Analiza și proiectarea sistemelor software

Curs 3

PLAN CURS

- Ingineria proiectării
- Proiectare în contextul ingineriei software
- Procesul de proiectare
- Calitatea proiectării
- Conceptele proiectării
- Principii ale proiectării OO (SOLID)

INGINERIA PROIECTĂRII

Def. Ingineria proiectării = setul de *principii*, *concepte* şi *practici* care conduc la proiectarea unui sistem sau produs de calitate superioară.

***Procesul* de proiectare** conduce la crearea de *reprezentări* diferite ale software-lui care ghidează activitatea de construire (codificare) a acestuia.

Modelul proiect (design) = setul modelelor rezultat al procesului de proiectare :

- Baza tuturor activităţilor ulterioare: evaluări, codificare, testare.

- Evaluat în raport cu cerinţele de calitate; îmbunătăţit înainte de generarea codului.

INGINERIA PROIECTĂRII

Proiectarea (procesul de proiectare):

- activitate creativă:
- crează o reprezentare (un model) a software-lui care oferă detalii despre
  - structurile de date,
  - arhitectură,
  - interfeţe
  - componentele necesare implementării sistemului.
- combină: cerinţele utilizator, nevoile business şi consideraţiile tehnice

Scop:

Producerea unui model care să expună următoarele calităţi ale software-lui ce va fi produs:

- corectitudine
- conformitate cu cerinţele
- uşurinţă în utilizare

INGINERIA PROIECTĂRII

Etapele proiectării:

- Reprezentarea arhitecturii sistemului.
- Definirea interfeţelor cu utilizatorii finali, cu alte sisteme şi cu dispozitive externe.
- Definirea componentelor constitutive şi a relaţiilor dintre acestea.
- Proiectarea componentelor constitutive.

Rezultat: **model** ce cuprinde **reprezentări** ale:

- arhitecturii (date şi funcţionalitate)
- interfeţelor externe
- componentelor şi interfeţelor interne
- instalării componentelor

Evaluare model:

- Detectare erori, inconsistenţe, omisiuni.
- Comparare variante multiple şi alegerea celei optime.
- Fezabilitate implementare în cadrul unor restricţii de timp şi cost.

INGINERIA PROIECTĂRII

Mod de realizare, de către inginerul software:

1. **Diversificare**:
- determinarea unui *repertoriu de alternative*
- obţinerea materiei prime: componente, soluţii pentru componente, cunoştinţe (utilizând cataloage, cărţi, experienţă anterioară)

2. **Convergenţă**:
- *alegerea soluţiei* pe baza cerinţelor utilizator şi a modelului analiză.
- convergenţă către o anumită configurare a componentelor în crearea produsului final.

INGINERIA PROIECTĂRII

Calităţi necesare inginerului software:

- intuiţie
- raţionament

Baza:

- *experienţă* în construirea de entităţi similare
- set de *principii* pentru ghidarea evoluţiei modelului
- set de *criterii* de evaluare a calităţii
- proces *iterativ* de proiectare

Sisteme software complexe

Software industrial – inerent complex, deseori depășind capacitatea intelectuală a unui singur om.

Exemple:

