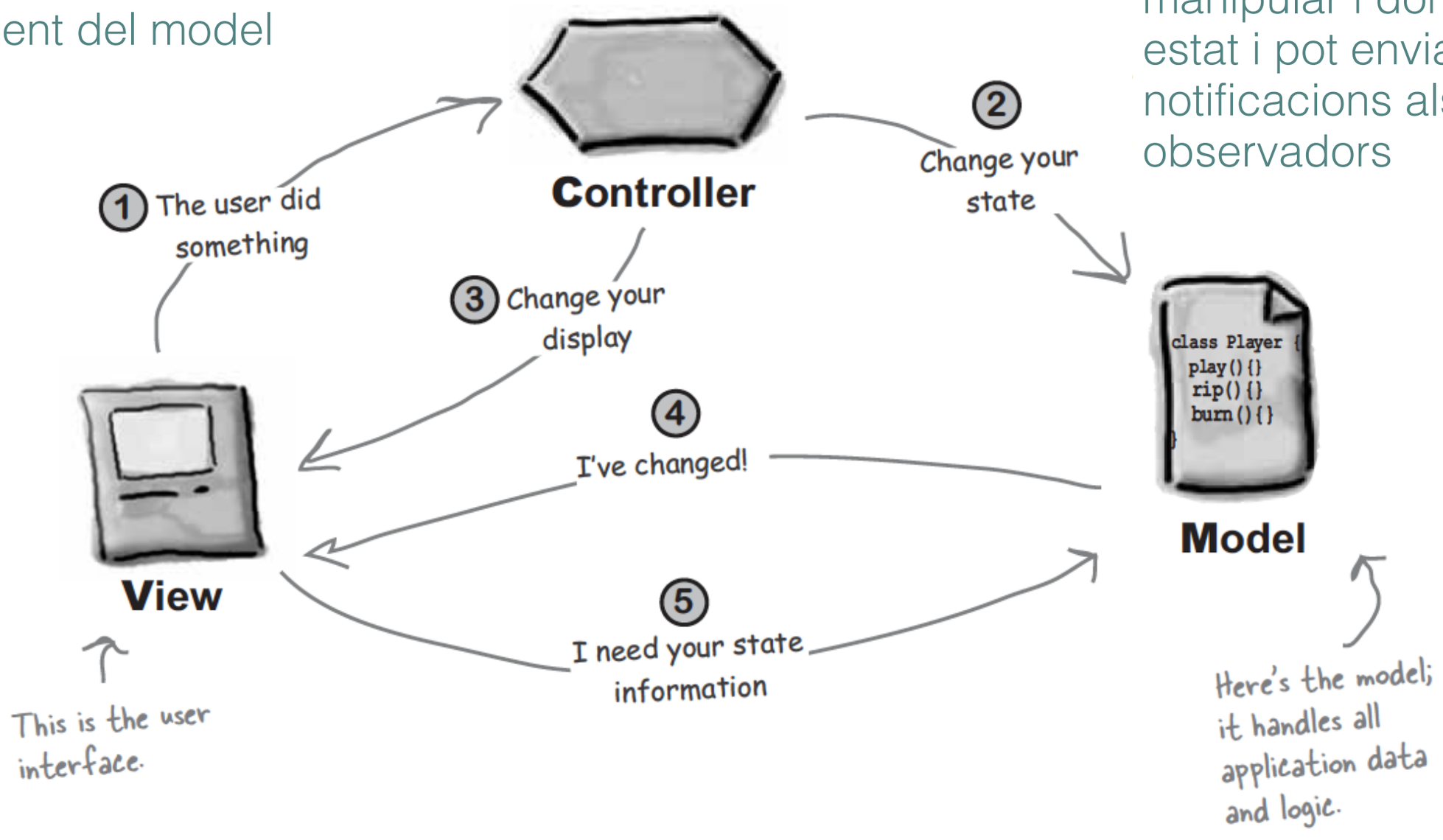


3.1. Introducció

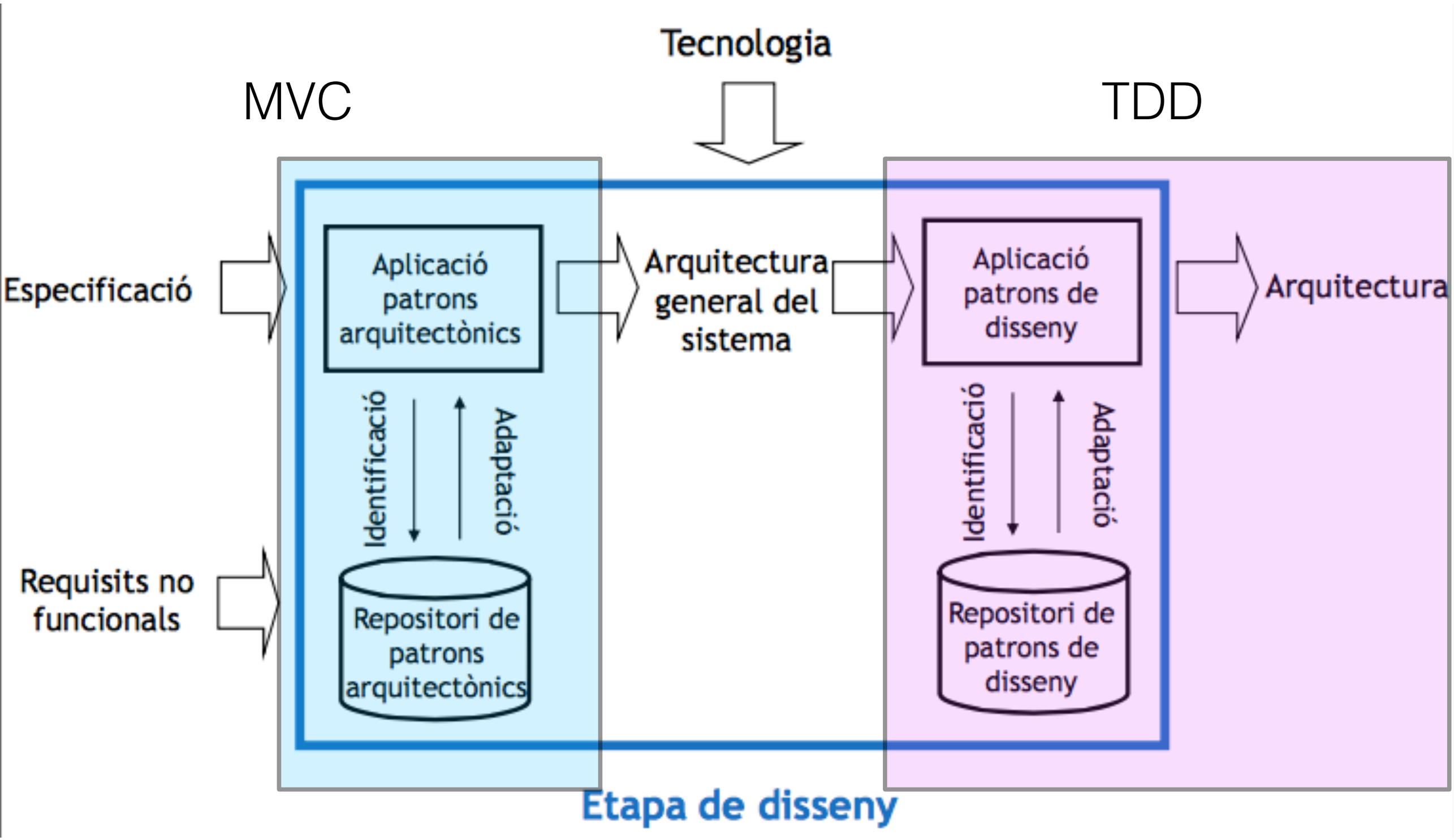
VISTA:
Dóna la presentació del model. La vista normalment mostra l'estat de les dades i el seu valor directament del model

CONTROLADOR:
Agafa l'entrada de l'usuari i li dóna el què significa al model

MODEL:
El model guarda totes les dates, l'estat i la lògica de l'aplicació. Dóna una interfície per manipular i donar el seu estat i pot enviar notificacions als observadors



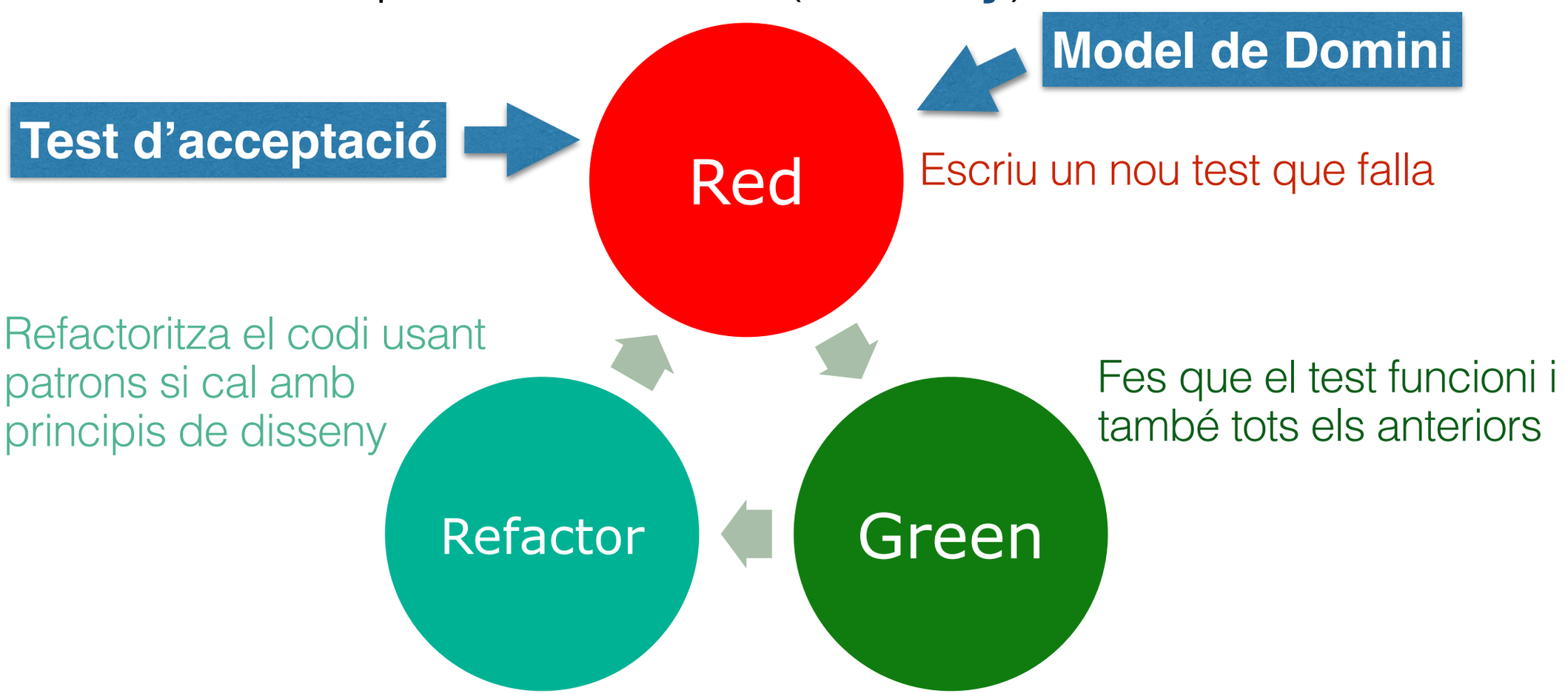
3.1. Introducció



3.1. Introducció

TDD (Test Driven Development): Basat en **dues** senzilles regles:

