Paradigma Orientata Obiect

Cursul nr. 4

Mihai Zaharia

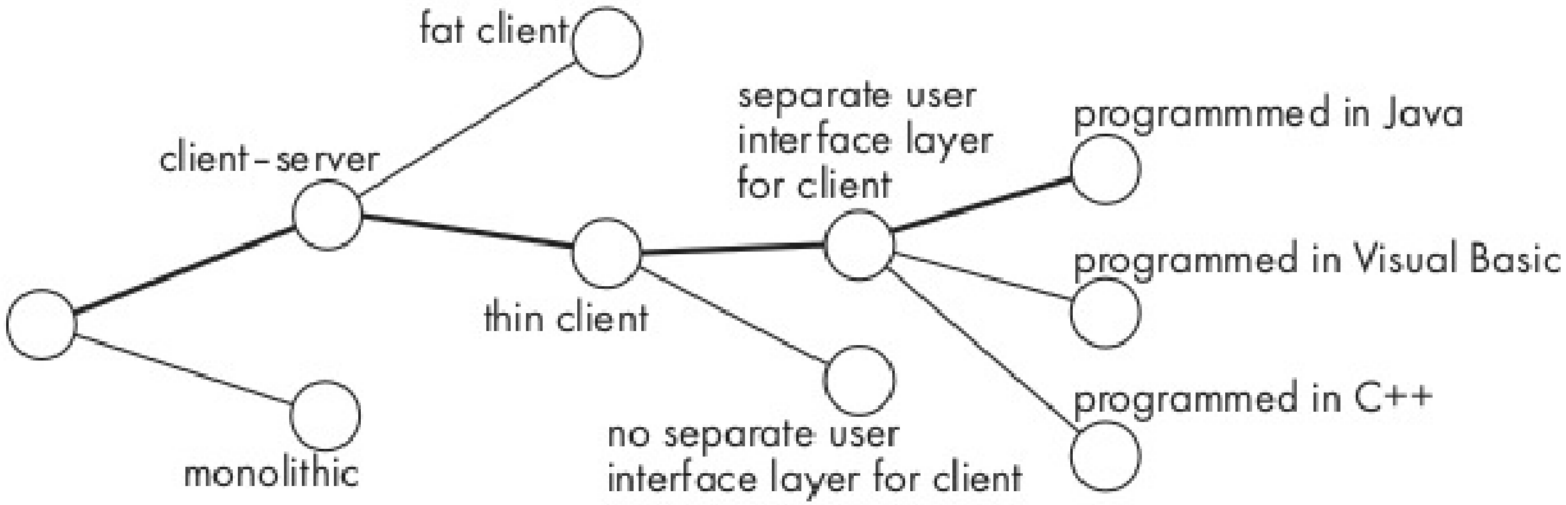
Proiectare? = o serie de decizii ....

Definiție: Proiectarea, în contextul aplicațiilor software reprezintă un proces de rezolvare a unor probleme, obiectivul fiind SĂ SE GĂSEASCĂ și SĂ SE DESCRIE o cale de a implementa NECESITĂȚILE FUNCȚIONALE ale sistemului tinând cont și de CONSTRÂNGERILE impuse de:

- nivelul de calitate așteptat,
- specificațiile platformei(lor) țintă,
- de necesitățile specifice ale procesului(lor) care trebuie modelat
- și aspectele de business cum ar fi:
  - termenul de realizare și
  - fondurile disponibile pentru respectivul proiect.

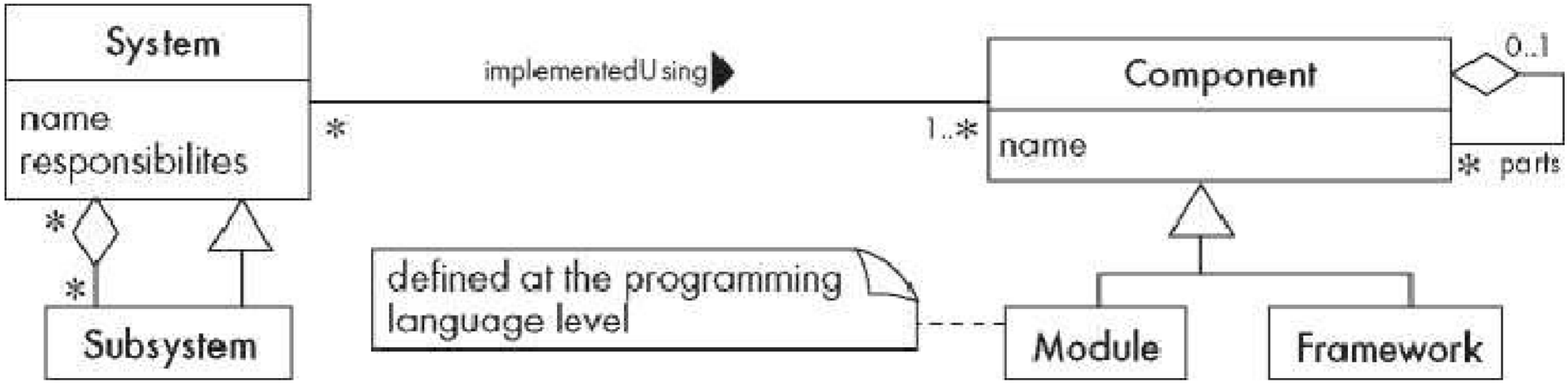
Sistemul

- Sistemul care va fi proiectat este alcătuit din:
  - subsisteme
  - componente și
  - module.



Arbore decizional general specific primei analize după stabilirea specificațiilor și a constrângerilor

Modelul domeniului



Acesta explică conceptele de sistem, subsistem, componentă și modul

Ce este de fapt UML?

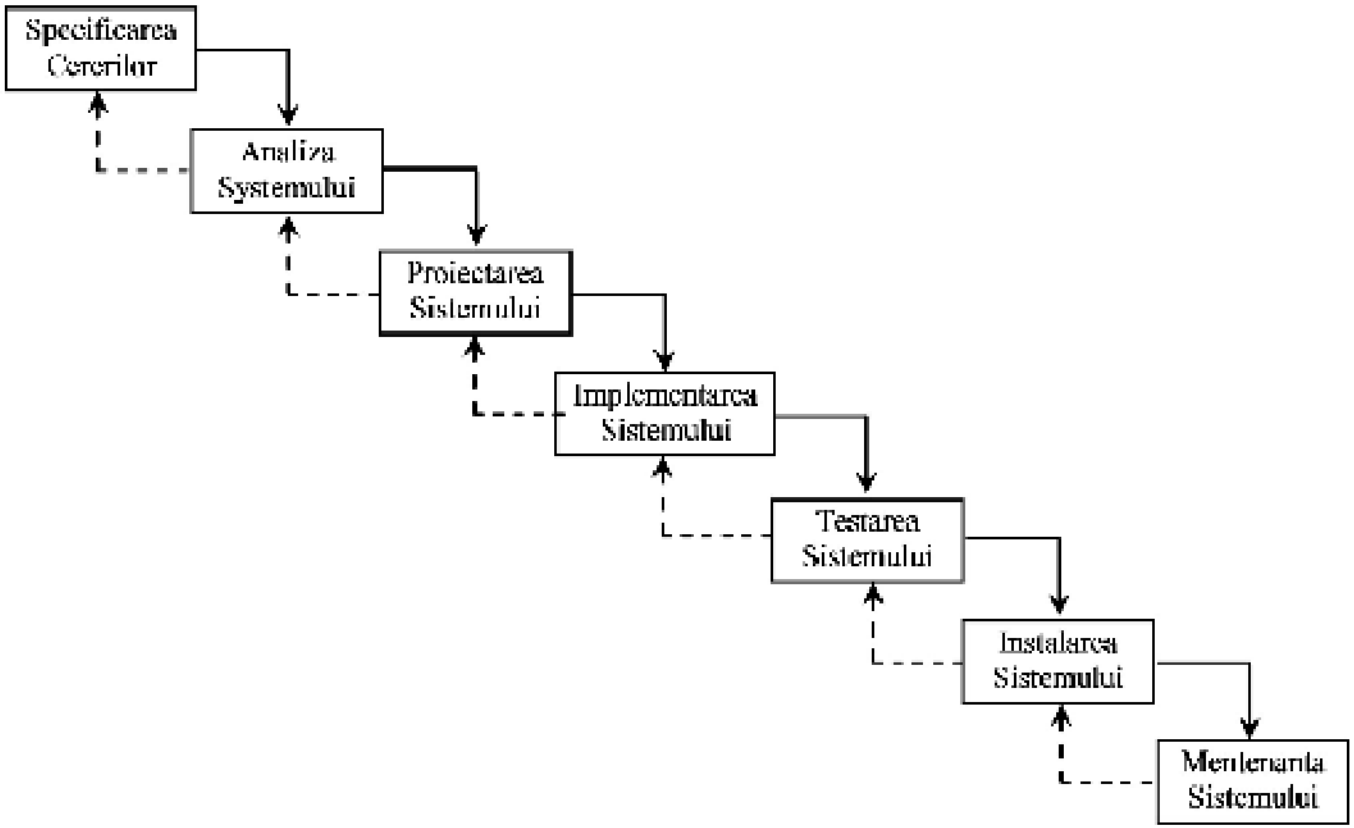
Definiție: (OMG)

- *" Unified Modeling Language (UML) reprezintă un limbaj grafic pentru vizualizarea, specificarea, dezvoltarea și documentarea componentelor unui sistem software de dimensiuni medii sau mari.*
- *UML oferă o manieră standard pentru a crea schema unui sistem pornind de la aspecte concrete cum ar fi blocuri de cod, scheme pentru bazele de date, componente reutilizabile și ajungând la aspecte abstracte precum capturarea fluxului de desfășurare a unei afaceri sau funcții ale sistemului."*

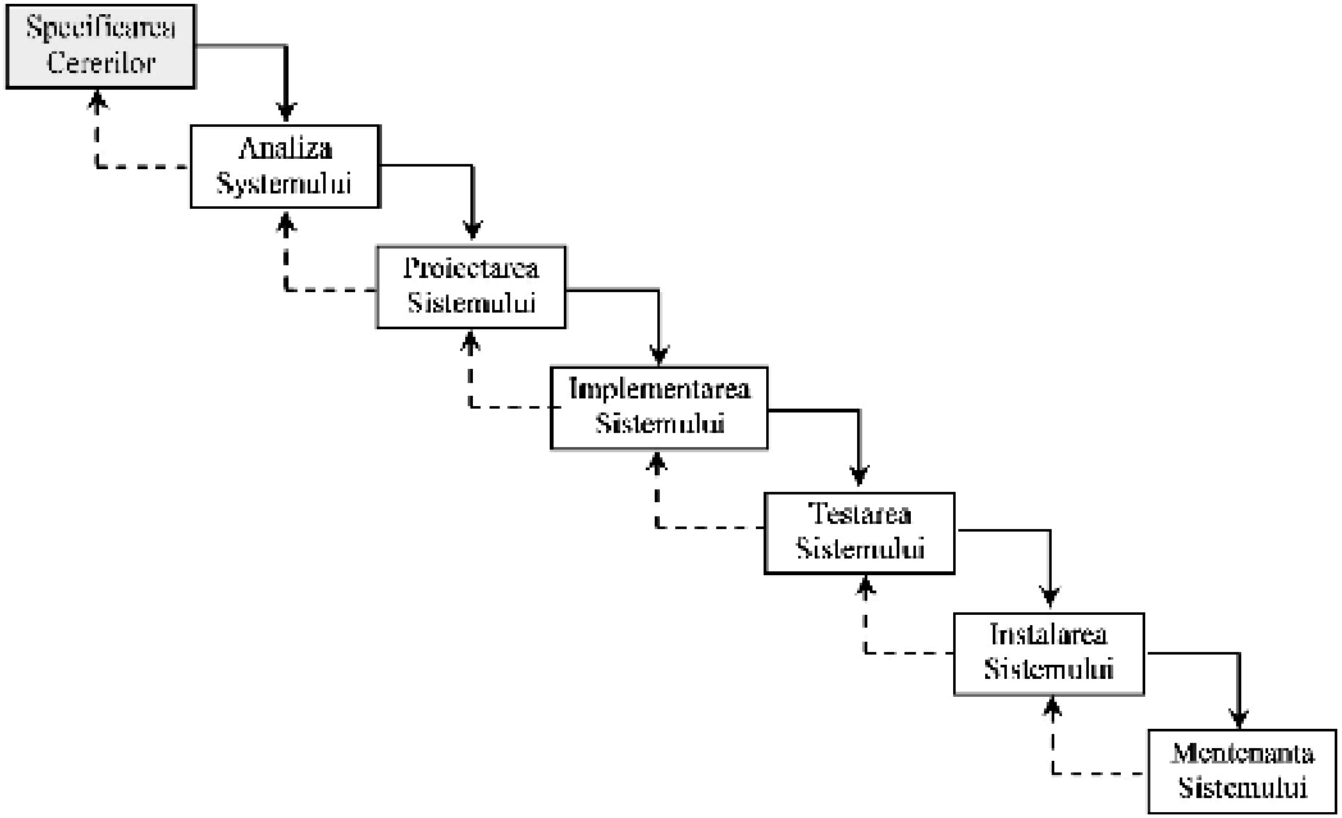
UML oferă semantici și notații pentru

1. User Interaction/Use Case Model
2. Interaction / Communication Model
3. State / Dynamic Model
4. Logical / Class Model
5. Physical Component Model
6. Physical Deployment Model
7. Mai definește și soluții pentru extinderea sau dezvoltarea după nevoi a unor mecanisme existente

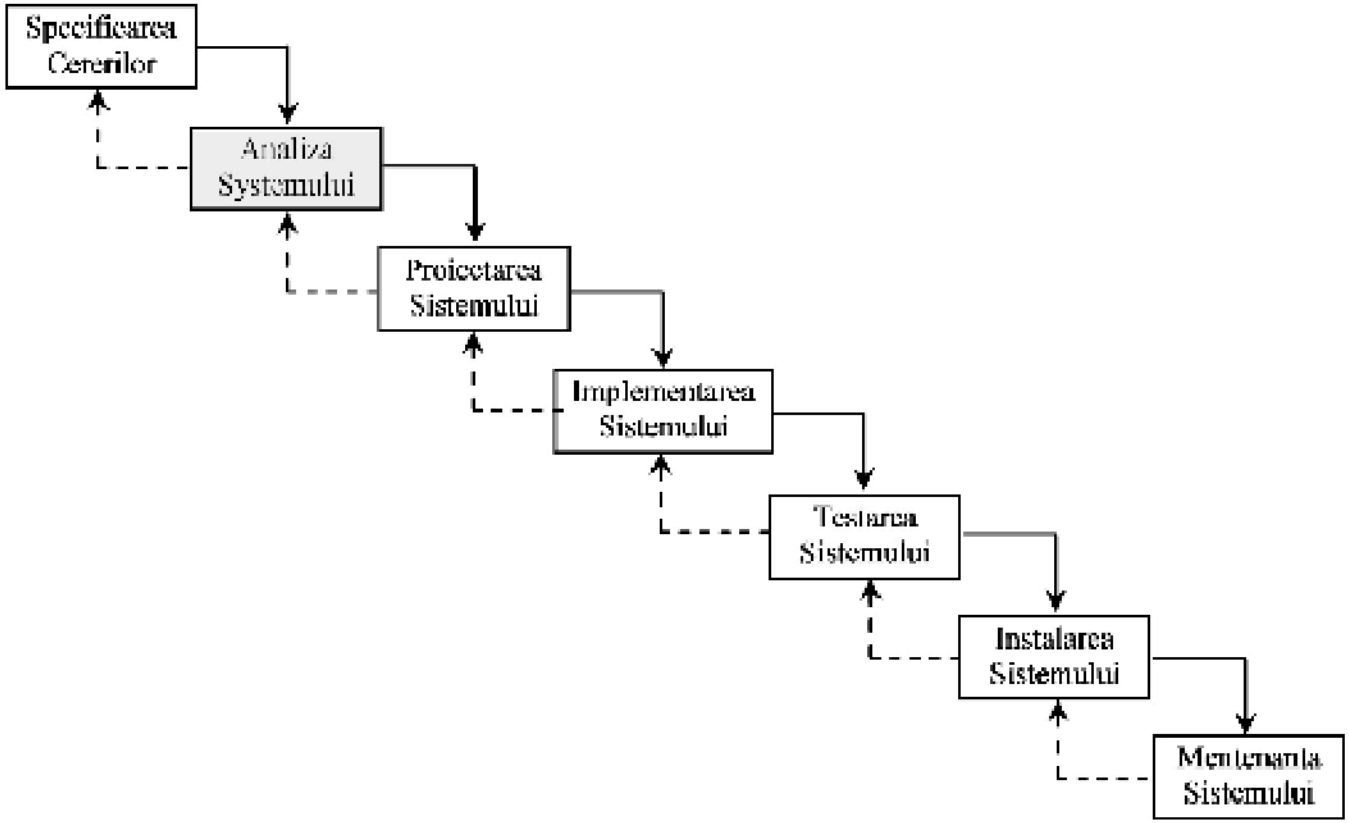
Procesul dezvoltare a unei aplicații



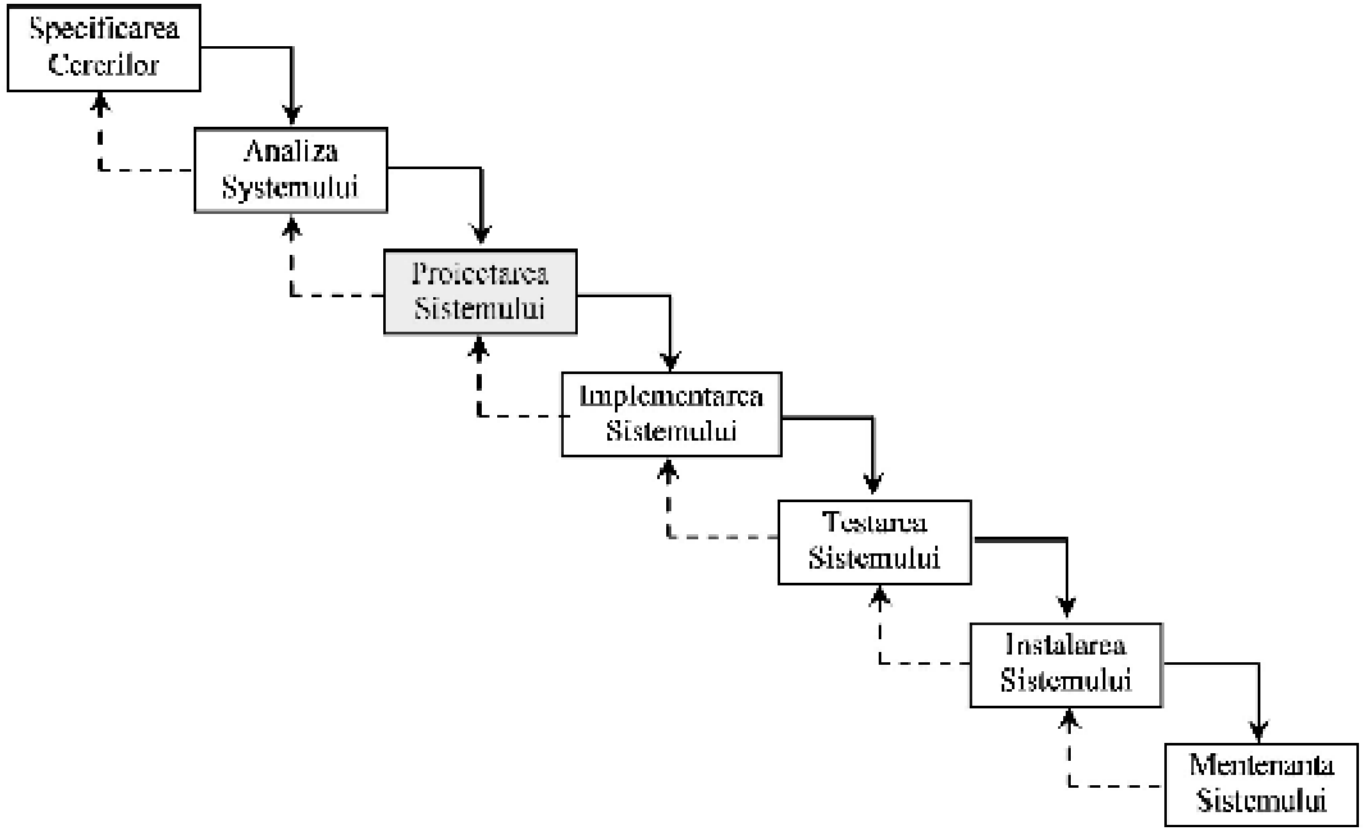
Specificarea Cererilor



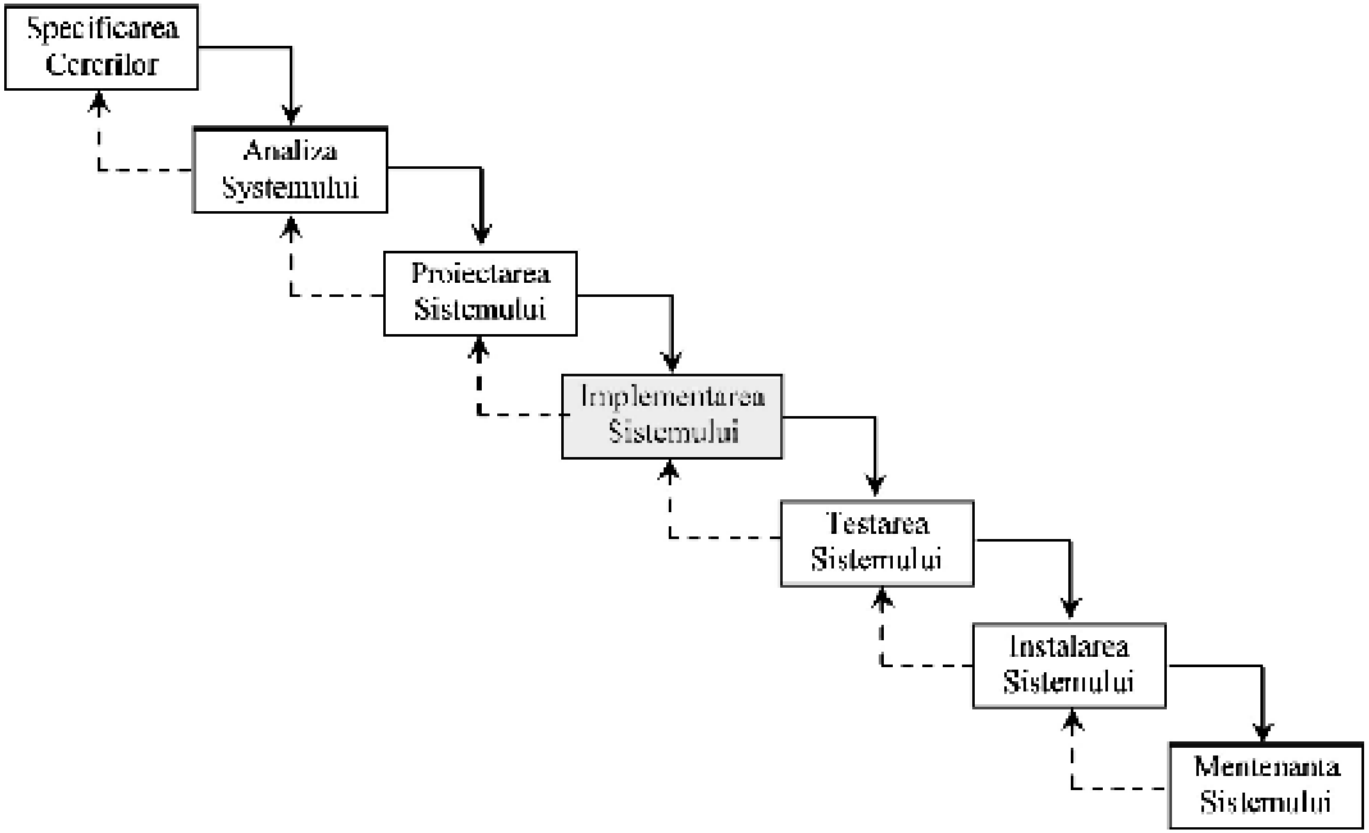
Analiza sistemului



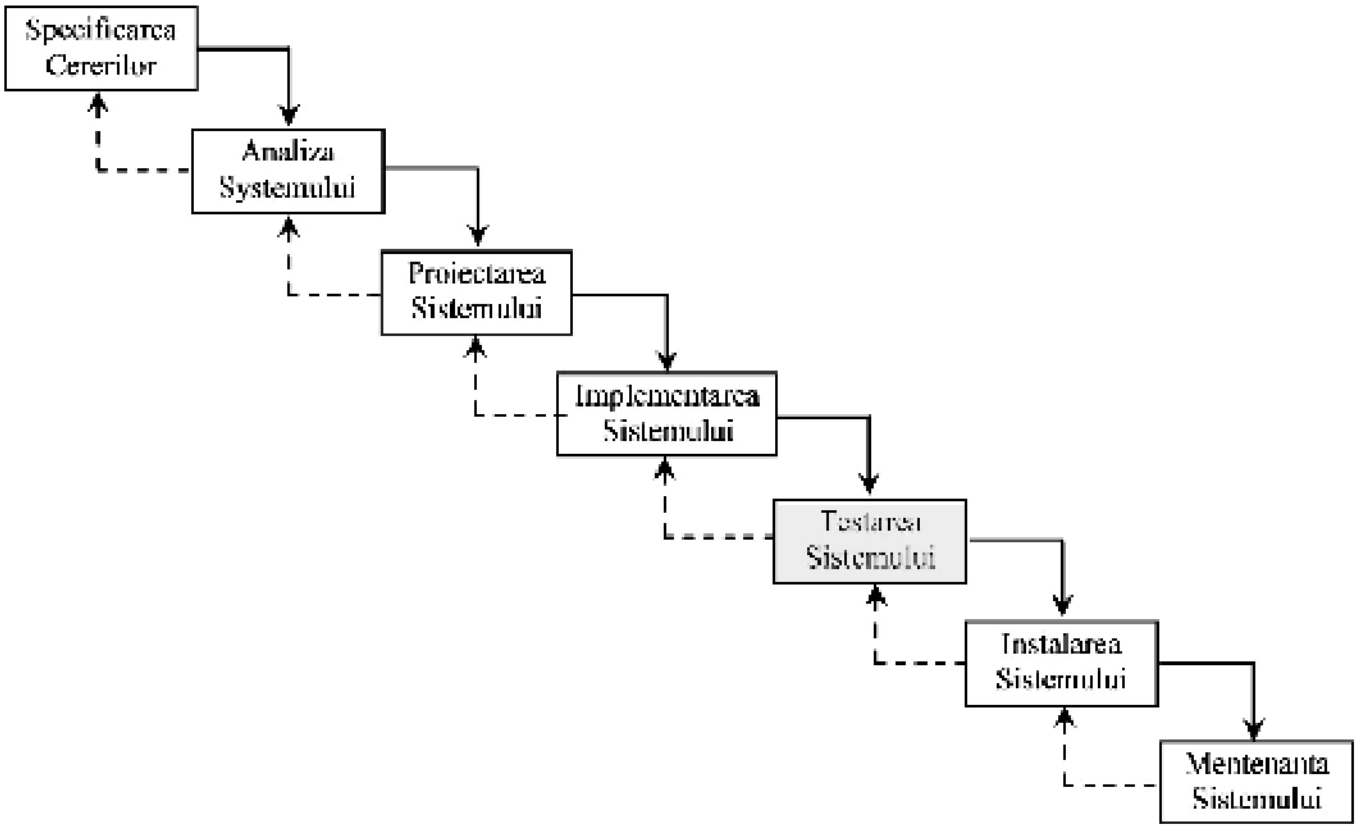
Proiectarea sistemului



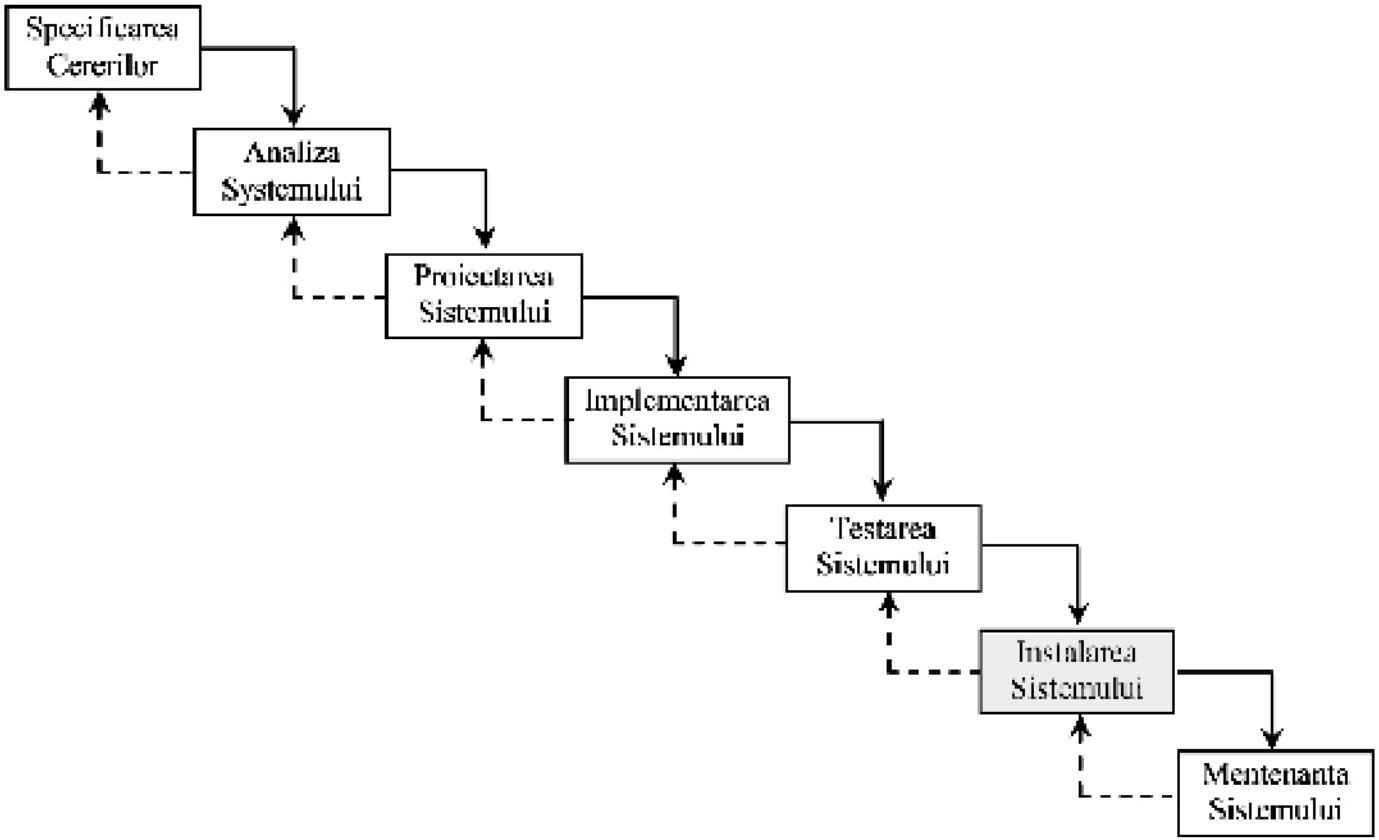
Implementarea sistemului



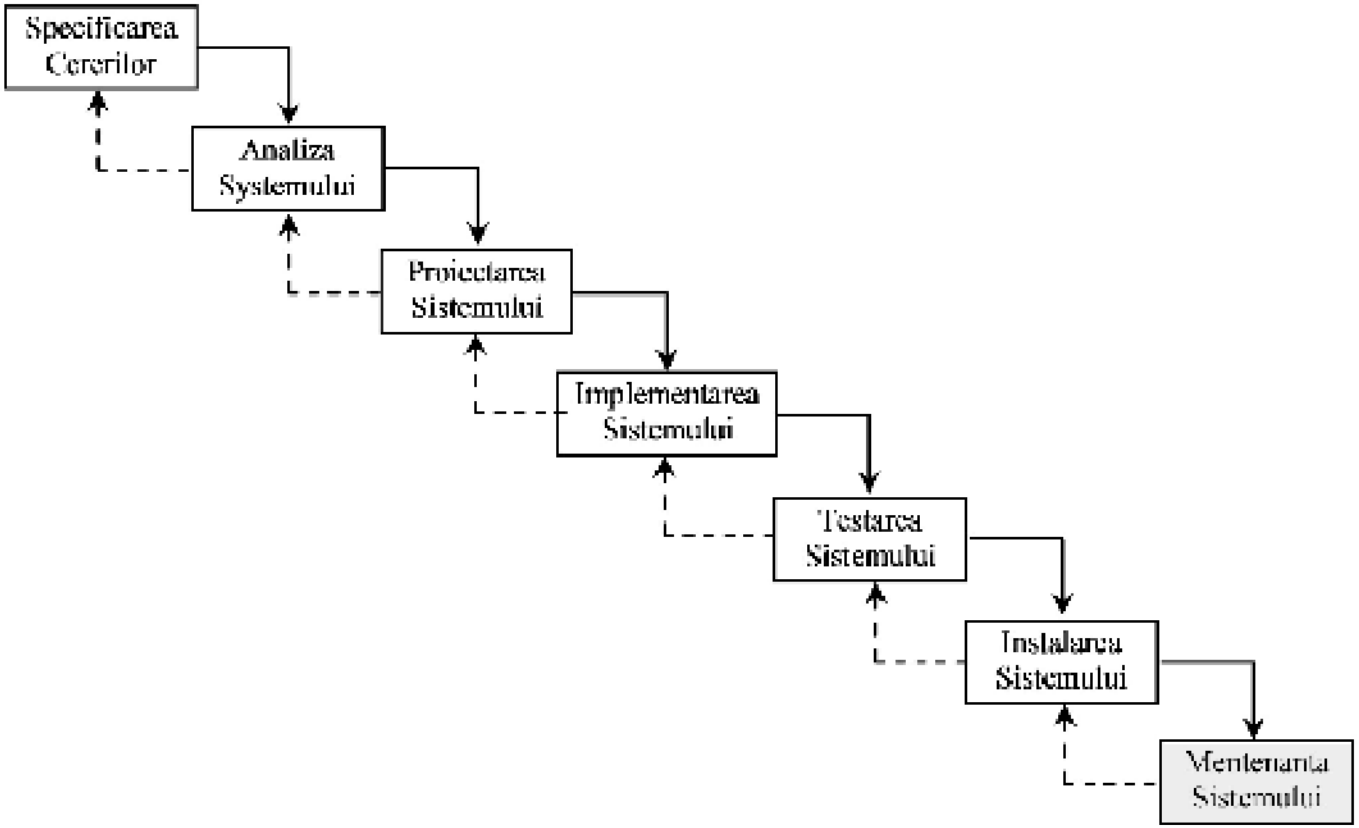
Testarea sistemului



Instalarea sistemului



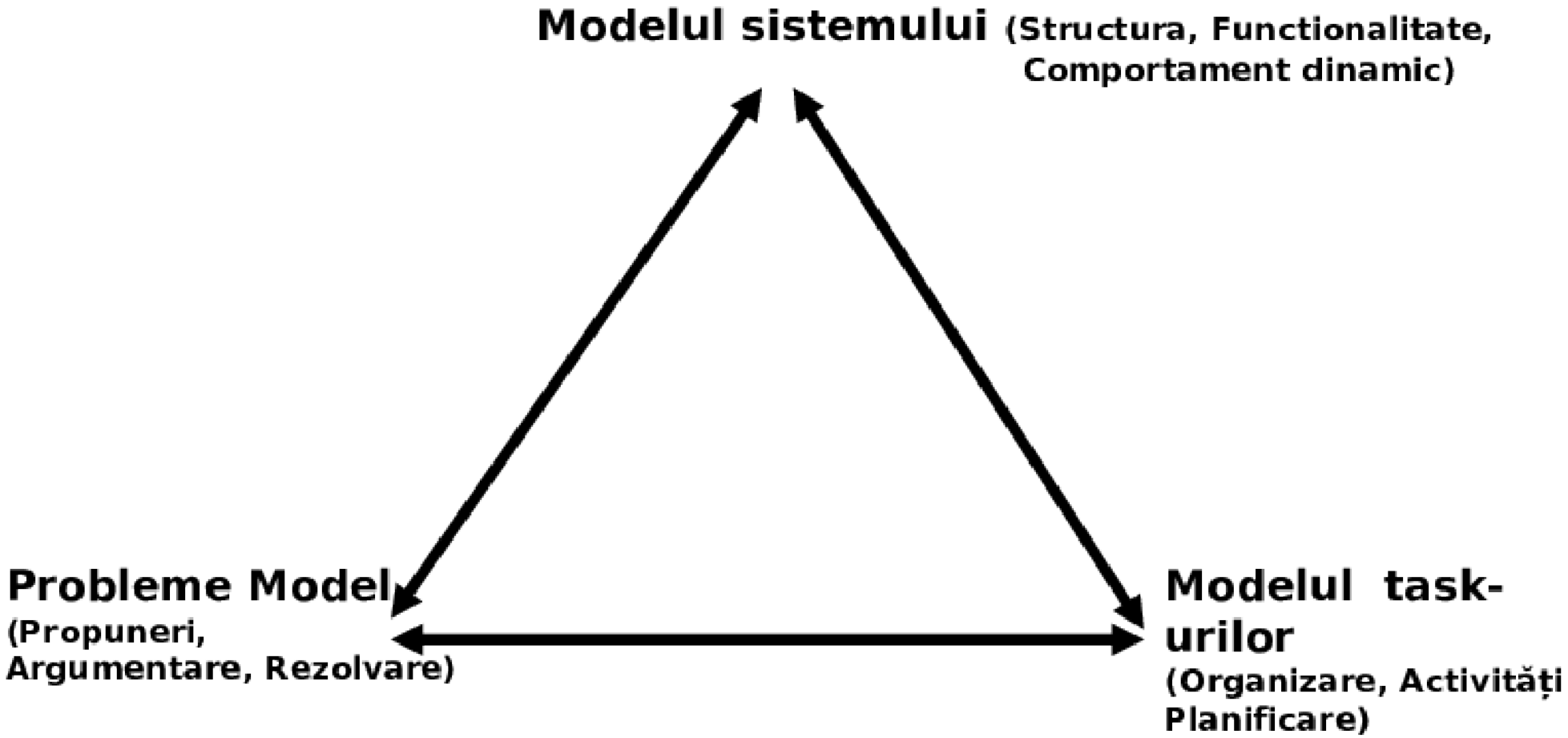