1. Escriu el nou codi només si el test automàtic ha fallat (**Disseny** i **Implementació**)
2. Elimina la duplicació de codi. (**Disseny**)

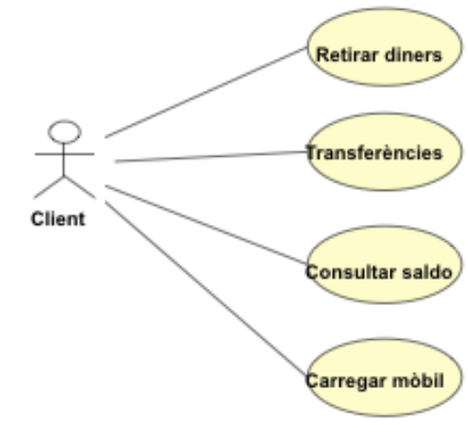


3.1. Introducció

Escenario simple de *ProcesarVenta* para el pago en efectivo

1. El Cliente llega al terminal PDV.
2. El Cajero inicia una nueva venta.
3. El Cajero inserta el identificador del artículo.
4. El Sistema registra la línea de venta y presenta la descripción del artículo, precio y suma parcial.
El Cajero repite los pasos 3 y 4 hasta que se indique.
5. El Sistema muestra el total con los impuestos calculados.
6. El Cajero le dice al Cliente el total, y pide que le pague.
7. El Cliente paga y el Sistema gestiona el pago.
...

Casos d'ús

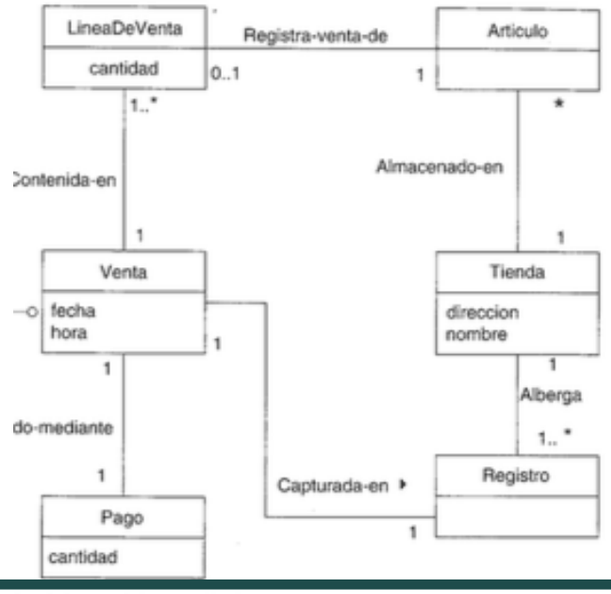


DCU

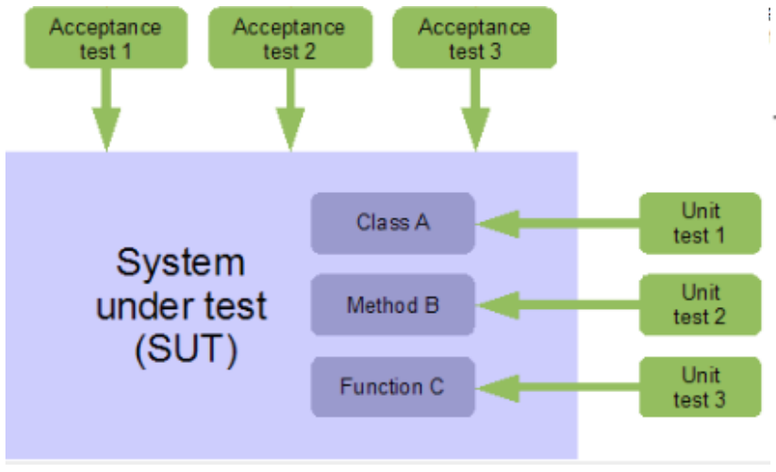
Com a [rol d'usuari]
vull [objectiu]
per què així [raó]

En cas que [context]
quan [event]
el sistema [resultat]

User Stories+Criteris d'acceptació



Model de Domini



Tests d'acceptacio + Tests unitaris

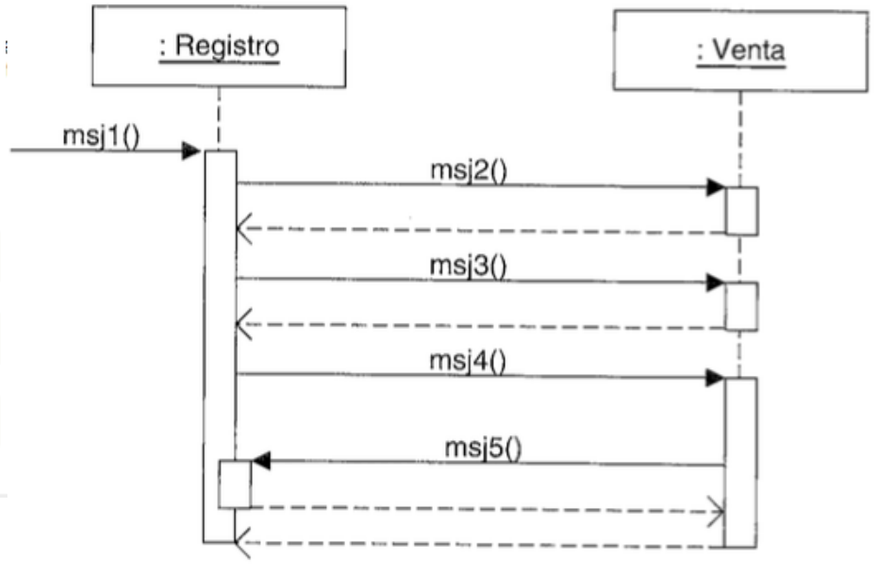


Diagrama de Seqüència (DS)

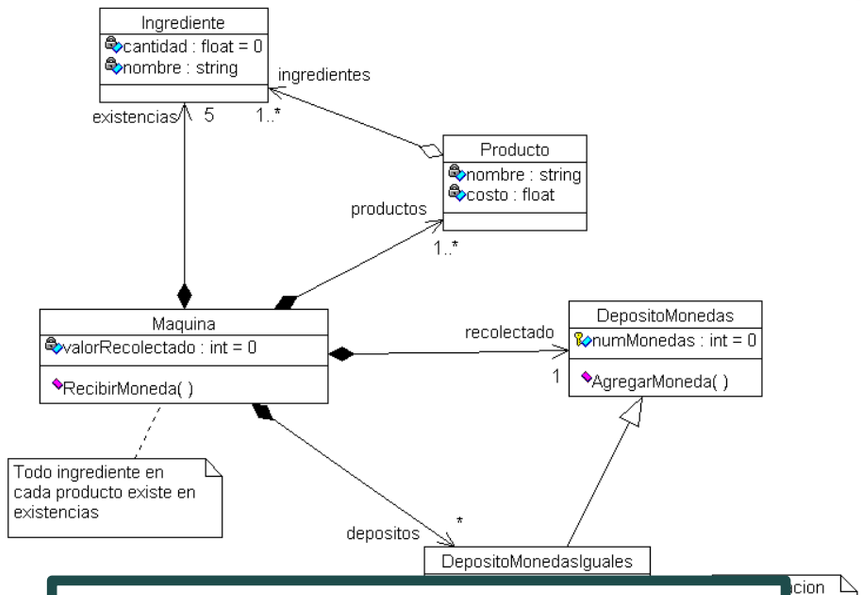


Diagrama de Classes de Disseny (DCD)

3.1. Introducció

- La creació del **model de disseny** és la construcció d'una solució basada en el paradigma orientat a objectes.
  - **Esquema de classes de les dades: (model estàtic)** Construcció els ***diagrames de classes*** que resumeixen la definició de les classes de software que s'estan implementant.
  - **Esquema de comportament: (model dinàmic)** Construcció de ***diagrames d'interacció*** que representen com els objectes col·laboren per satisfer els requisits. (***Diagrames de Seqüència***)

- Són eines de disseny, que ajudaran a avaluar el disseny i *aprendre a dissenyar* ***amb patrons***

3.1. Introducció

Un **diagrama d'interacció** consisteix en un conjunt d'objectes i les seves relacions, incloent-hi els missatges que es poden enviar entre ells

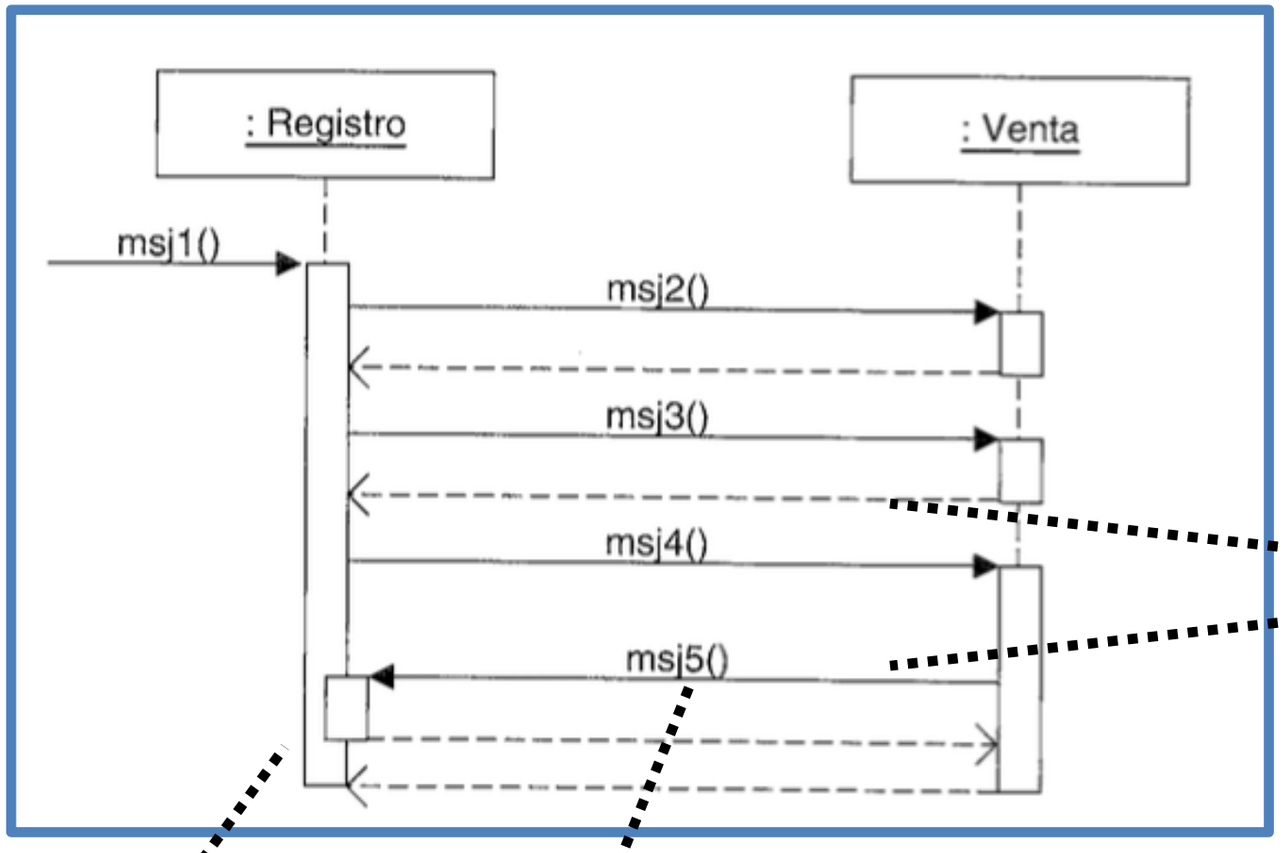
[modelar el **comportament** d'un sistema]