Intreținerea sistemului



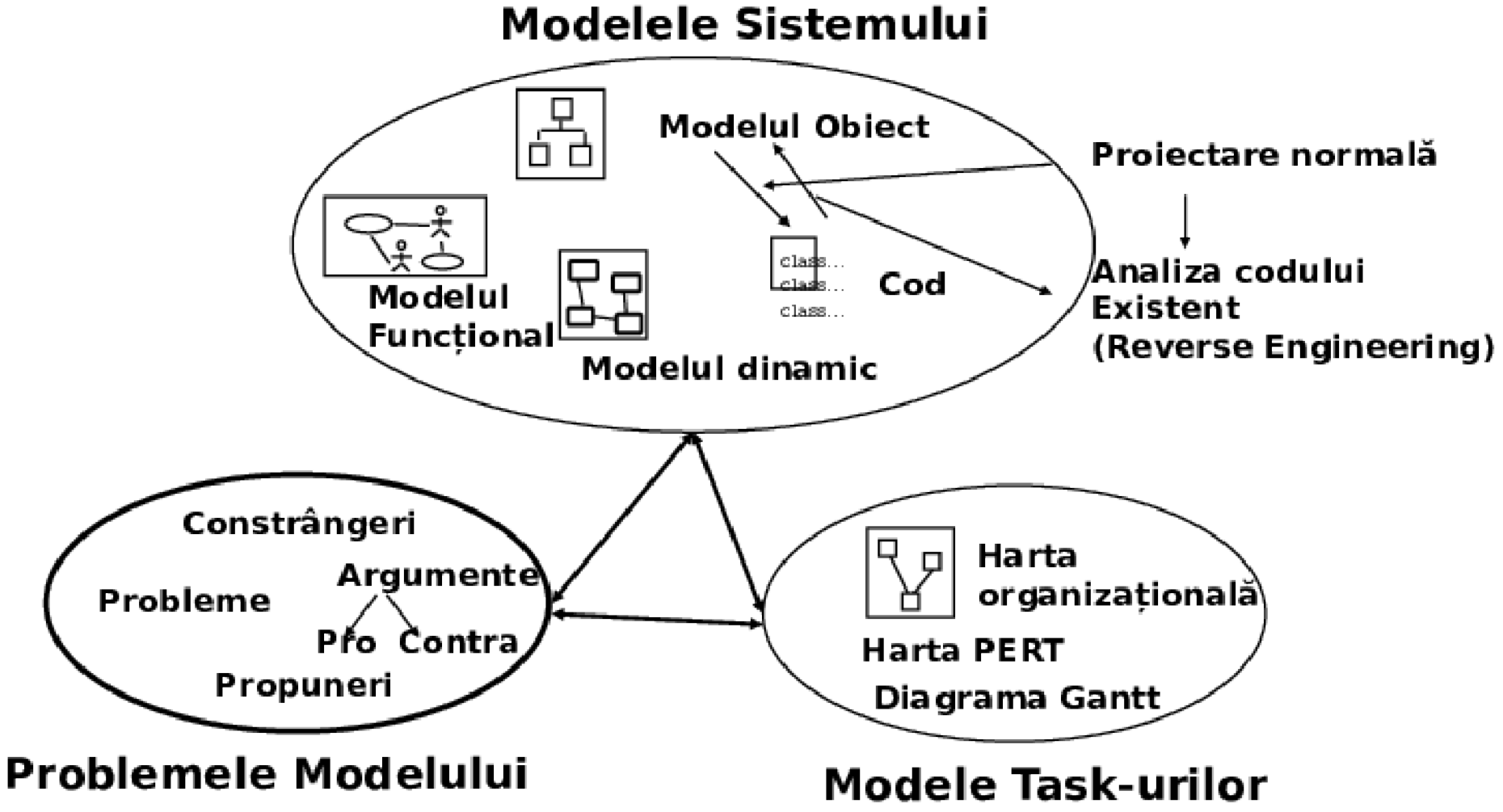
Modelele sunt folosite pentru a furniza descrierile abstracte

- Modelul sistem:
  - Modelul obiect:
  - Modelul functional:
  - Modelul dinamic:
- Modelul task-urilor:
  - Harta PERT: Care sunt legaturile dintre task-uri?
  - Planificarea: Cum poate fi aceasta realizată în durata de timp rezervată?
  - Harta organizațională: Care sunt rolurile în proiect sau organizație?
- Problemele Modelului:

Relațiile modelului

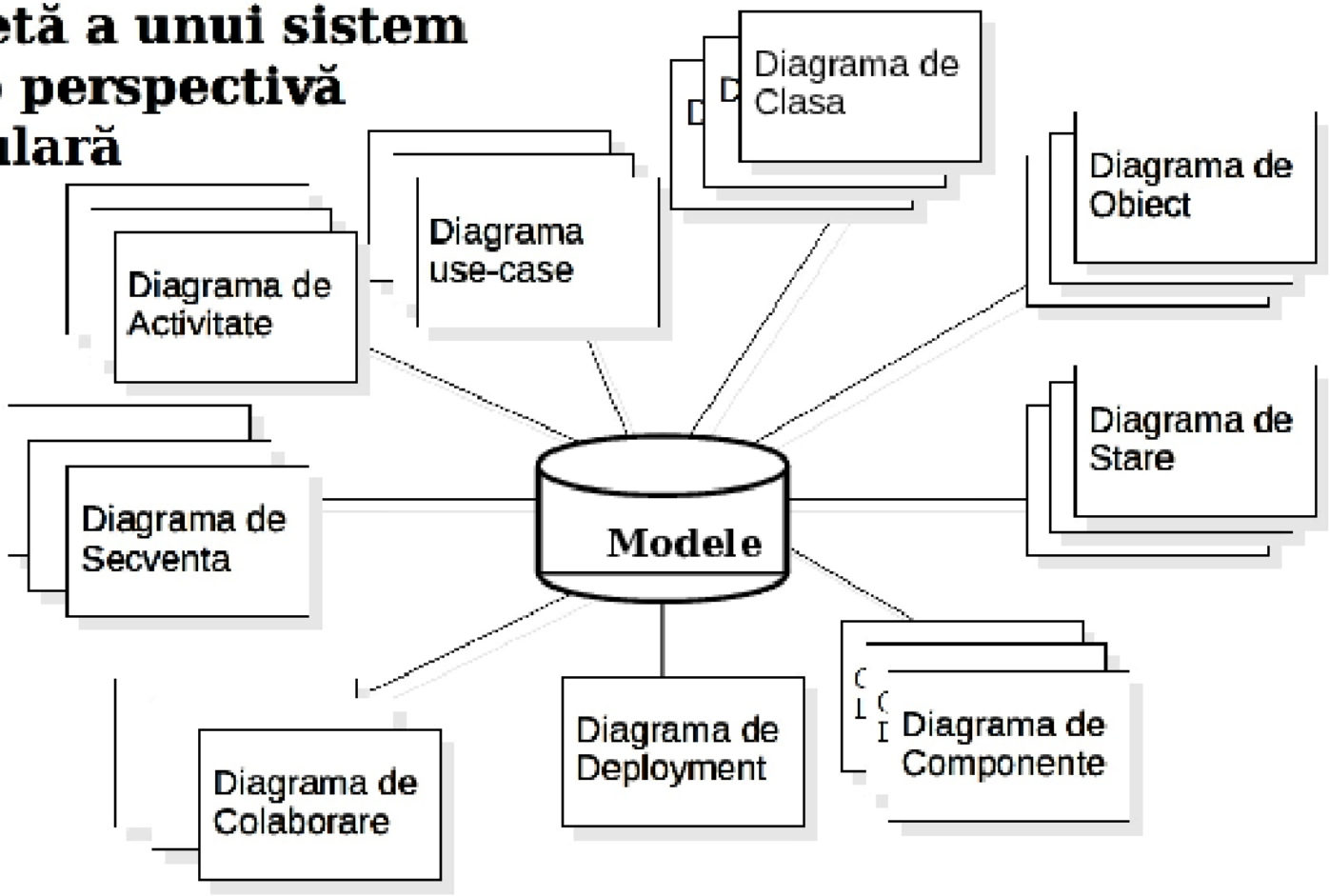


Problema modelării



O diagramă este o reprezentare vizuală într-un model

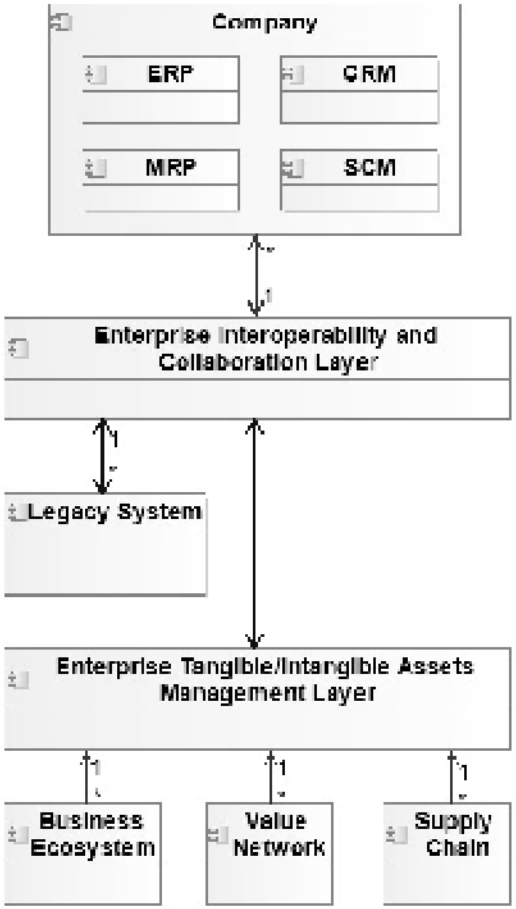
Un model este o descriere completă a unui sistem dintr-o perspectivă particulară



Tipuri de proiectare specifice

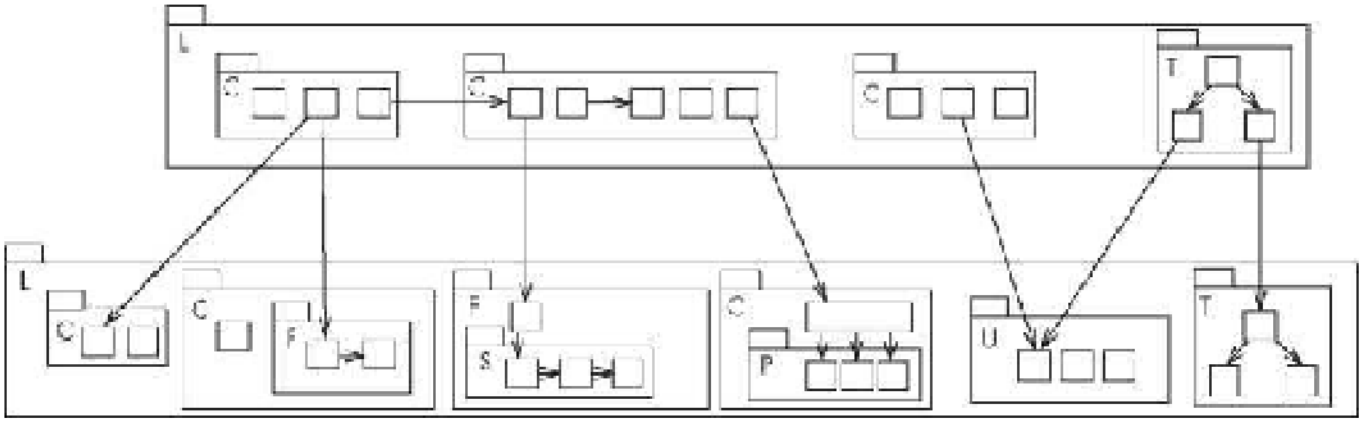
- Proeictare arhitecturală:
- Proiectarea claselor
- proiectarea interfeței vizuale
- proiectarea bazelor de date
- proiectarea algoritmilor
- proiectarea protocoalelor

Bune practici în proiectare(BPP) Divide and conquer

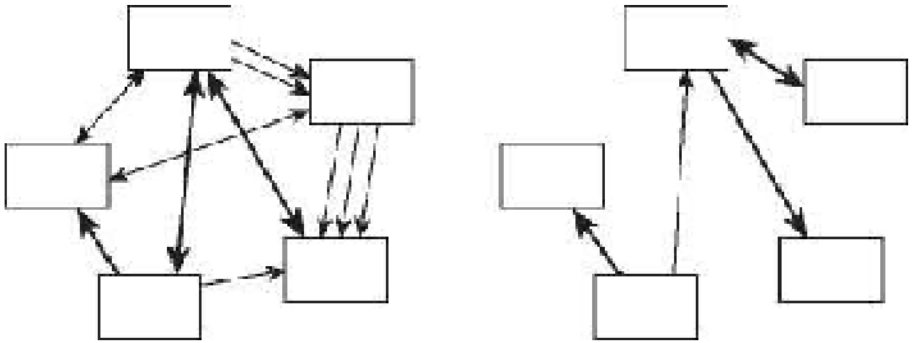


BPP - Increase cohesion where possible

- Coeziunea: “Aplică D&C dar păstrează cât mai ”
- Tipuri de coeziune posibile:
- Funcțională
- La nivel de strat arhitectural
- La nivel de comunicații
- Secvențială
- Procedurală
- Temporală



BPP - Reduce coupling where possible



- Cuplarea apare atunci când sunt interdependențe între un modul sau altul
- În figură sunt prezentate cele două cazuri dominante sisteme cu coeziune stransa sau slaba.
- Dacă un sistem are o coeziune mai stransă (câteodată din cauza unei proiectări deficitare câteodată datorită limitărilor specifice paradigmei de programare și a bibliotecilor sau facilităților tehnologiei folosite el este mai greu de inteles sau și de modificat deci are o mentenanță problematică de-a lungul ciclului de viață.

Tipuri de cuplare:

- Conținut
- Comun
- Control
- Etichetă
- Date
- Apel de funcție
- Utilizare de tip
- Incluziune/Import
- Externă

BPP - Keep the level of abstraction as high as possible

- ascund pădurea ca să nu mă încurc de copaci
- detaliile depre copaci frunze și veverișe pot fi furnizate
  - la o etapă ulterioară pe ciclul de proeictare
  - prin compilator sau la runtime
  - prin valori implicite

BPP - Increase reusability where possible

- Există două principii complementare legate de reutilizare.
- “proiectează pentru reutilizare”
- “proiectează prin reutilizare”

BPP - Reuse existing designs and code where possible

BPP - Design for flexibility

BPP - Anticipate obsolescence

BPP - Design for portability

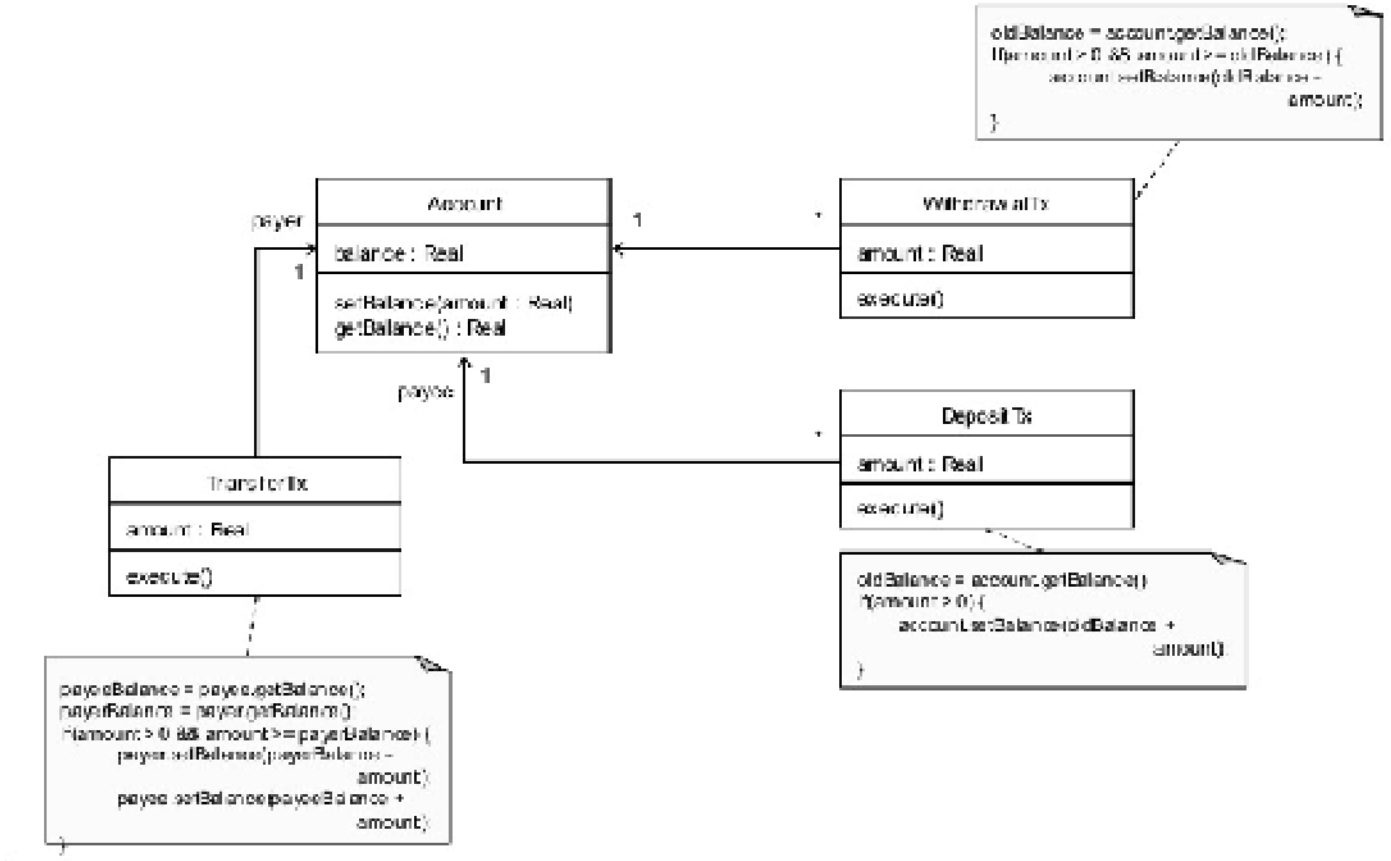
BPP - Design for testability

Scopurile proiectării OOP

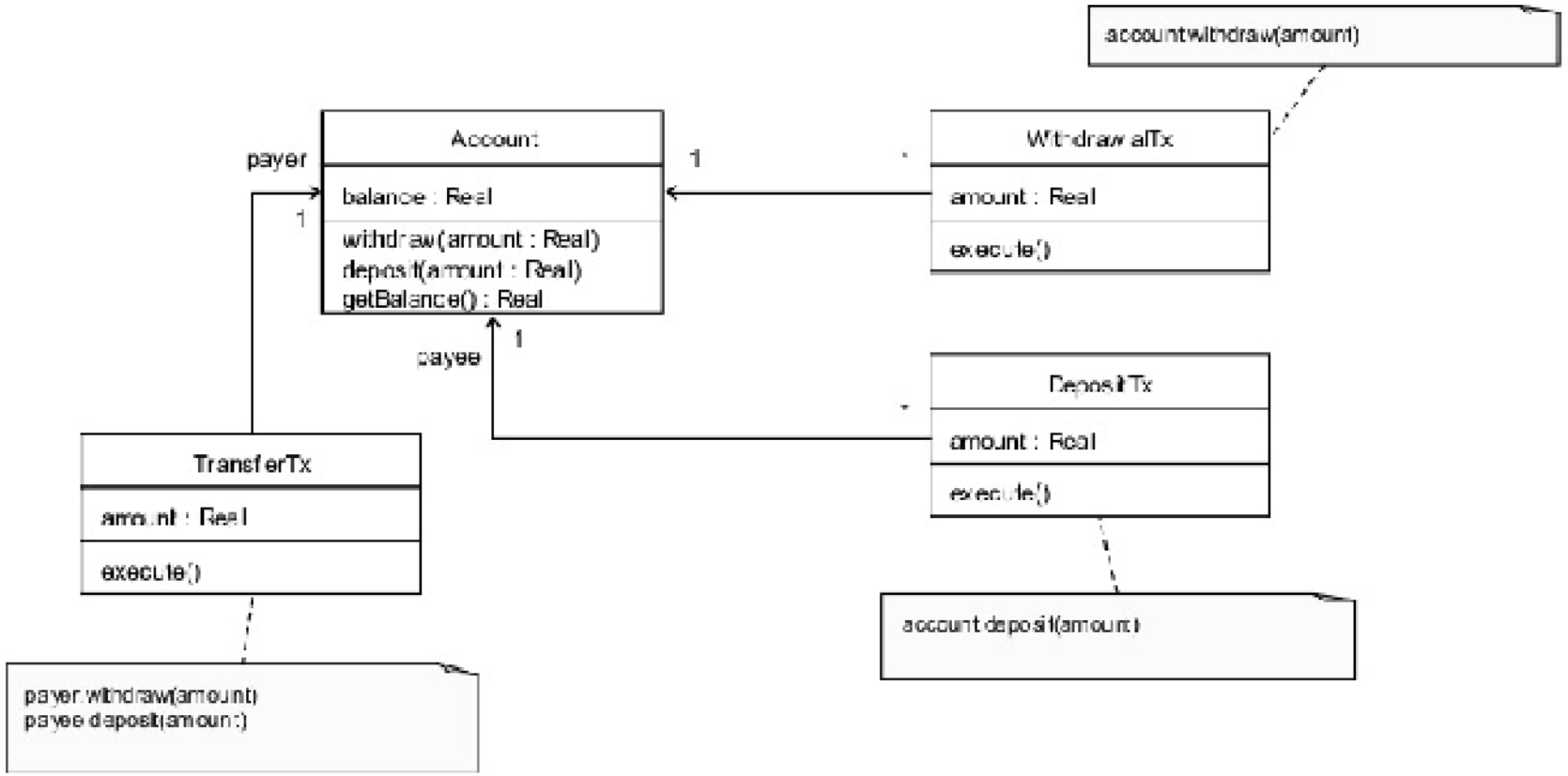
Principii OOP pentru proiectarea claselor

- Coeziunea Claselor
- Închisă vs. Deschisă
- Răspundere Unică
- Separarea Interfețelor
- Dependența Inversă
- Substituția Liskov
- Legea lui Demeter
- Reutilizarea Abstracțiilor

Coeziunea Claselor - înainte..