Aquests diagrames es desenvolupa en paral·lel amb diagrama de classes

1. **Diagrames de seqüència**: destaquen l'ordre temporal dels missatges
2. **Diagrames de col·laboració**: destaquen l'organització estructural dels objectes

Ambdós diagrames (seqüència i col·laboració) són semànticament equivalents. Es pot passar d'un a l'altre sense pèrdua d'informació

Tipus de diagrames d'interacció

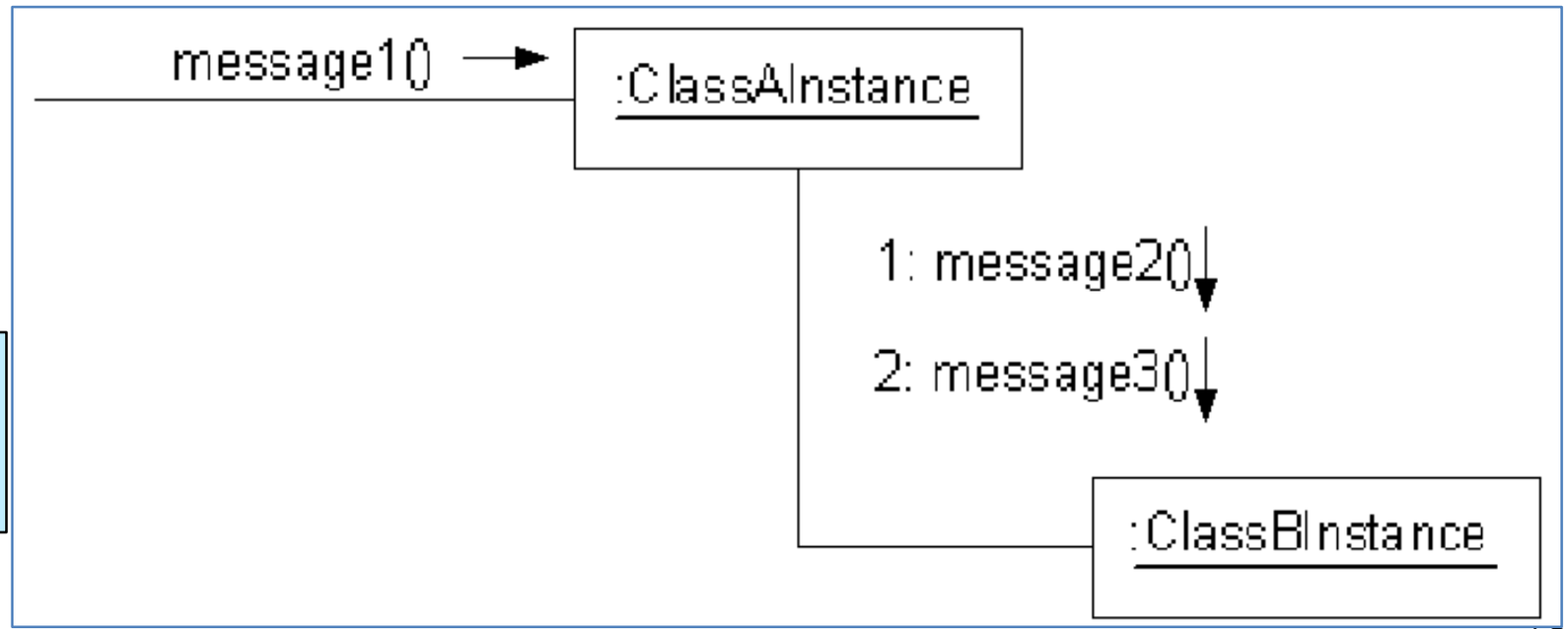


(a) Diagrama de seqüència

Retorns

Bloc activació

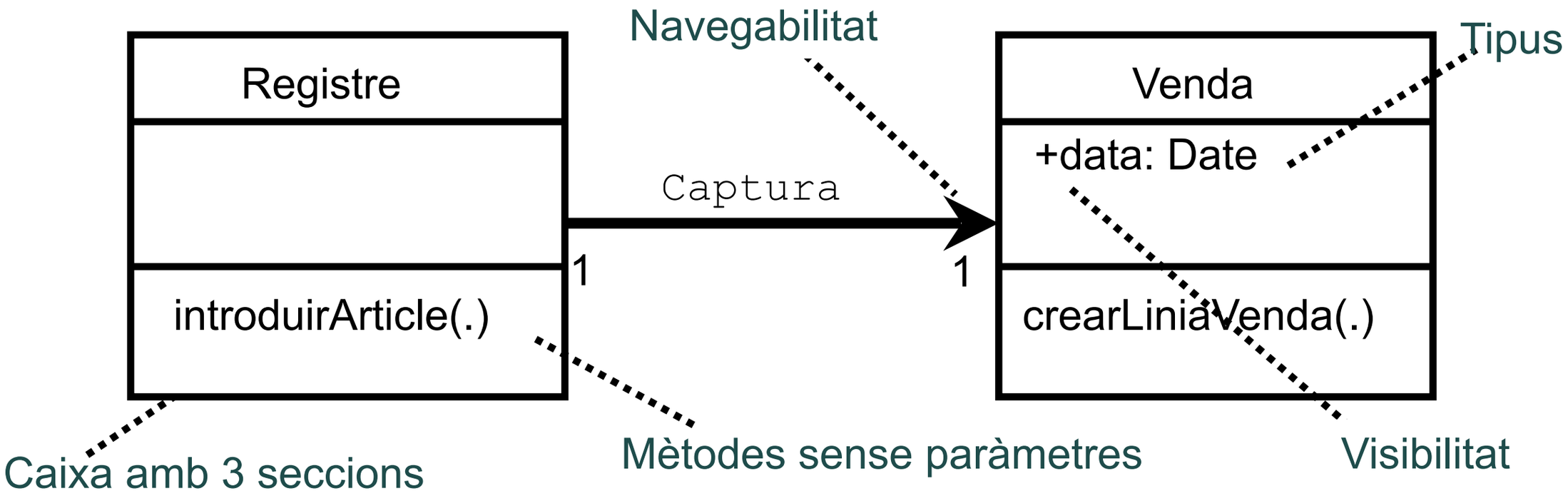
Missatge



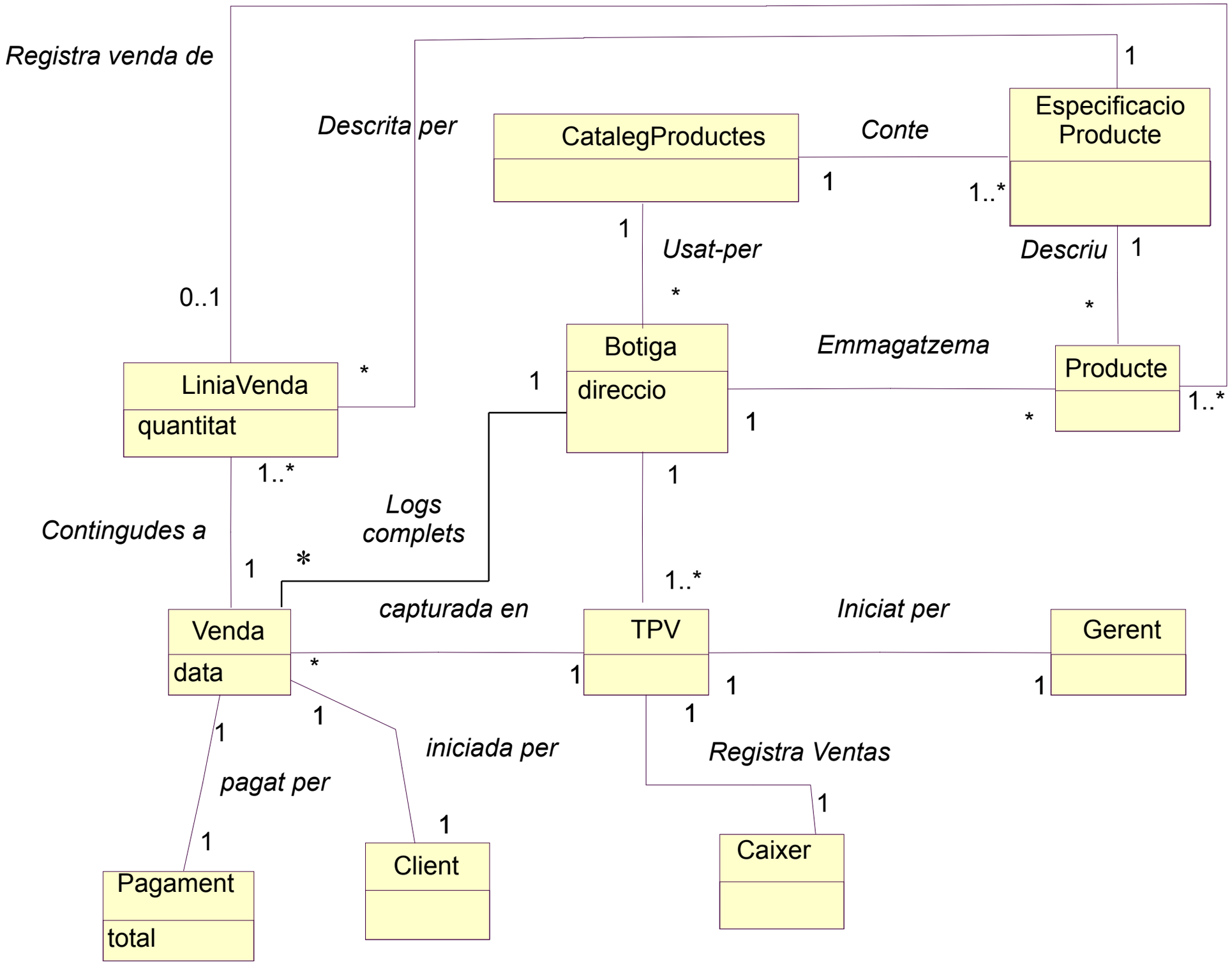
(b) Diagrama de col·laboració

3.1. Introducció

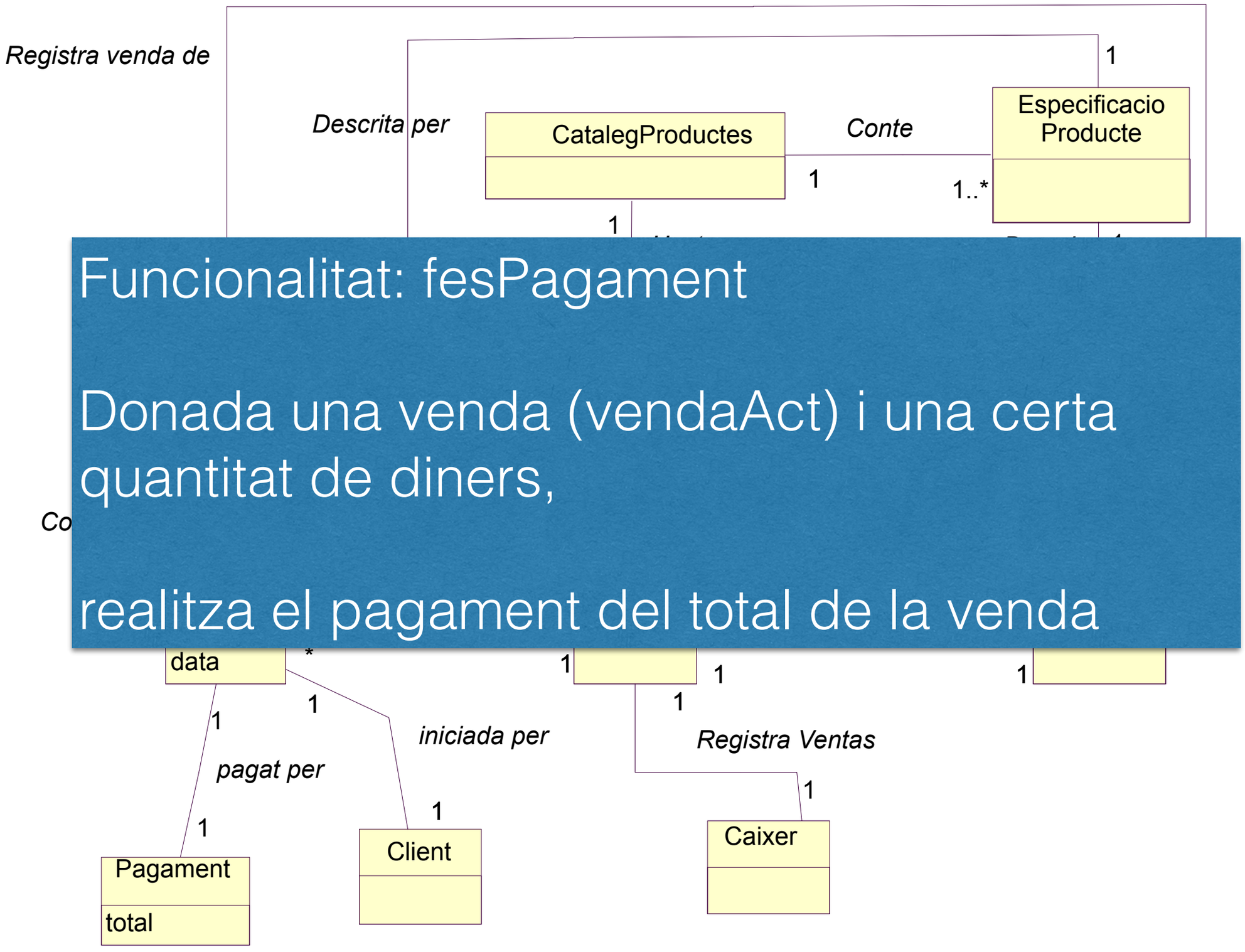
Un **Diagrama de Classes de Disseny** (DCD) il·lustra les especificacions per classes software i interfícies en una aplicació



Exemple model domini TPV



Exemple model domini TPV

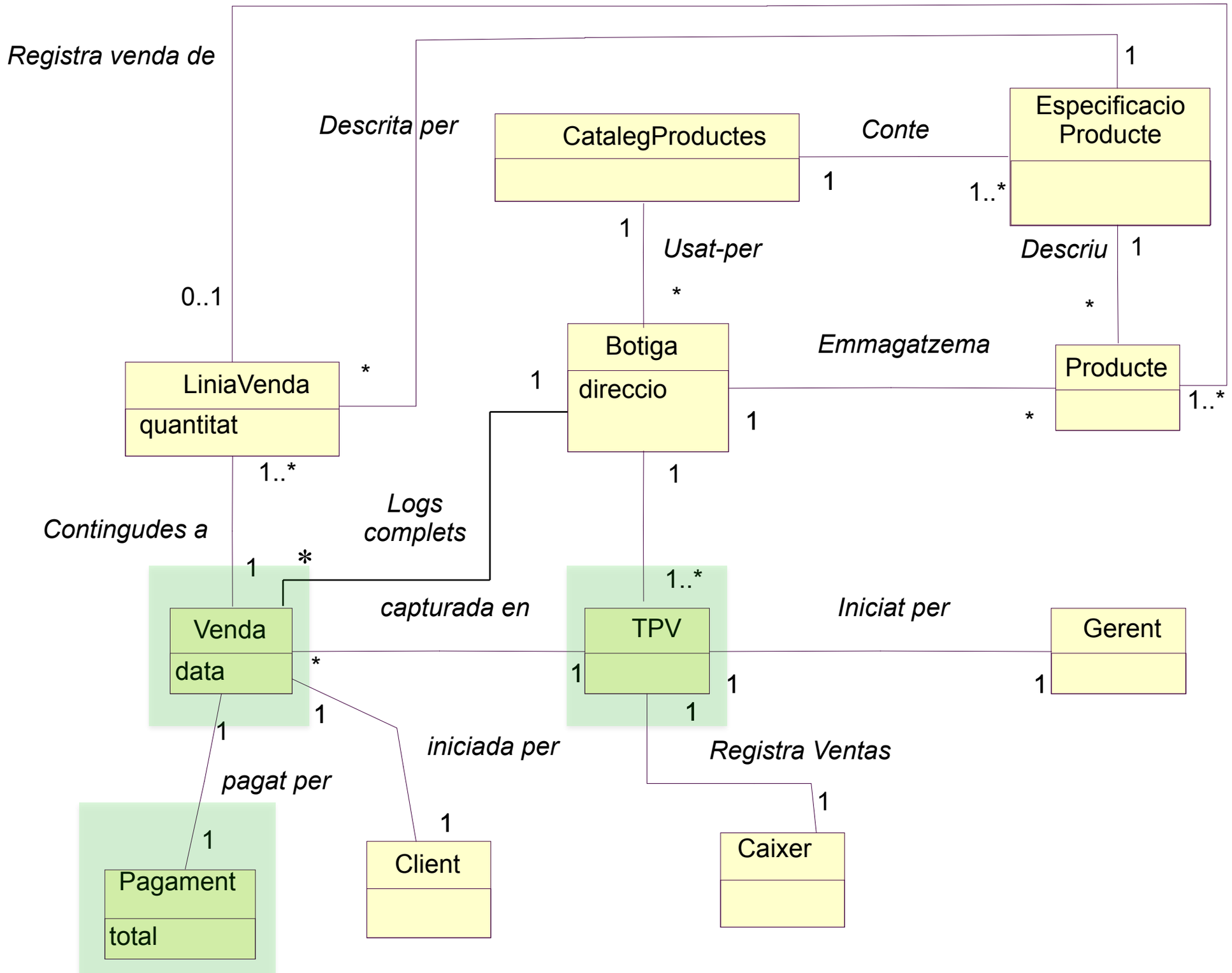


Funcionalitat: fesPagament

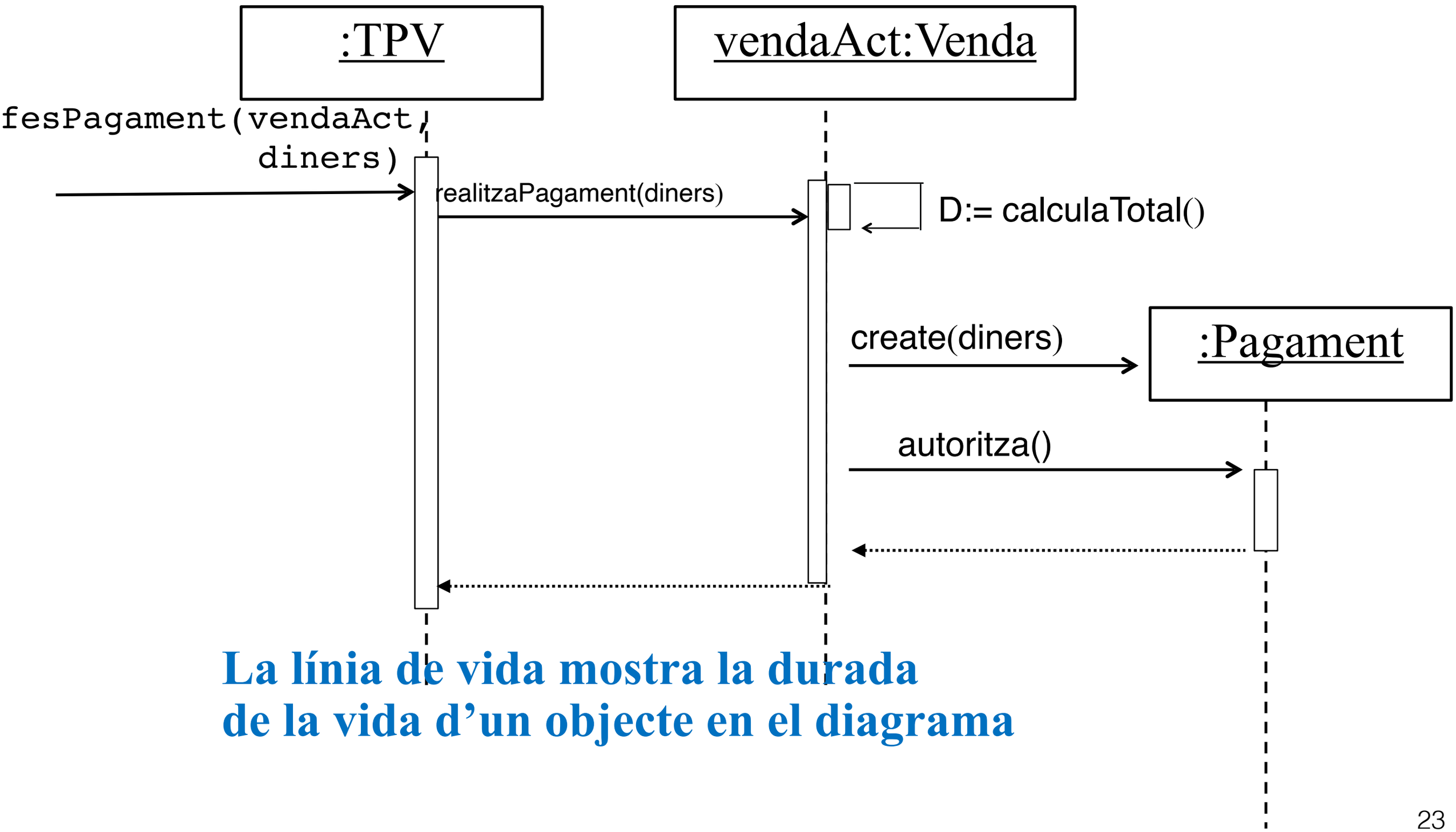
Donada una venda (vendaAct) i una certa quantitat de diners,