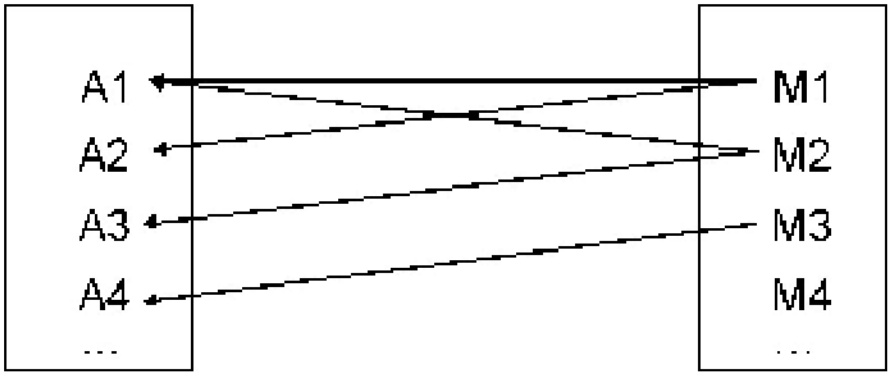


Coeziunea Claselor - și după refactorizare



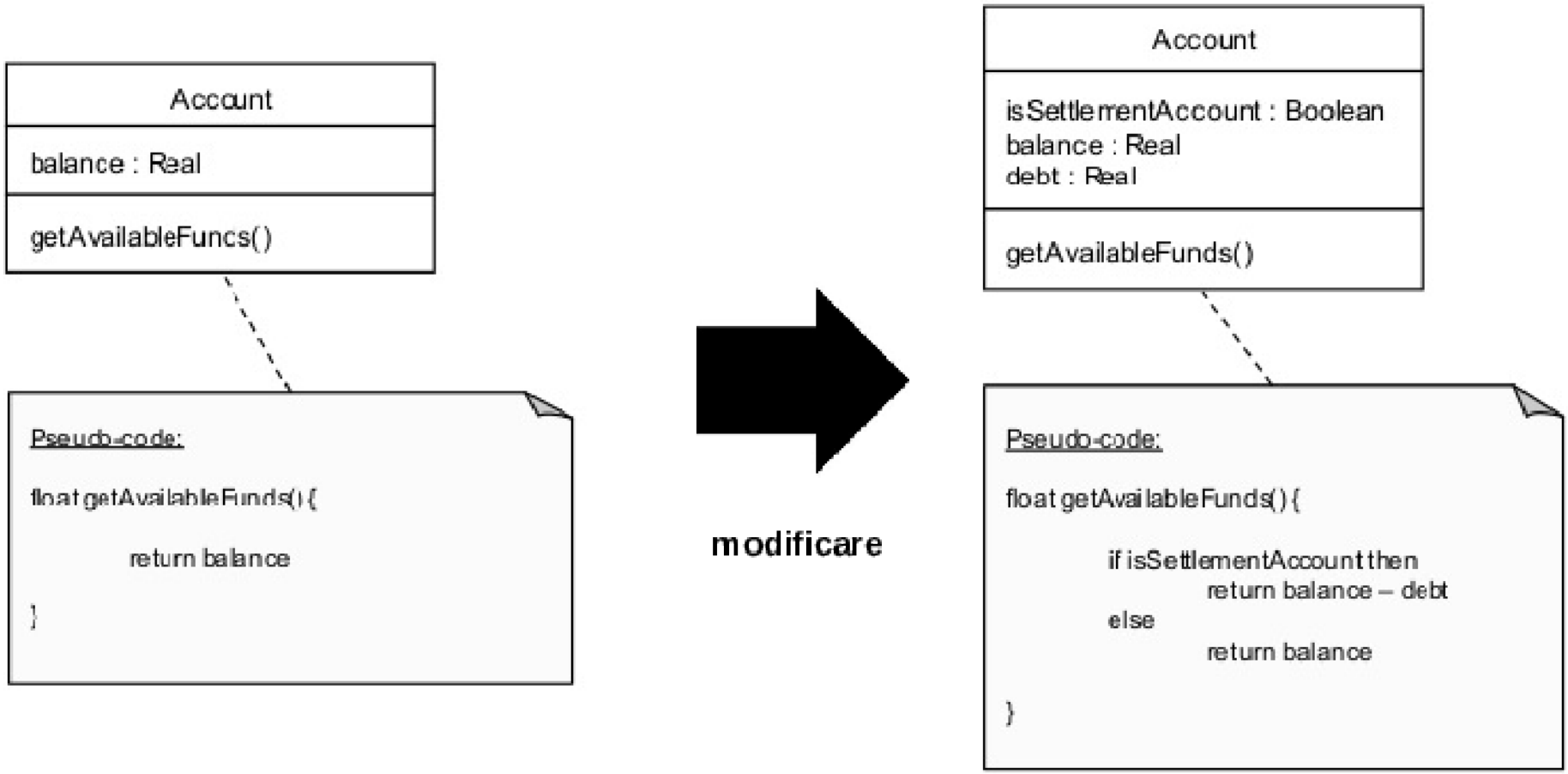
Metrici pentru măsurarea coeziunii

- Clasa are un număr de A - atribute
- Clasa are un număr de M - metode
- Fiecare atribut A_i este accesat de R(A) metode

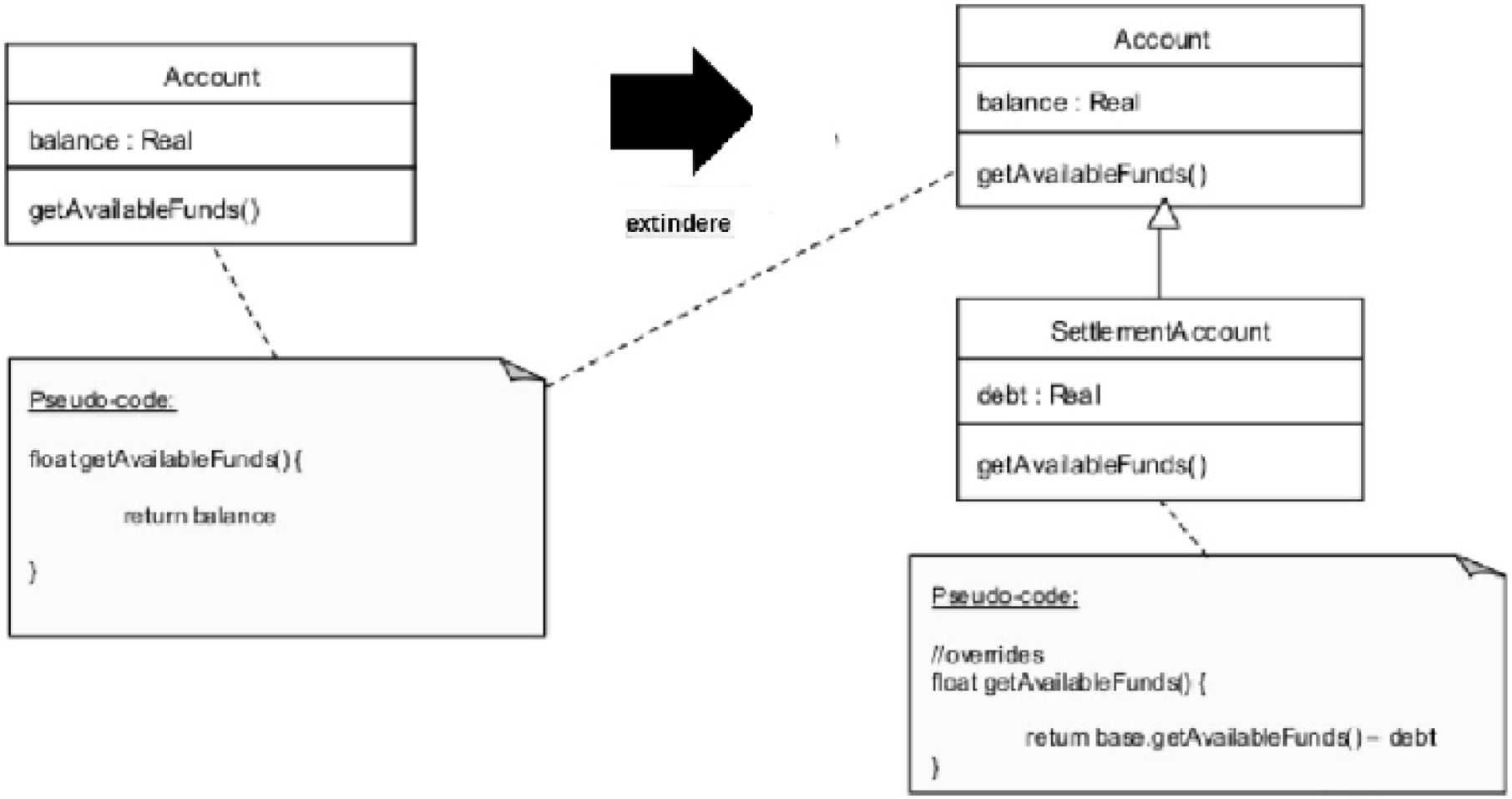


$$LCM = \frac{\frac{\sum_{0}^{A-1} R_i(A)}{A} - M}{1 - M}$$

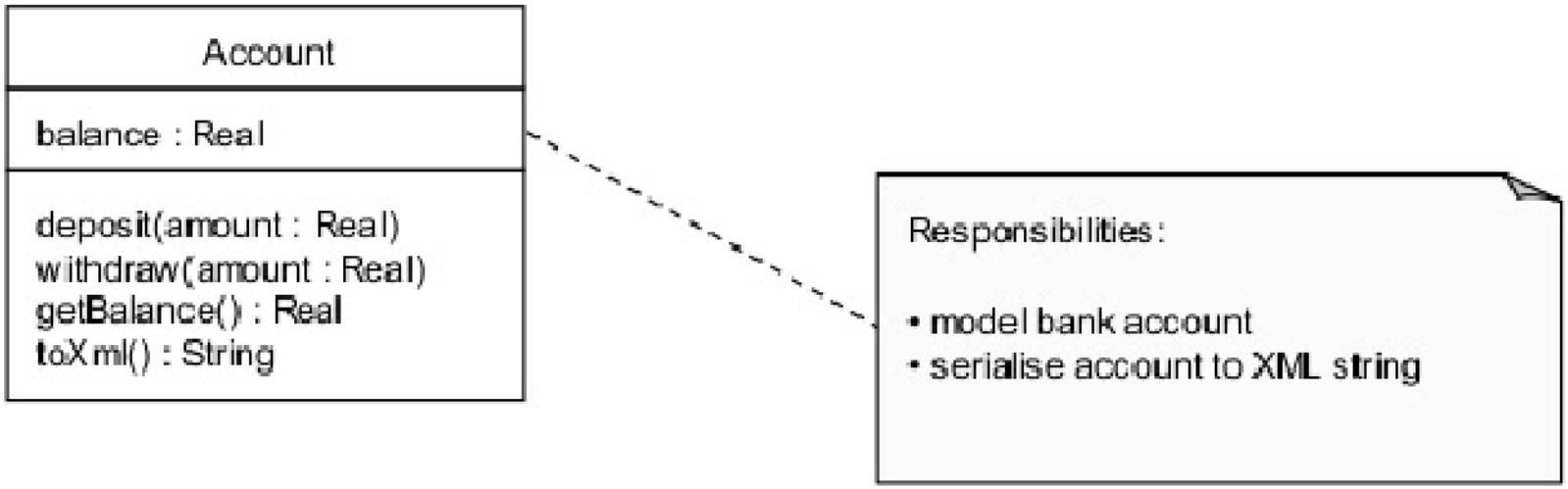
Închisă vs. Deschisă? - înainte ....