realitza el pagament del total de la venda

Exemple model domini TPV



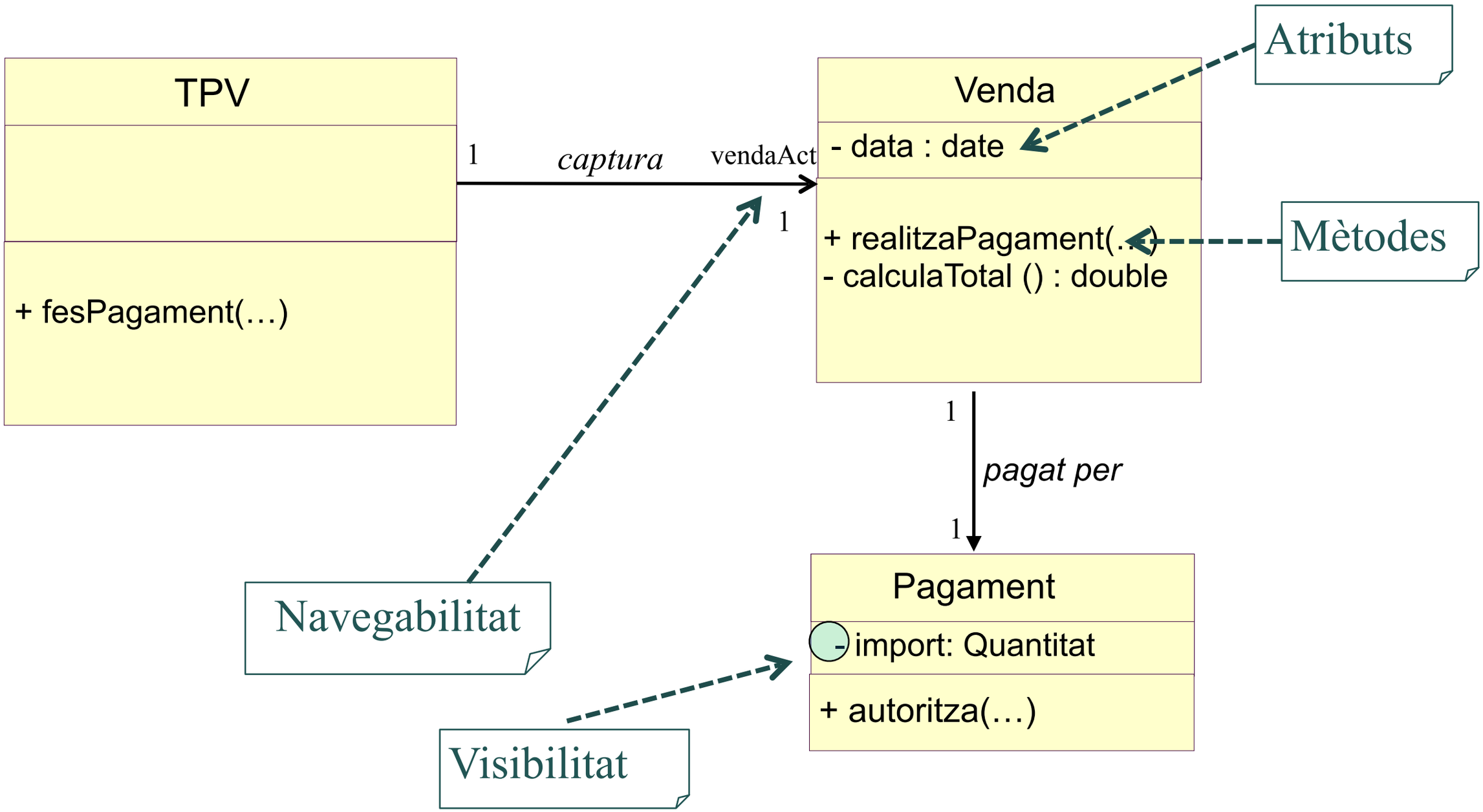
Criteri d'acceptació: fesPagament



La línia de vida mostra la durada de la vida d'un objecte en el diagrama

Diagrama de classes

Classes, Associacions, navegabilitat i visibilitat en una part del Terminal Punt de Venda



3.1. Introducció

Aspectes que denoten un disseny “dolent” (*code smell*)

- **Rigidesa**: l’impacte d’un canvi en el software és impredecible (cada canvi produeix una cascada de canvis en moltes altres classes o el codi és tan complicat que costa entendre’l)
- **Inmobilitat**: No es pot reutilitzar codi o parts de codi
- **Fragilitat**: A cada canvi, el software es trenca en llocs on no hi han relacions conceptuals
- **Viscositat**: Impossibilitat de canviar el codi sense canviar el disseny. Provoca que fer un pedaç addicional al codi és més fàcil que no pas canviar tot el disseny

→ Dependències entre classes

3.1. Introducció

Característiques que permeten avaluar si un disseny és "bo":

- **Acoblament:**

  mesura del grau de connexió, coneixement i dependència d'una classe respecte d'altres classes.

Acoblament BAIX

Quan més acoblament té una classe:

- més difícil resulta comprendre-la aïlladament.
- més difícil de reutilitzar-la, perquè requereix la presència de les altres classes.

- **Cohesió:**

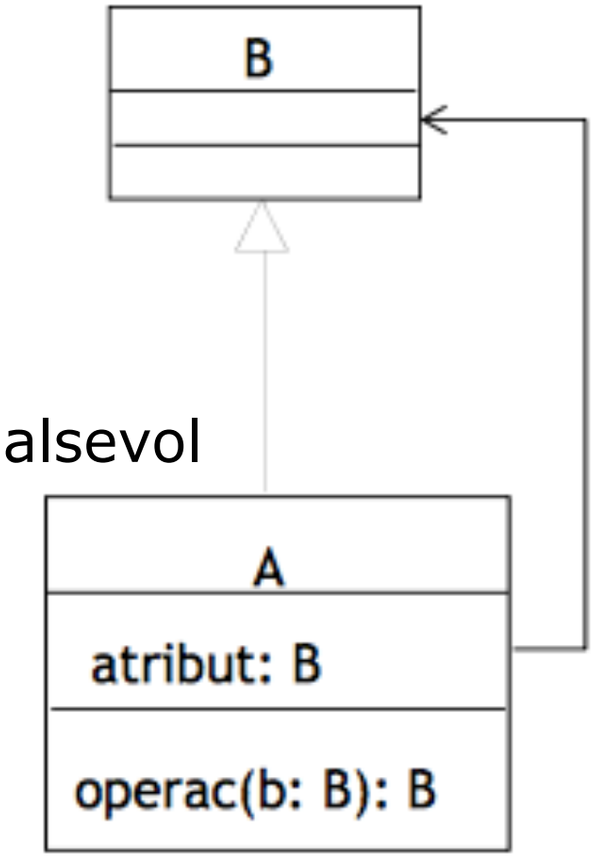
  mesura del grau de relació i de concentració de les diverses responsabilitats d'una classe (atributs, associacions, mètodes,…)

ALTA Cohesió

Una classe amb alta cohesió **no** té molts mètodes, que estan molt relacionats funcionament, i **no** realitza molt de treball. Col·labora amb altres classes.

Quan hi ha acoblament?

- Tipus d'acoblament entre les classes A i B:
  - A té un atribut que es refereix a una instància de B
  - Un objecte A invoca mètodes d'un objecte B
  - A té un mètode que referència a una instància de B de qualsevol forma (paràmetre o retorn)
  - A és una subclasse directa o indirecta de B
  - B és una interfície i A implementa B



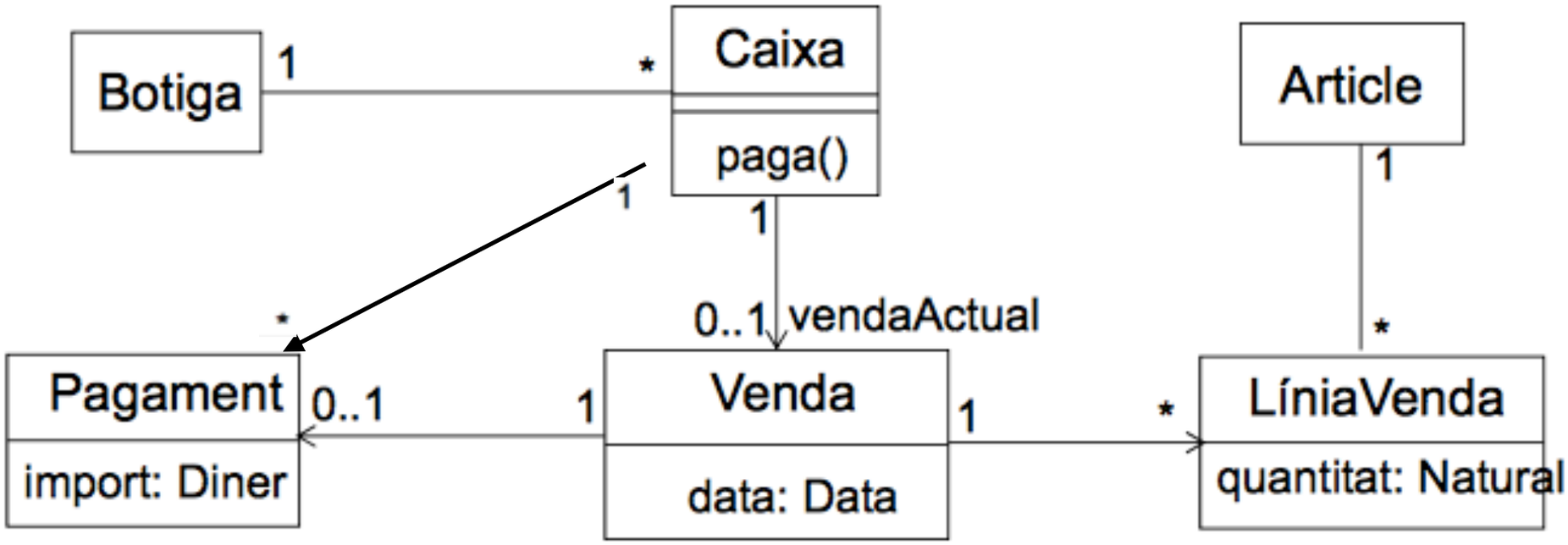
- Convé que l'acoblament sigui **baix**:
  - Si hi ha un acoblament de A a B, un canvi en B pot implicar canviar A.

- Excepcions:

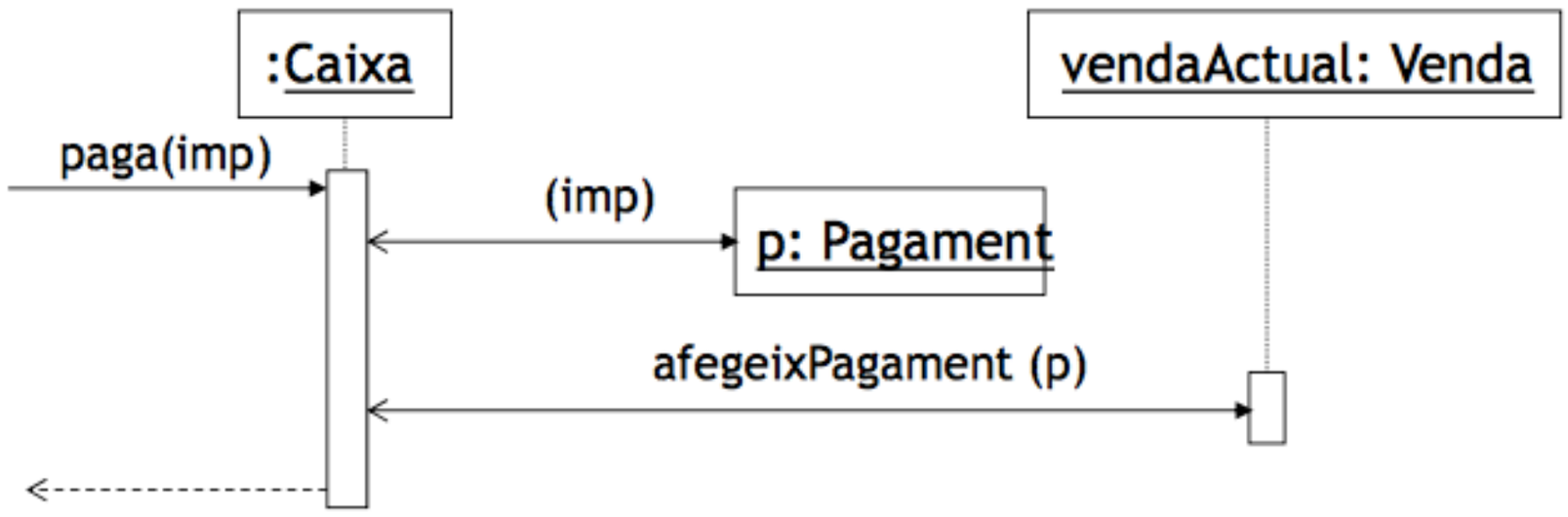
  – L'acoblament amb classes estables ben conegudes o estables no acostuma a ser problema, sobretot si estan protegides de canvis

Larman, C. "Applying UML and Patterns. An Introduction to Object-oriented Analysis and Design", Prentice Hall, 2005, 3a edició. (pag. 214-221)

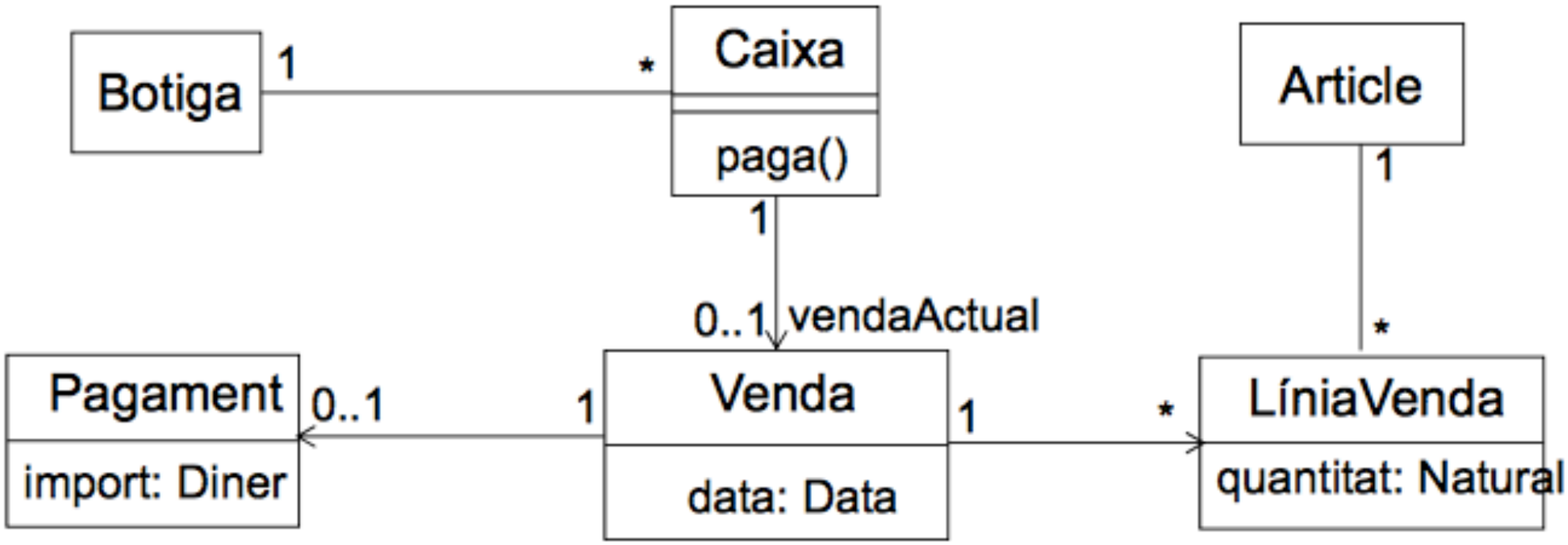
Acoblament alt



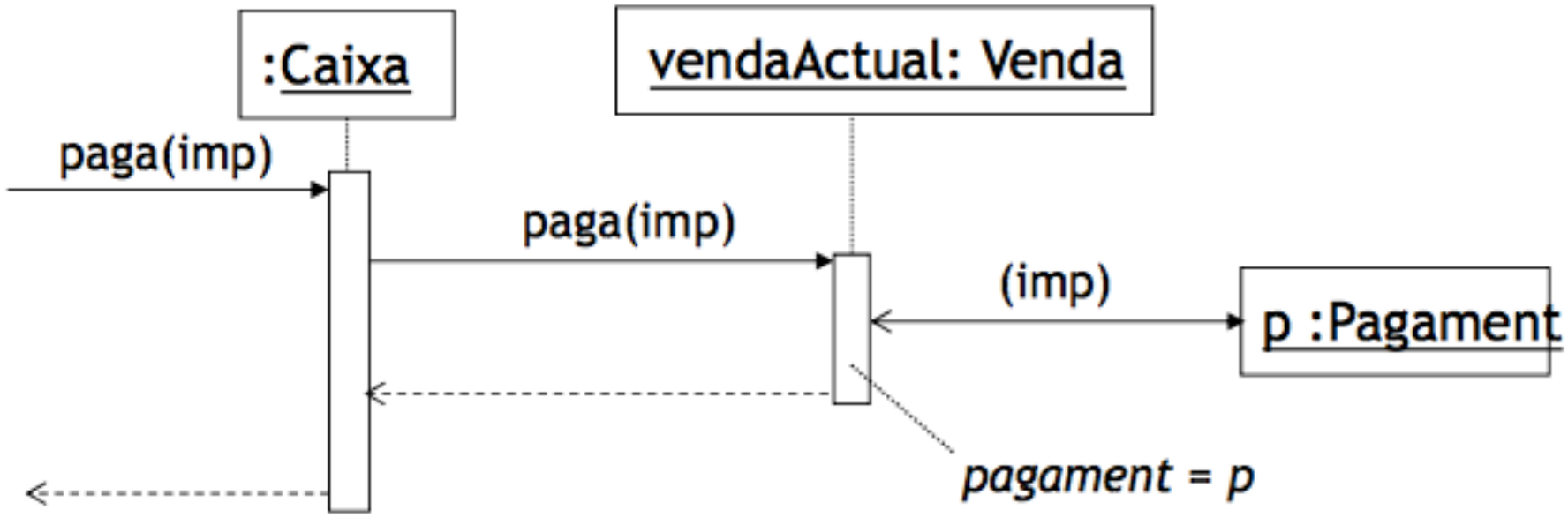
Alternativa 1:



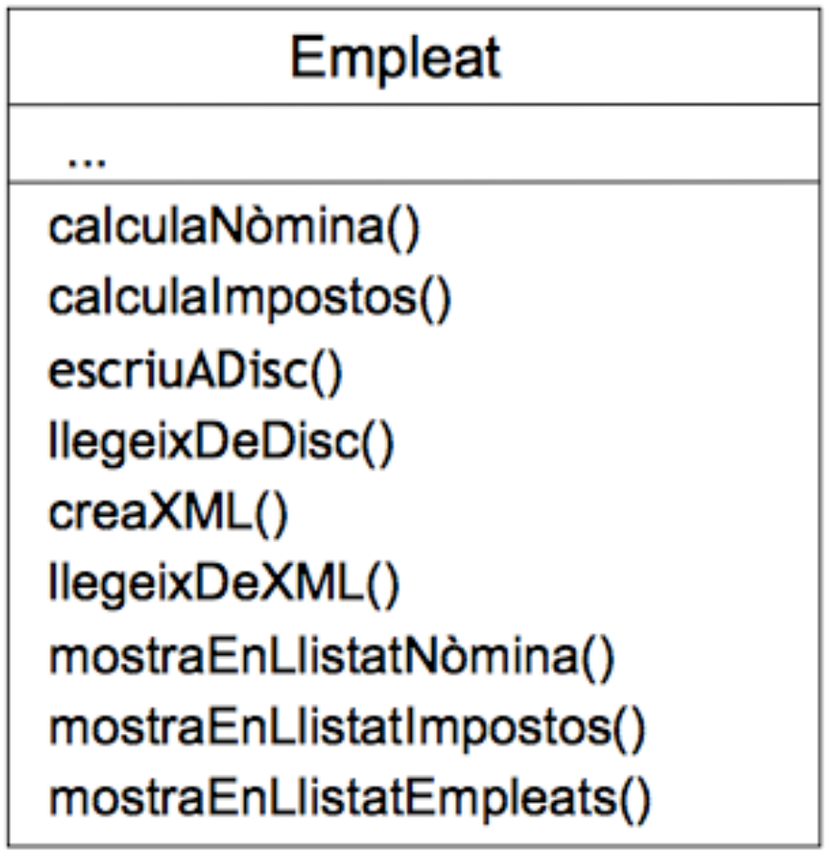
Acoblament baix



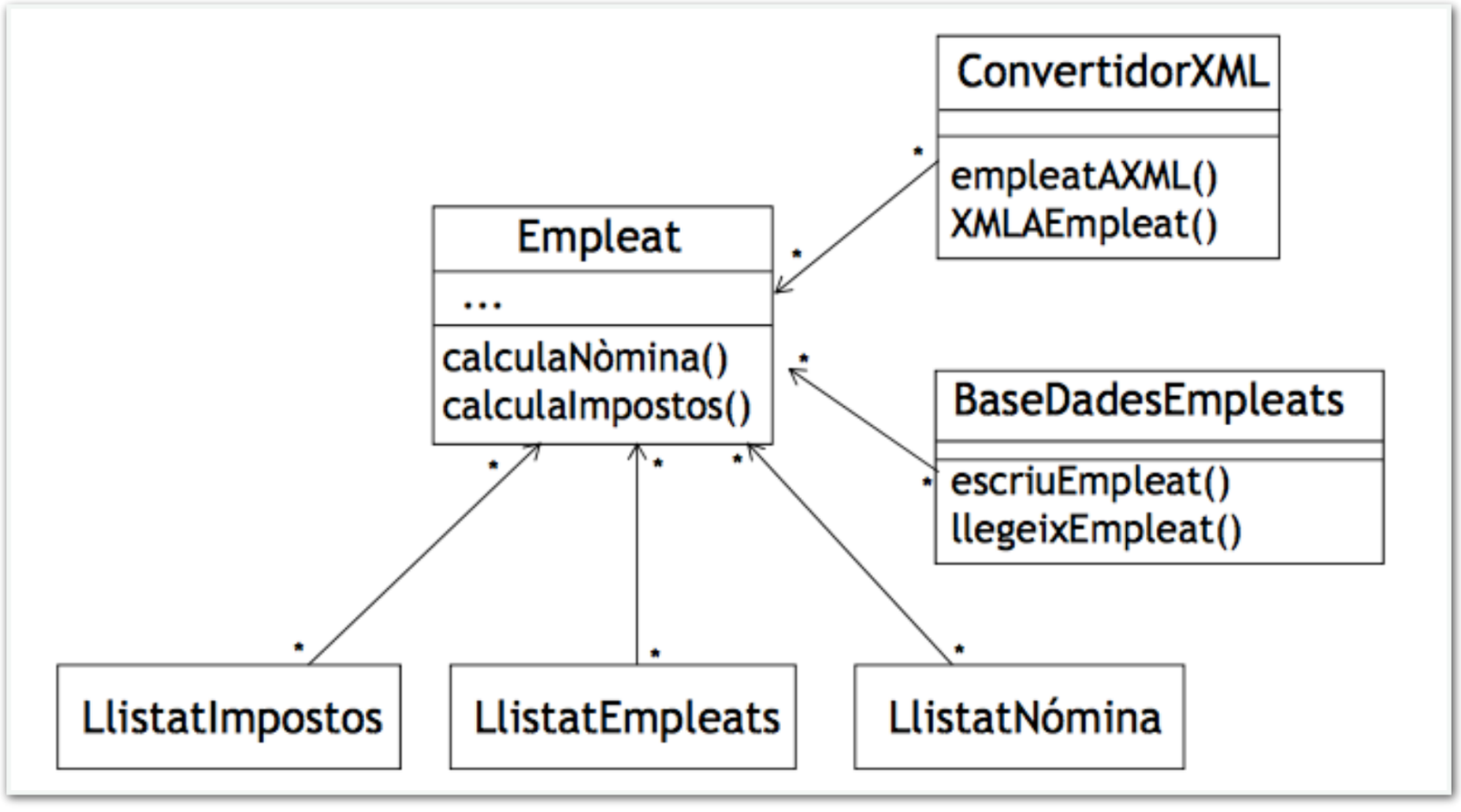
Alternativa 2:



Exemple cohesió



BAIXA Cohesió

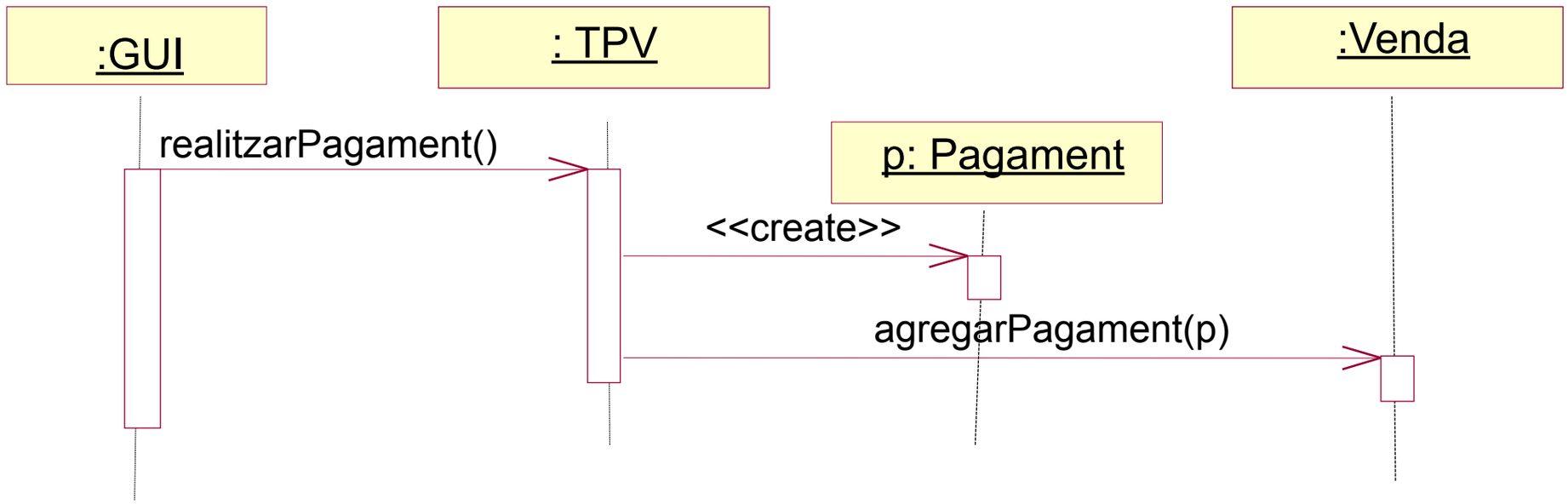


ALTA Cohesió

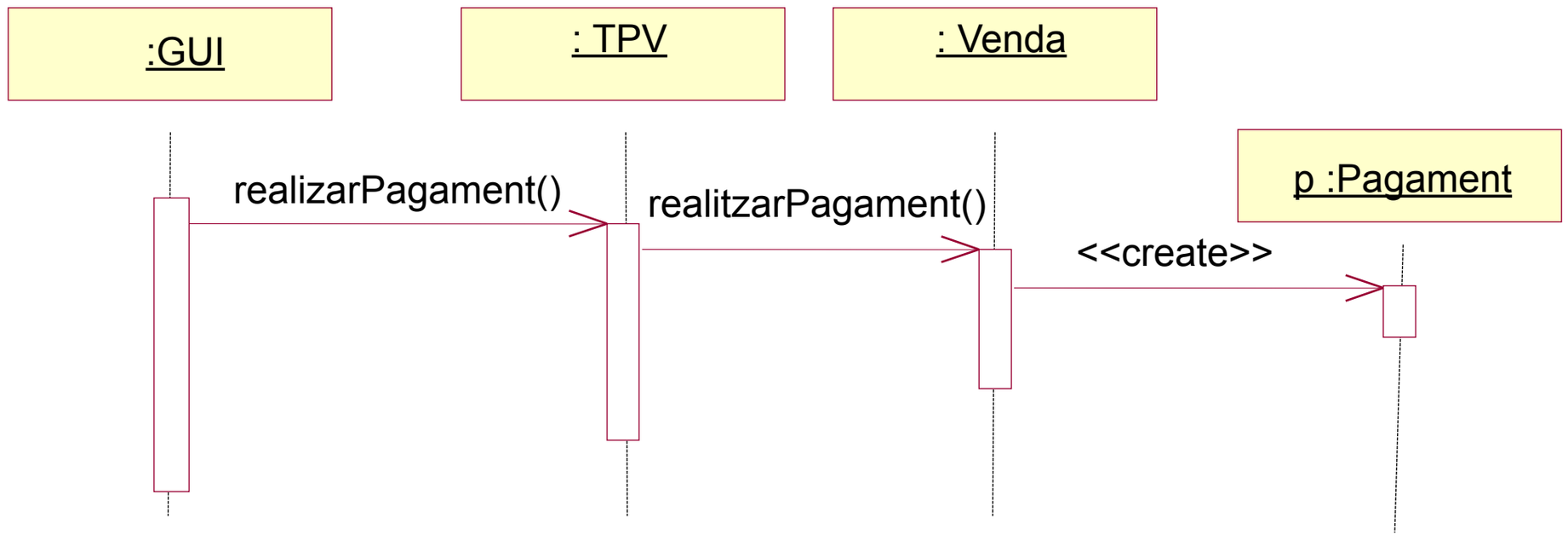
Exemple d'aplicació

- Quin d'aquests dissenys produeix una major cohesió en *TVP*?

(a)

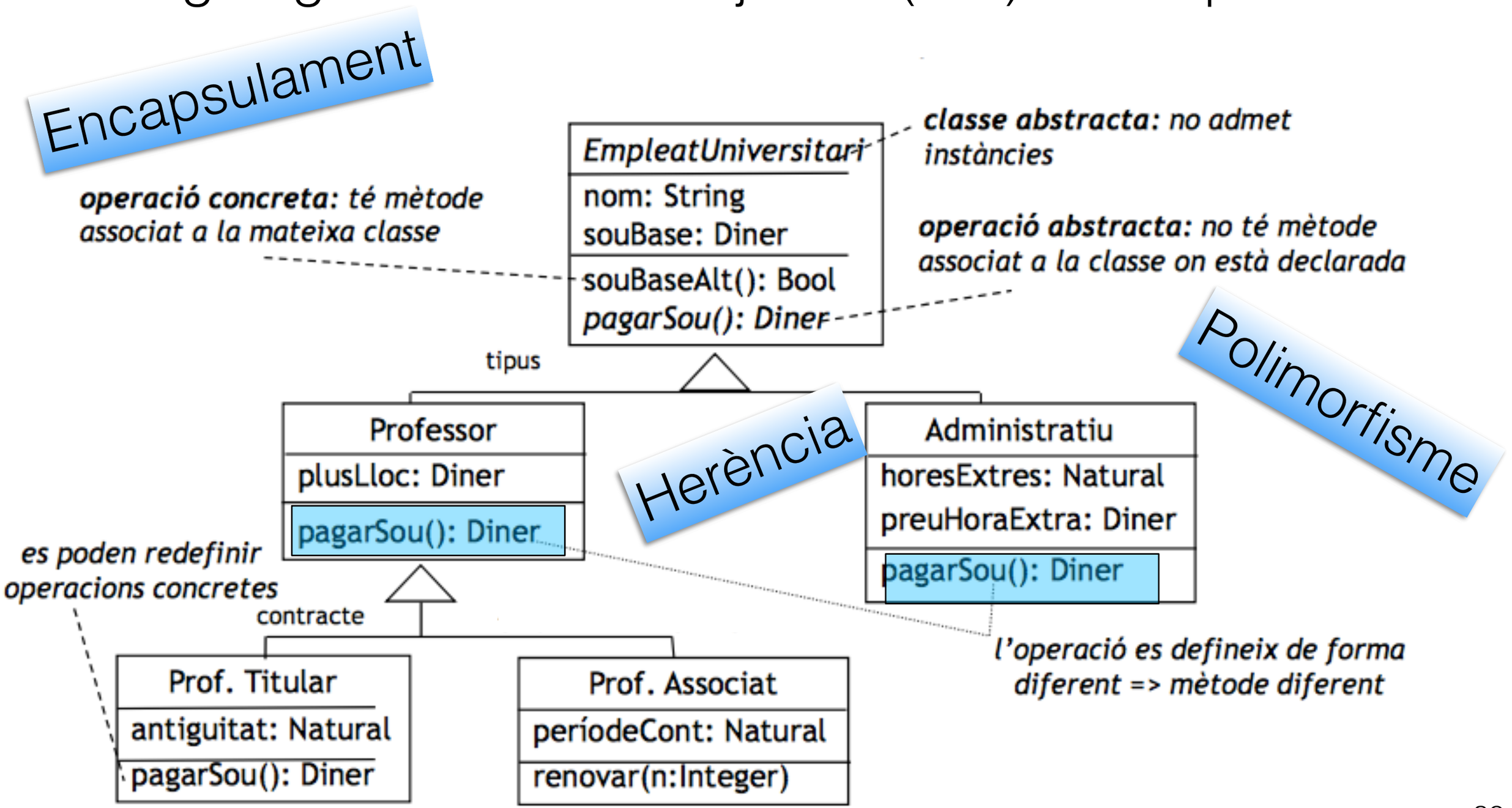


(b)