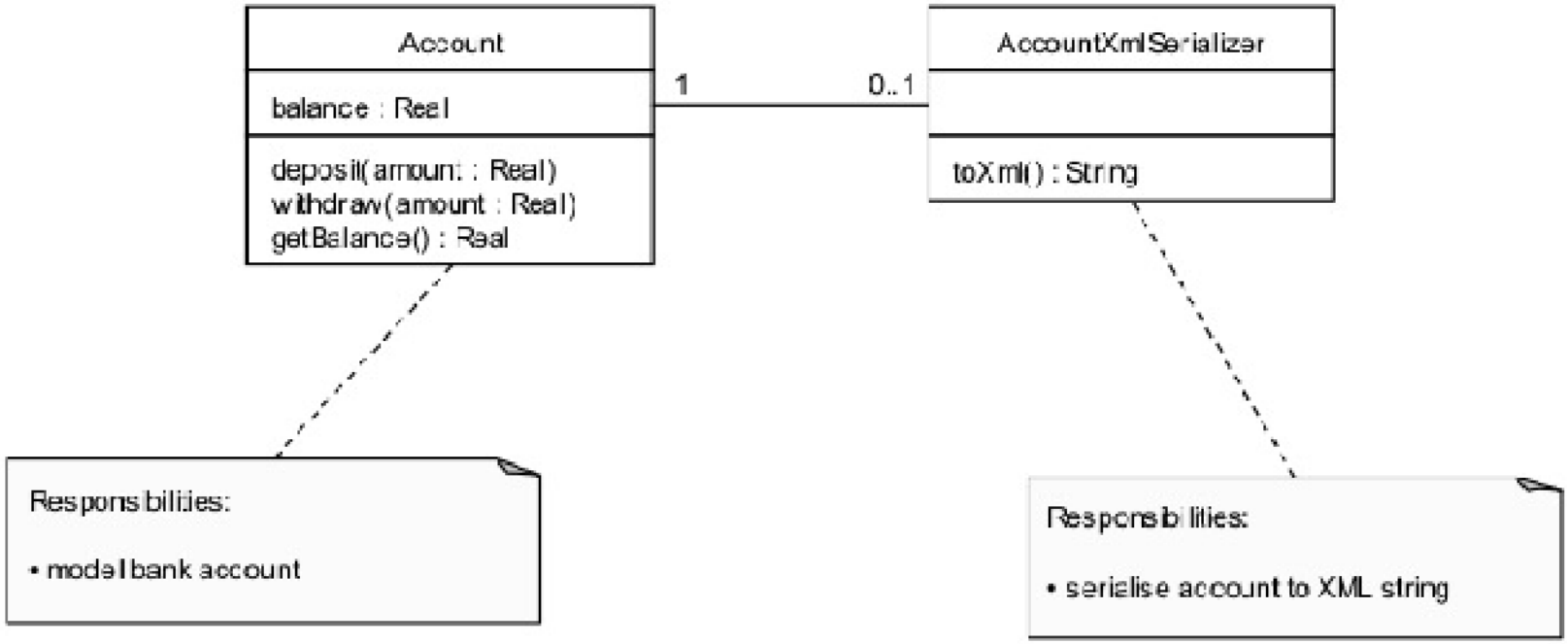
Închisă vs. Deschisă? - după refactorizare



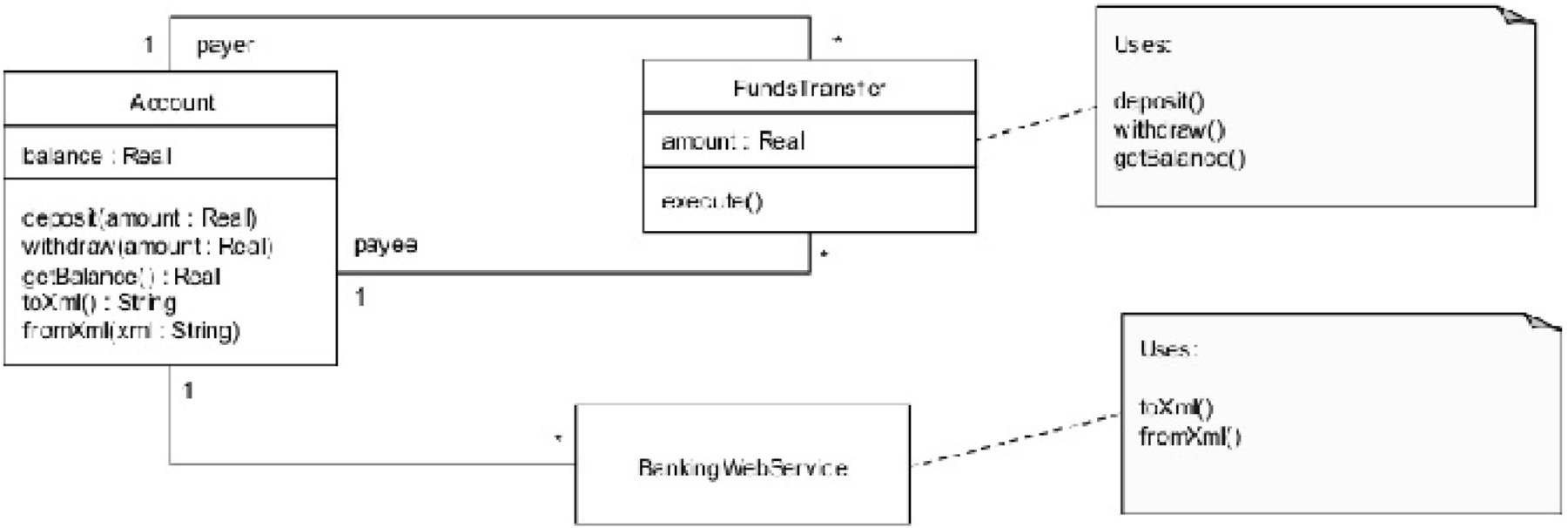
Răspundere Unică - înainte...



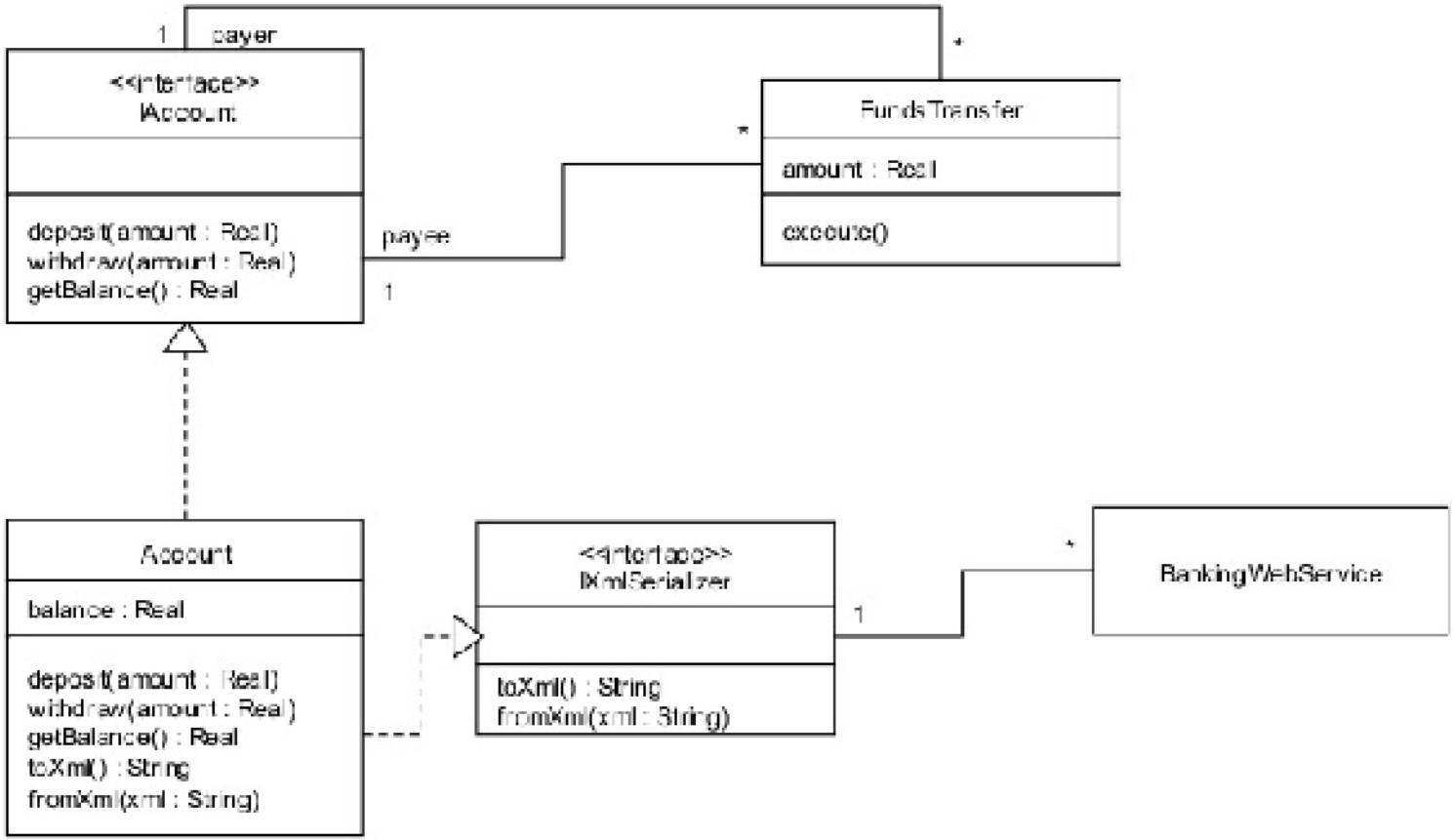
Răspundere Unică - după refactorizare



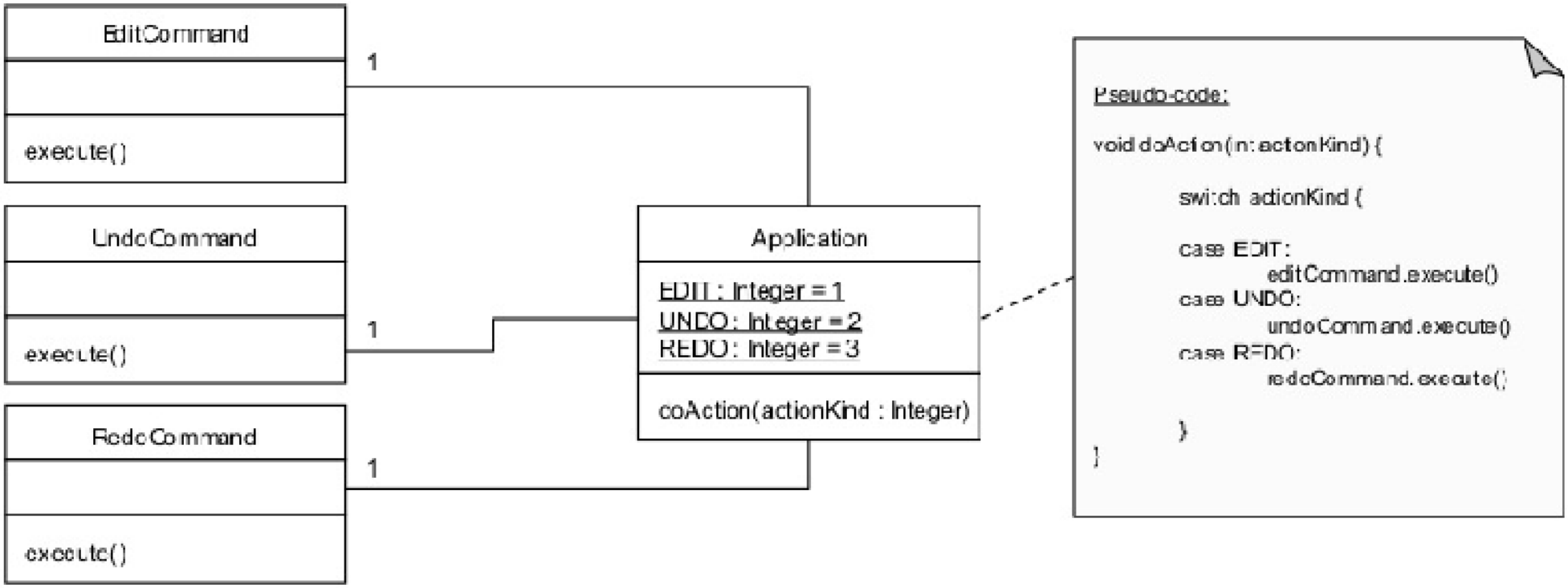
Separarea Interfețelor - înainte....



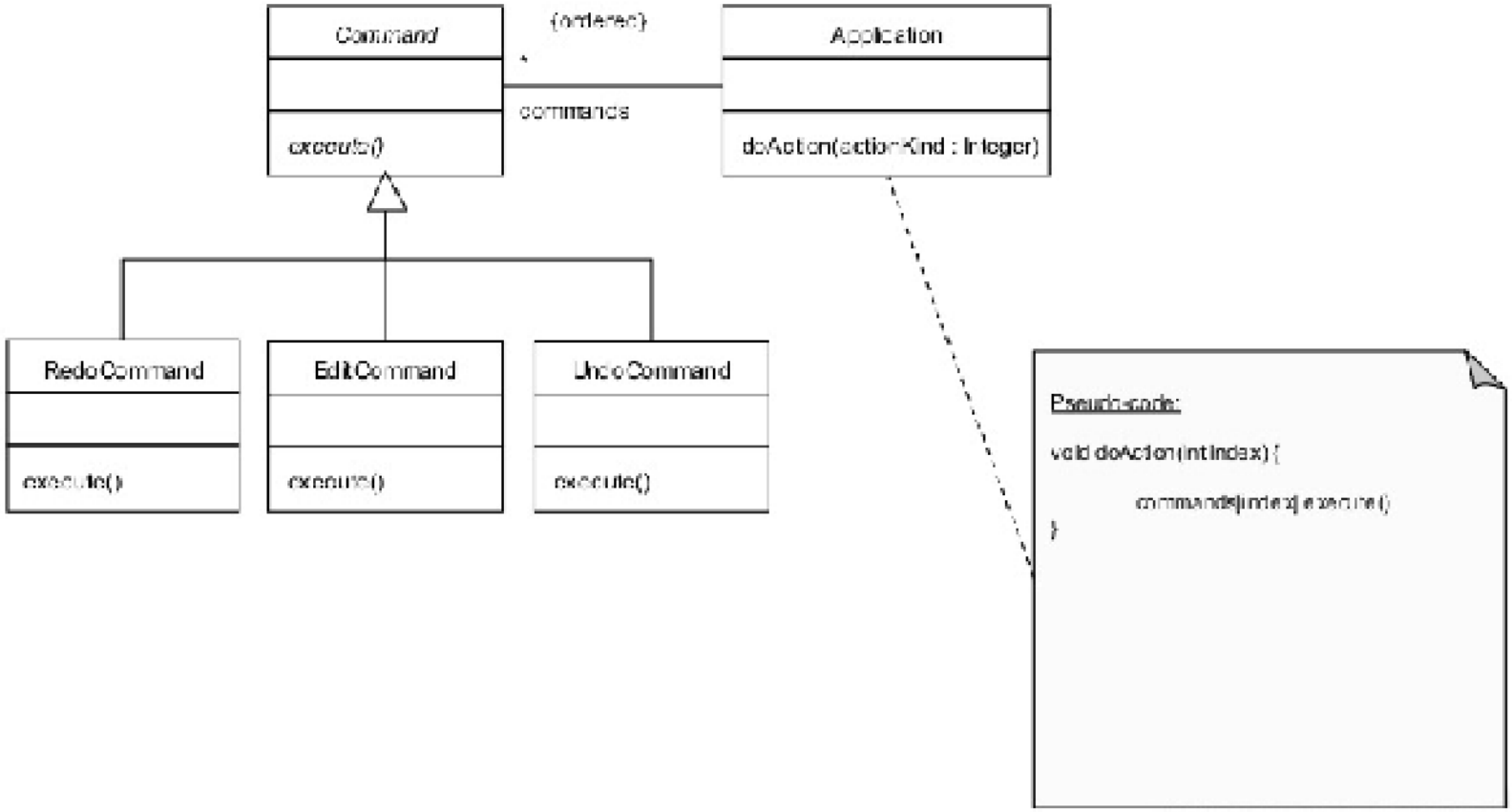
Separarea Interfețelor - după refactorizare



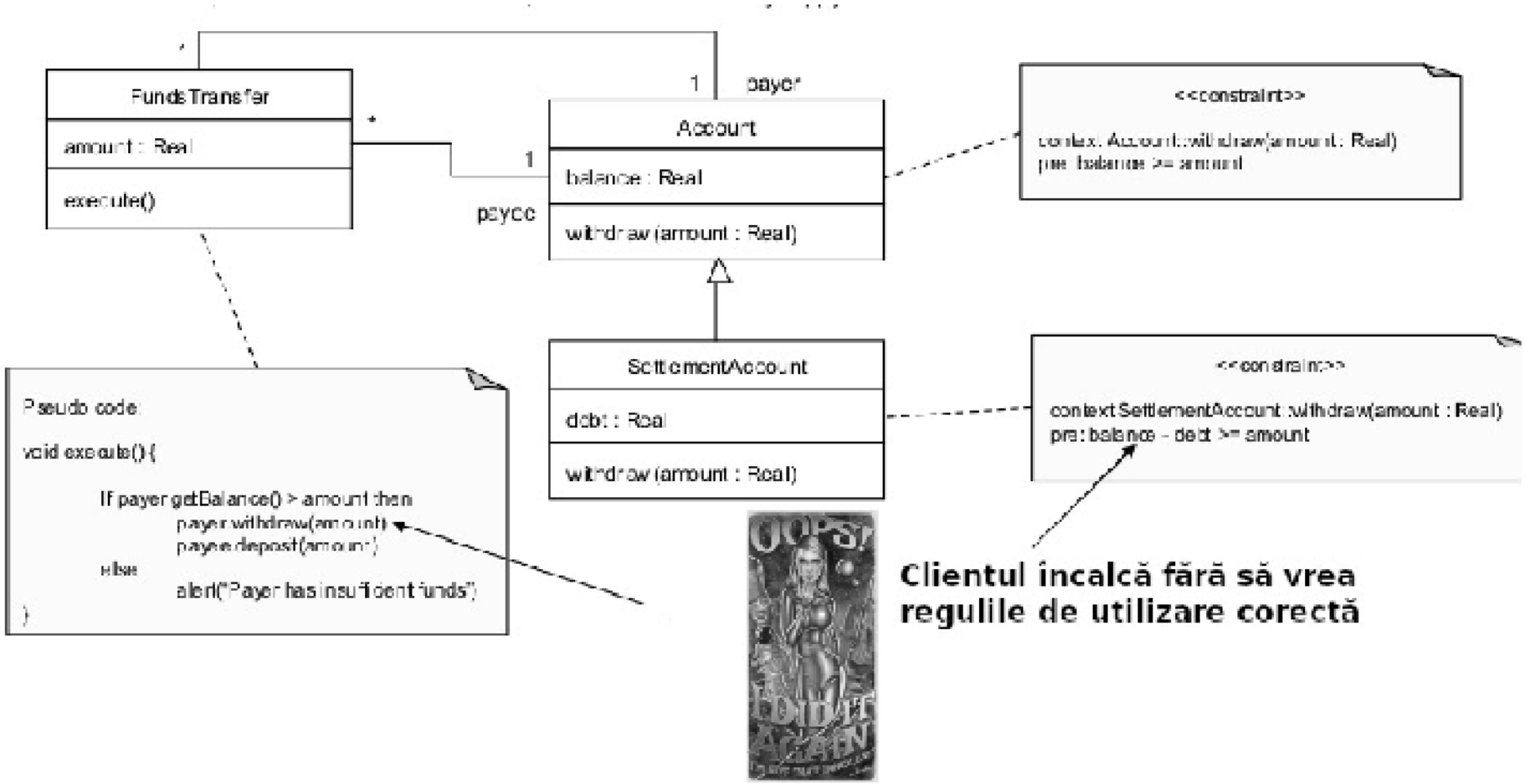
Dependența/Relația Inversă - înainte ...



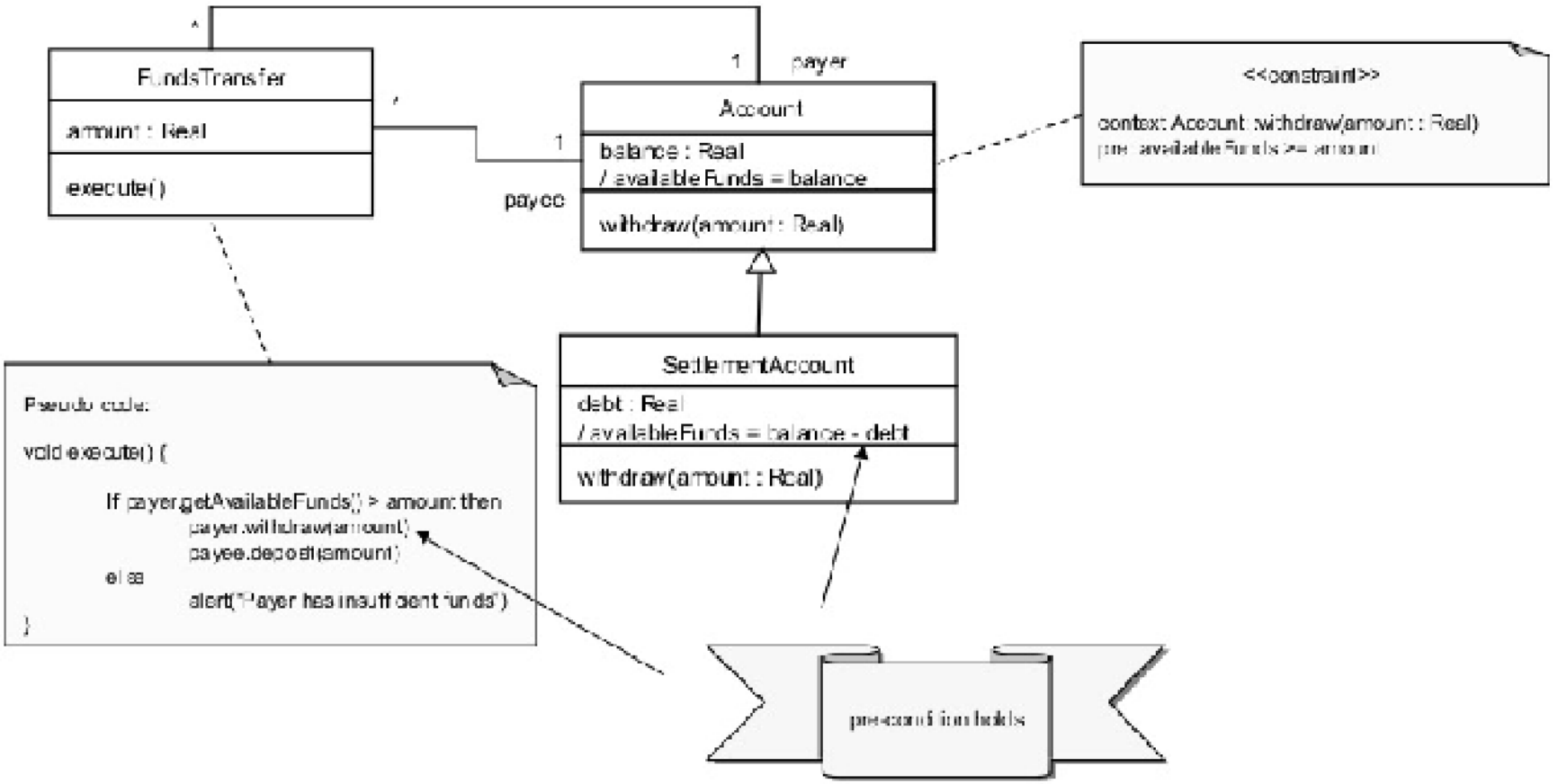
Dependența Inversă - după Factorizare



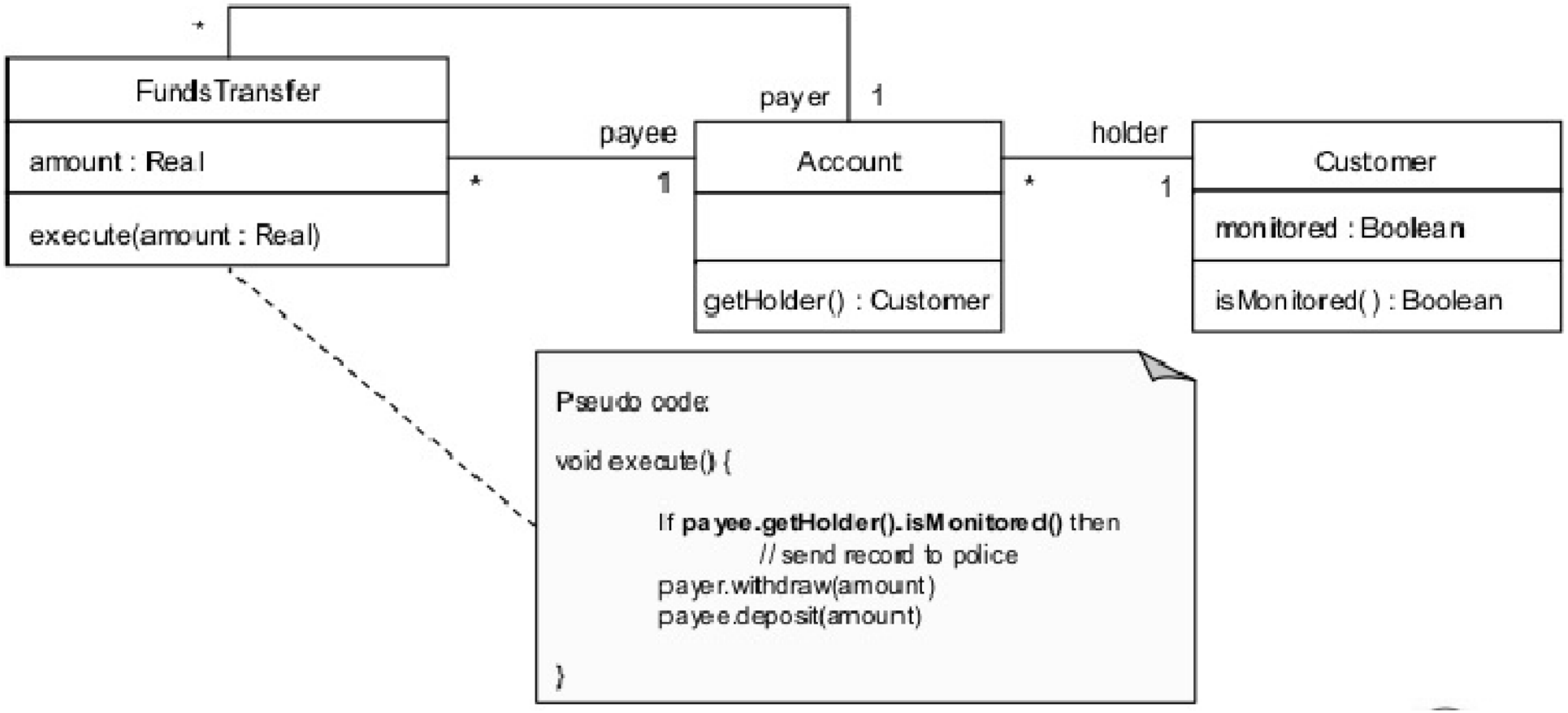
Substituția Liskov - înainte...



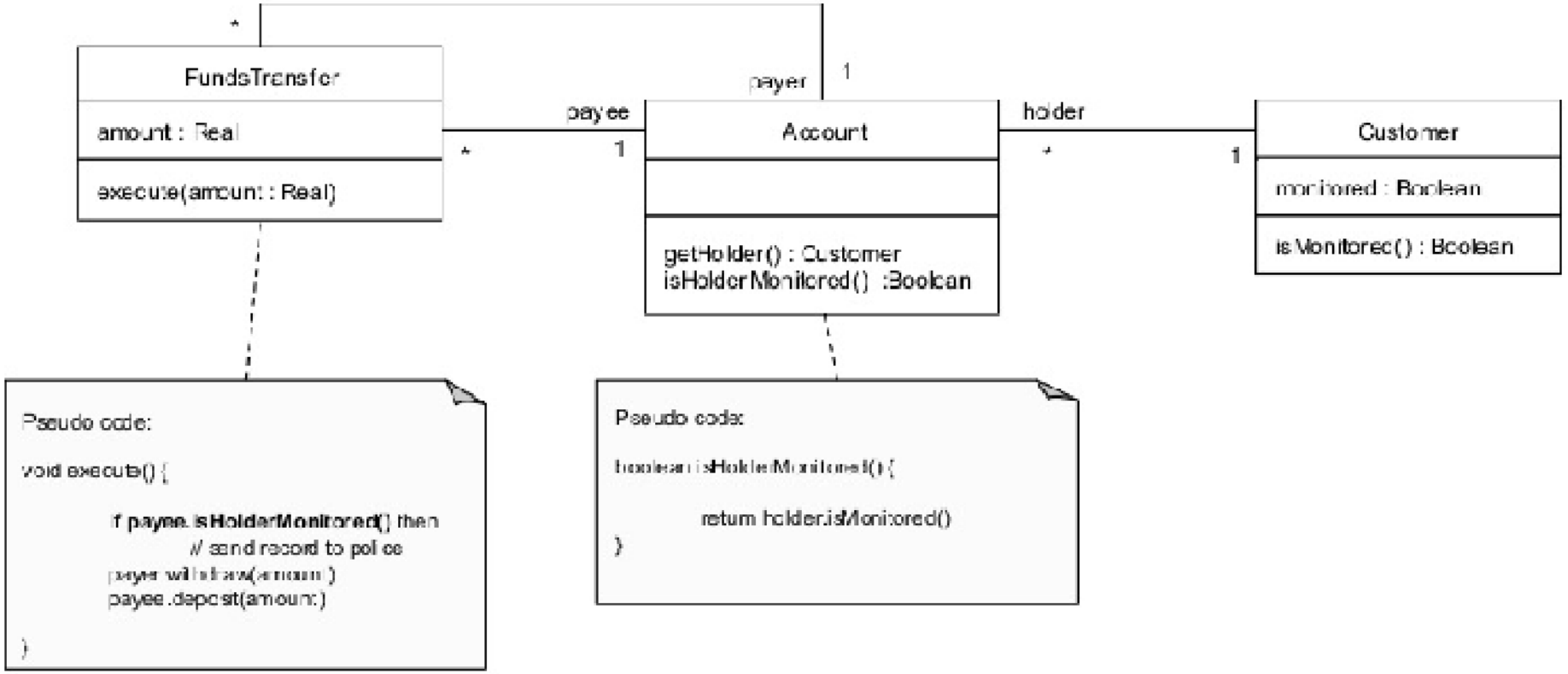
Substituția Liskov - după Refactorizare
un curs cu use case si ce mai este aici



Legea lui Demeter - înainte...