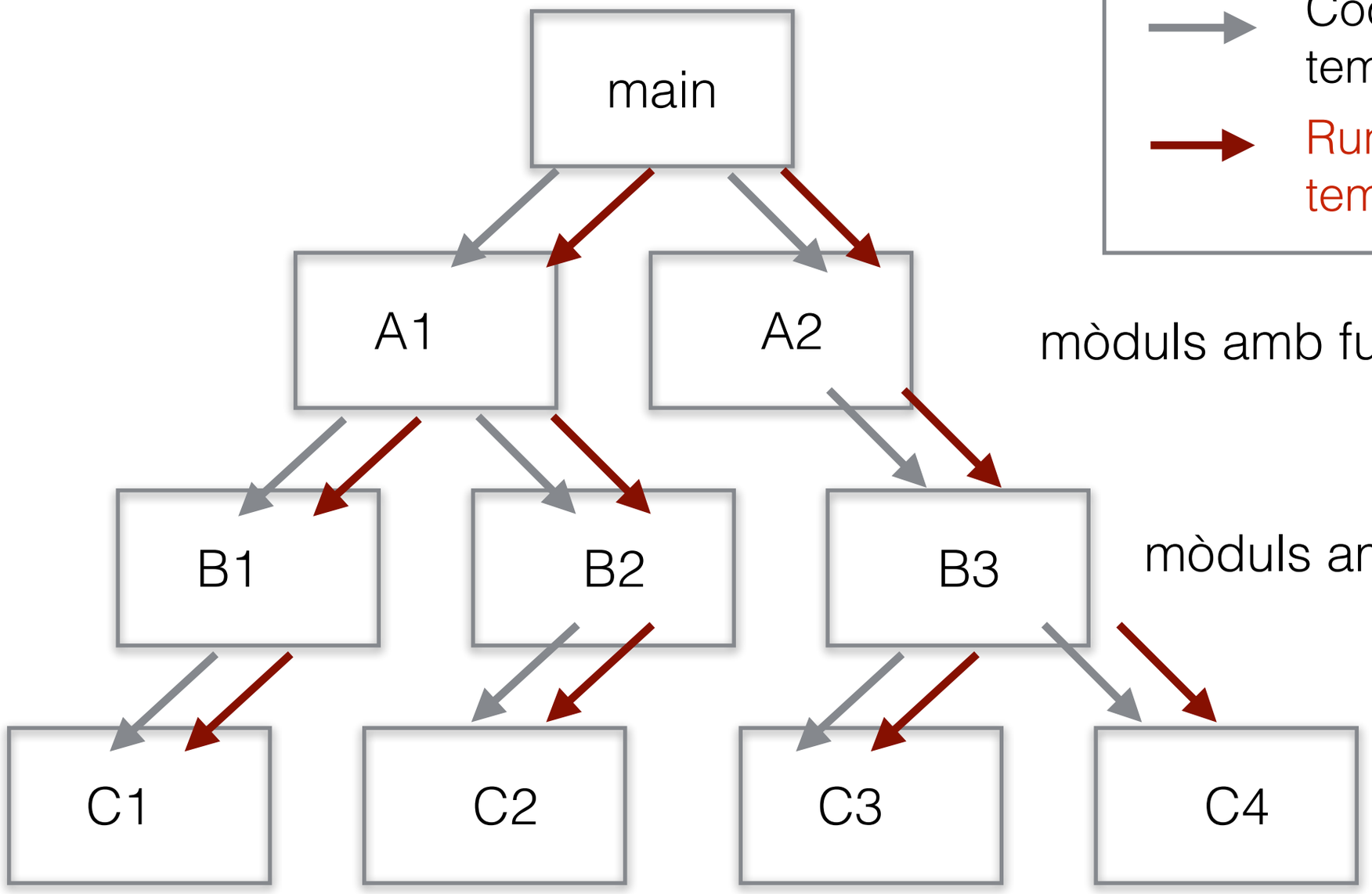
3.1. Introducció

Llenguatges Orientat a Objectes (OO): Què aporta?



3.1. Introducció

En dissenys no orientats a objectes:



Code dependency: en temps de compilació

Run-time dependency: en temps d'execució

mòduls amb funcions d'alt nivell

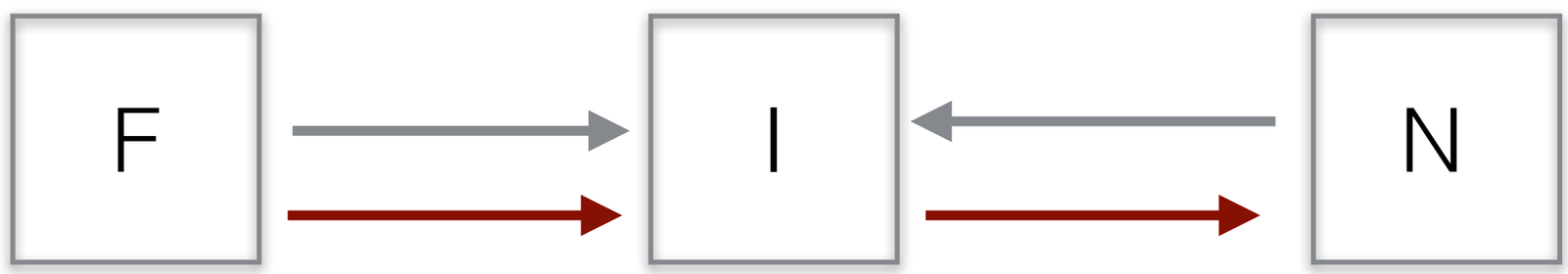
mòduls amb funcions de nivell mig

mòduls amb funcions de baix nivell

El sentit de les dependències de codi és el mateix que les dependències en execució

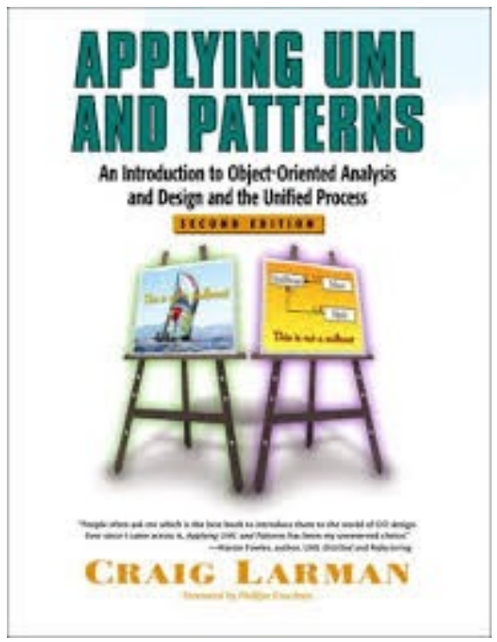
3.1. Introducció

Aportació del disseny Orientat a Objectes



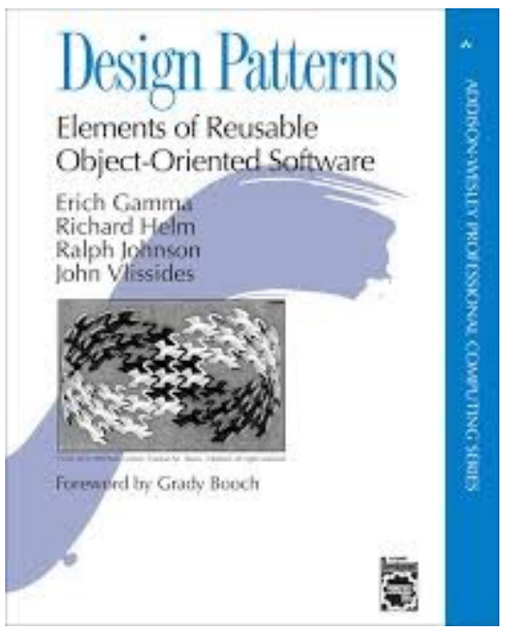
L'arquitectura d'execució no restringeix l'arquitectura del codi

Introducció als patrons



Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process.

Molt generals però bones guies de disseny

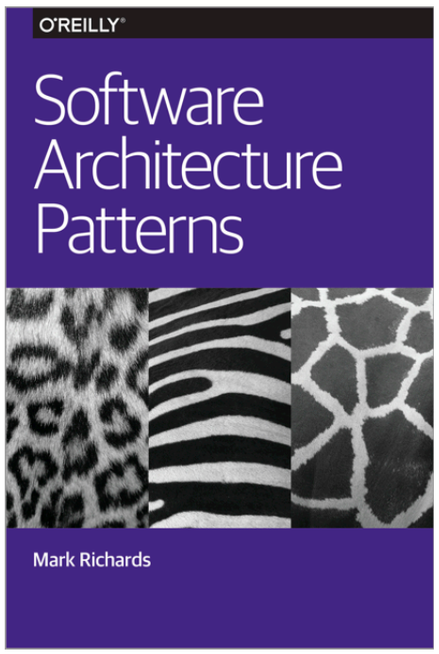


Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. *Addison-Wesley*.

S'anomenen Gang of Four (GoF), descrit a la dècada dels 90 i descriu en detall 23 patrons de disseny

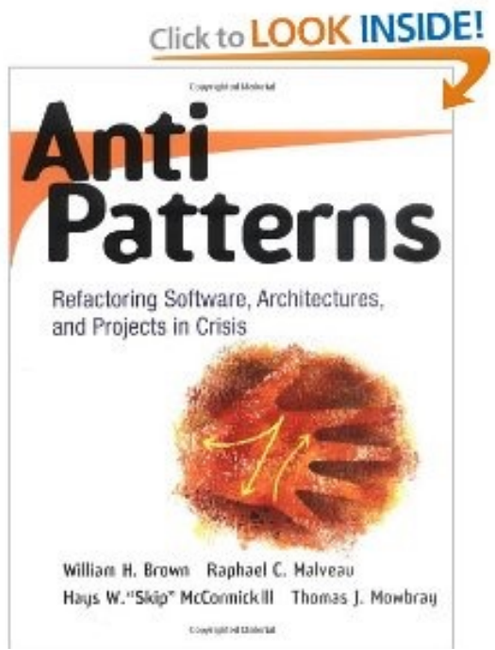
Molt utilittzats en les comunitats de desenvolupadors

Introducció als patrons



Mark Richards, Software Architecture Patterns, O'Reilly, 2015.

Patrons arquitectònics bàsics amb exemples i avaluacions



William J. Brown, Raphael C. Malveau, Hays W. McCormick, Thomas J. MowbrayWiley 1ra. Edició

*Si el concepte de **patrons** (bones solucions a problemes coneguts) resulta interessant, potser encara és més interessant el concepte d'**anti-patró** (errors comuns solucionant problemes coneguts)*