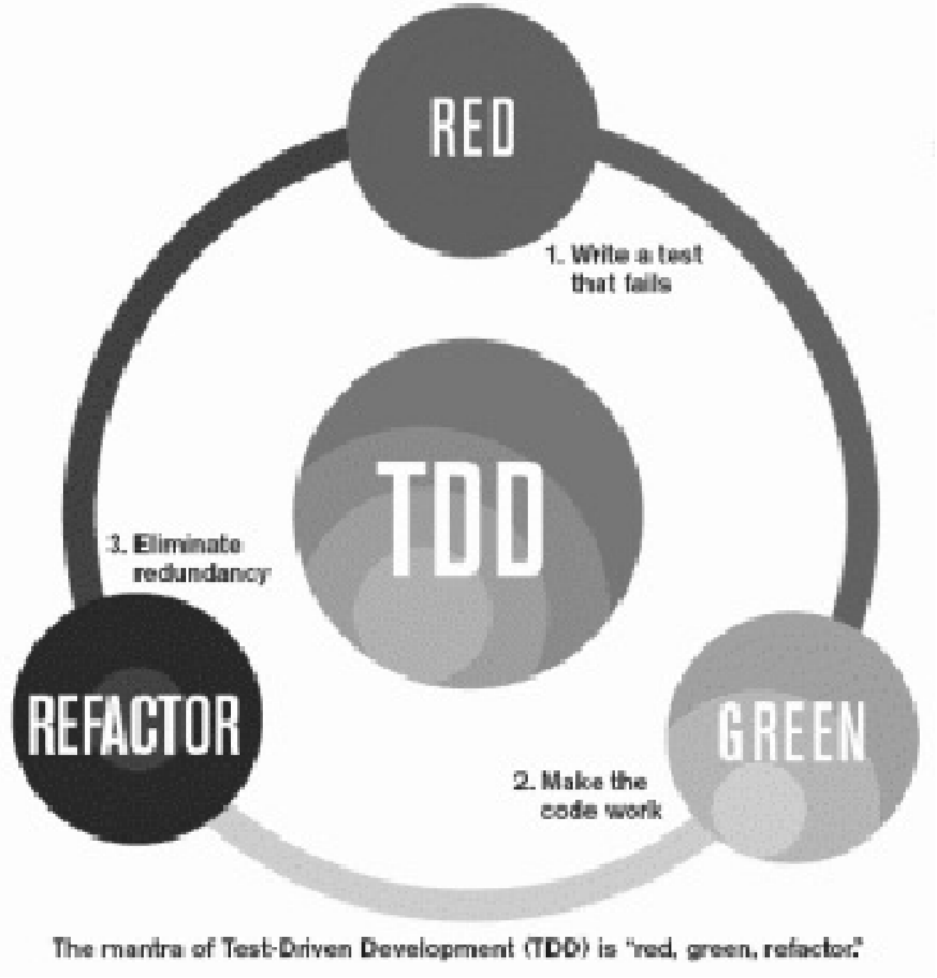


Legea lui Demeter - după Refactorizare



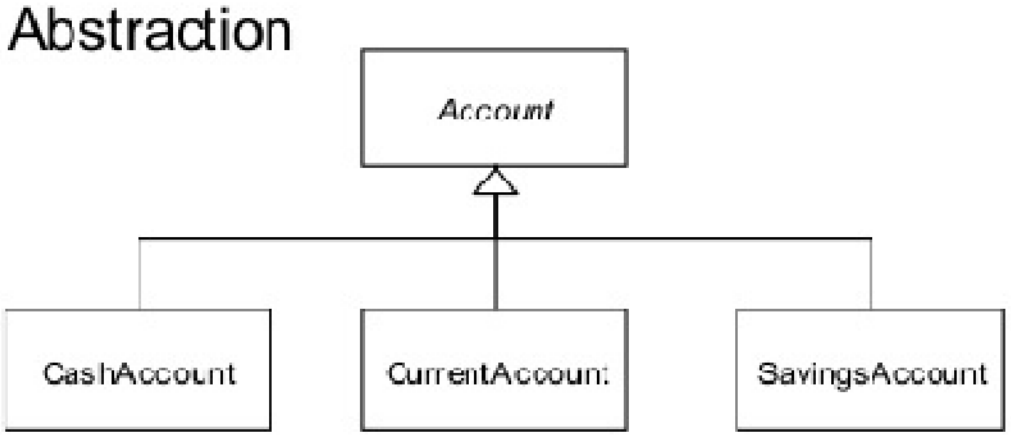
Test-Driven Development (TDD)

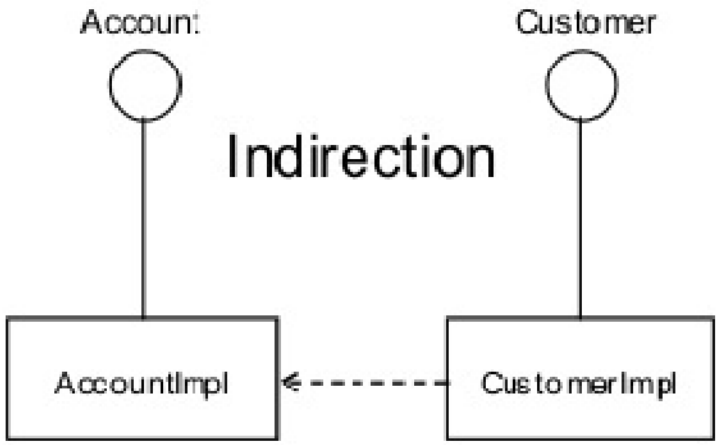
```
while(!OutOfBeer())
{
scrie(un test)
execută(toate testele)
scrie(codul țintă)
execută(testele)
refactorizeaza(codul)
}
```



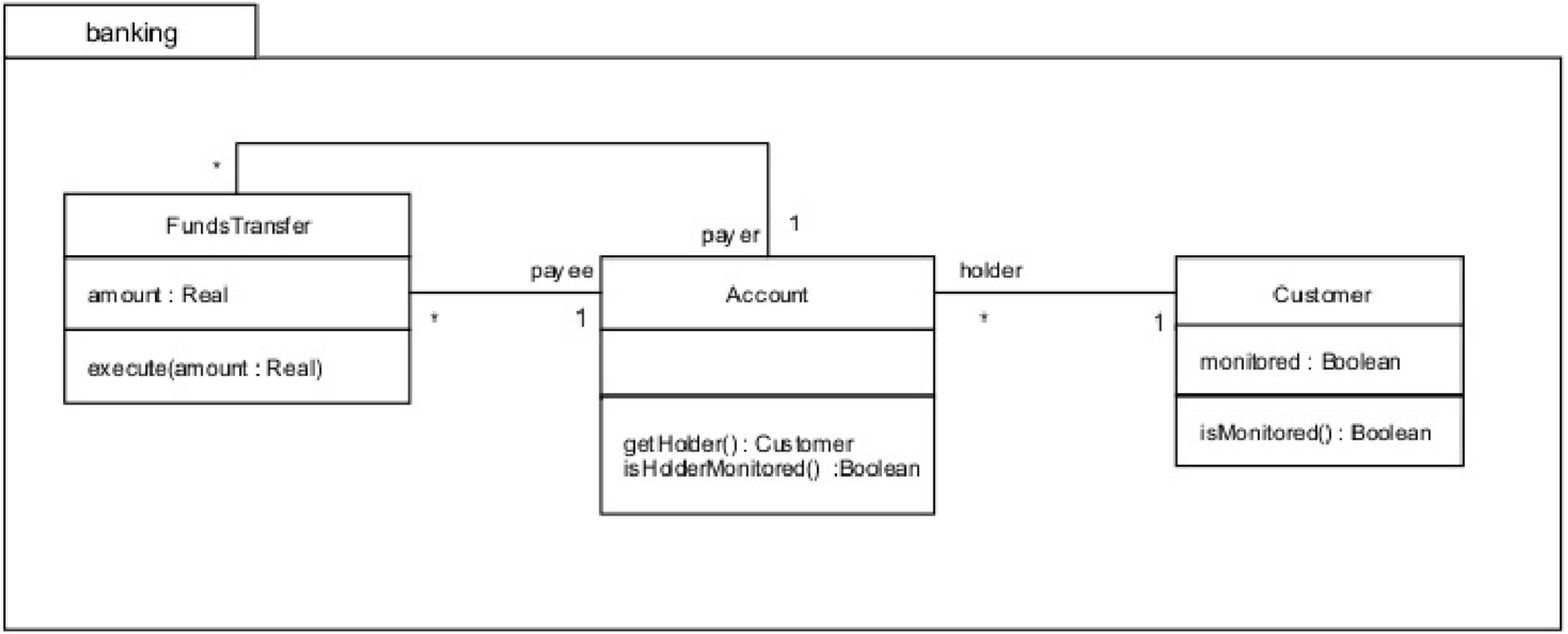
The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Reutilizarea Abstracțiilor

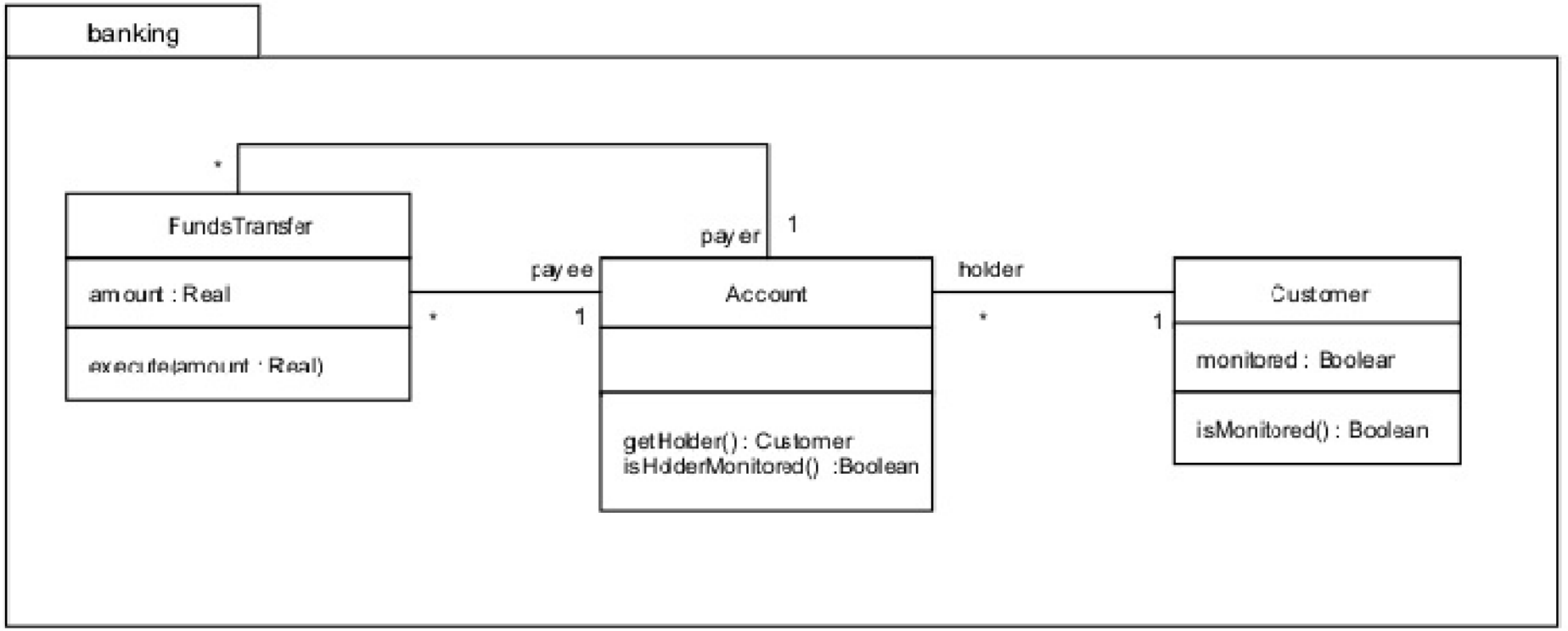




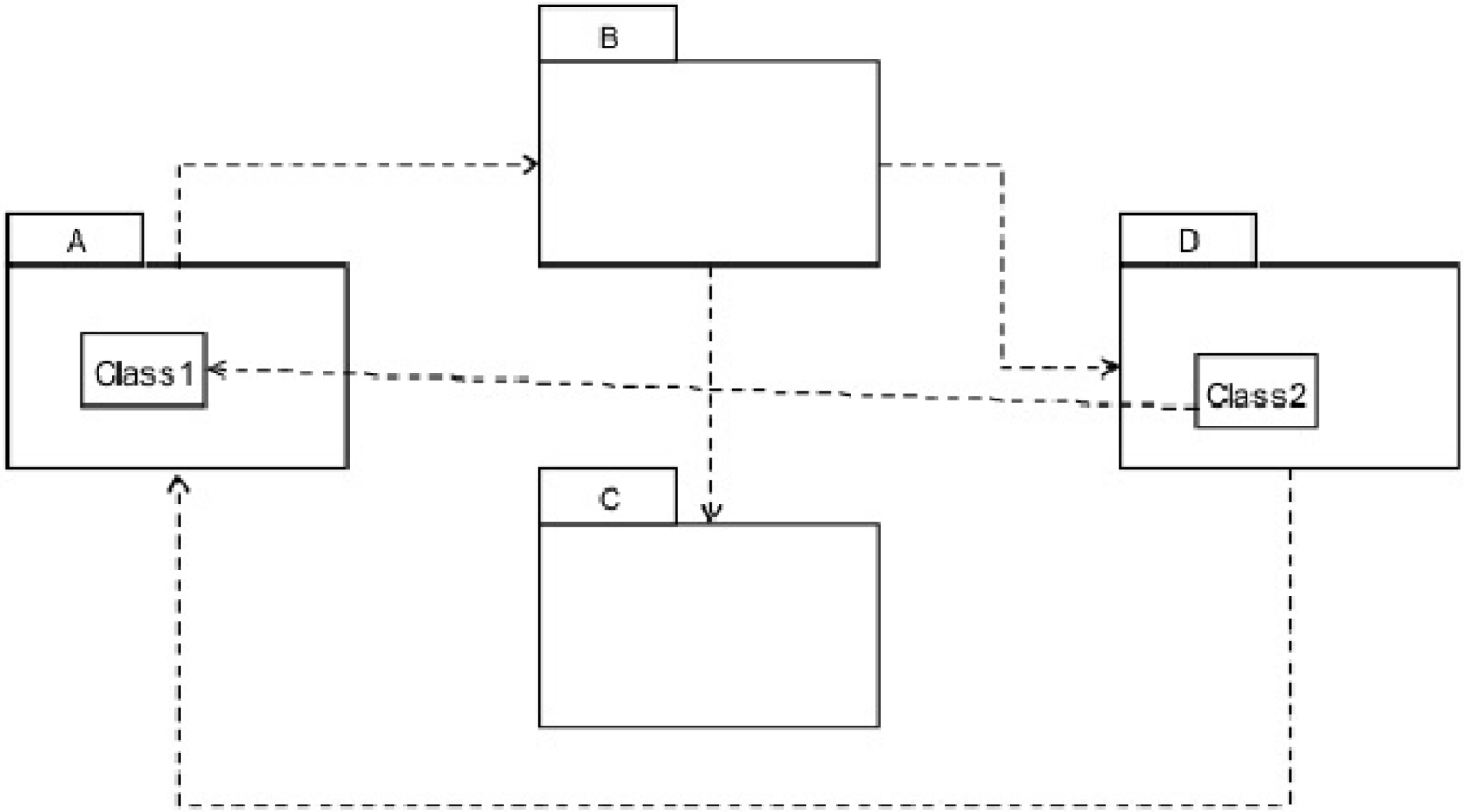
Common Closure



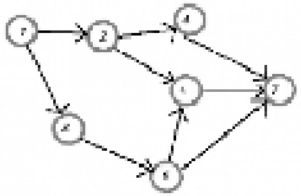
Common Reuse



Dependența aciclică



Dependenta aciclică - Refactorizare



DGA