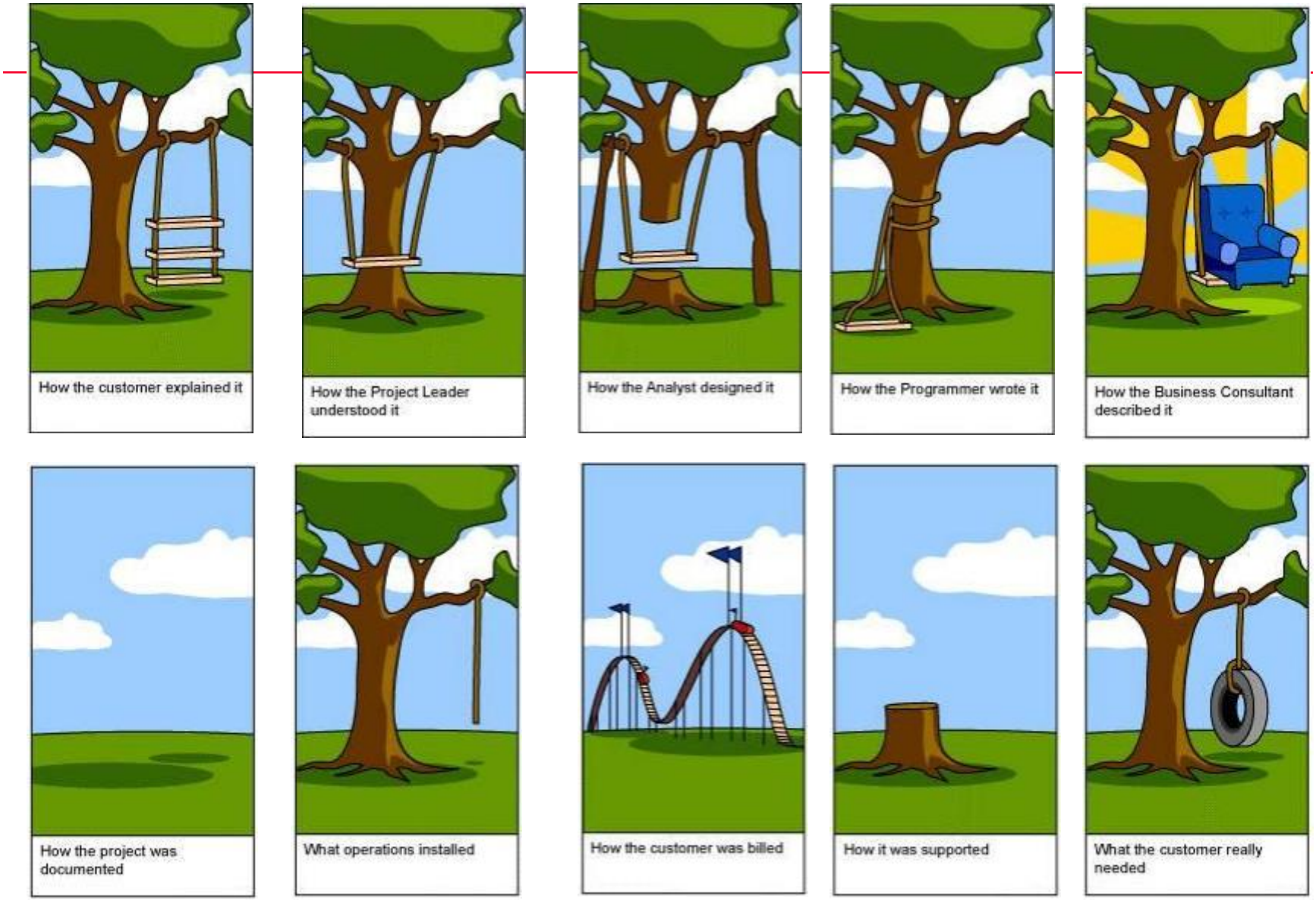
- Sisteme reactive conduse de evenimente, cu resurse limitate de timp şi spaţiu.
- Menţinerea integrităţii a sute de mii de înregistrări de informaţii la care se permit actualizări şi interogări concurente.
- Sisteme pentru comanda şi controlul traficului aerian
- Framework-uri pentru creare de aplicaţii specifice unor anumite domenii.
- Aplicaţii de inteligenţă artificială.

Sisteme software complexe

Complexitatea domeniului problemei:

- Funcţii complexe
- Cerinţe funcţionale şi extra-funcţionale concurente, uneori contradictorii
- Cerinţe în schimbare

Probleme de clarificare a cerinţelor



Sisteme software complexe

Probleme de clarificare a cerinţelor

Dificultatea administrării procesului de dezvoltare

Flexibilitatea software-lui

Problemele de caracterizare a comportamentului sistemelor discrete ca răspuns la evenimente externe şi interne (imposibilitatea testării exhaustive).

Atributele sistemelor complexe (1)

1. Structură ierarhică

Descompunere sistem în subsisteme.

Considerarea relaţiilor dintre subsisteme.

Un sistem complex este definit de *componentele* sale şi de *relaţiile* dintre aceste componente.

2. **Relativitatea considerării componentelor primitive**

Nivele de abstractizare multiple

Aceeaşi componentă poate fi considerată primitivă de către un observator şi ca fiind pe un nivel superior de abstractizare de către alt observator.

Atributele sistemelor complexe (2)

3. **Separarea problemelor**

Legături puternice intra-componentă

Legături slabe inter-componente

Relativă izolare a componentelor.

Atributele sistemelor complexe (3)

4. **Tipare (patterns) comune**

Tipare (șabloane) comune folosite în diverse combinări şi aranjamente.

5. **Forme intermediare stabile**

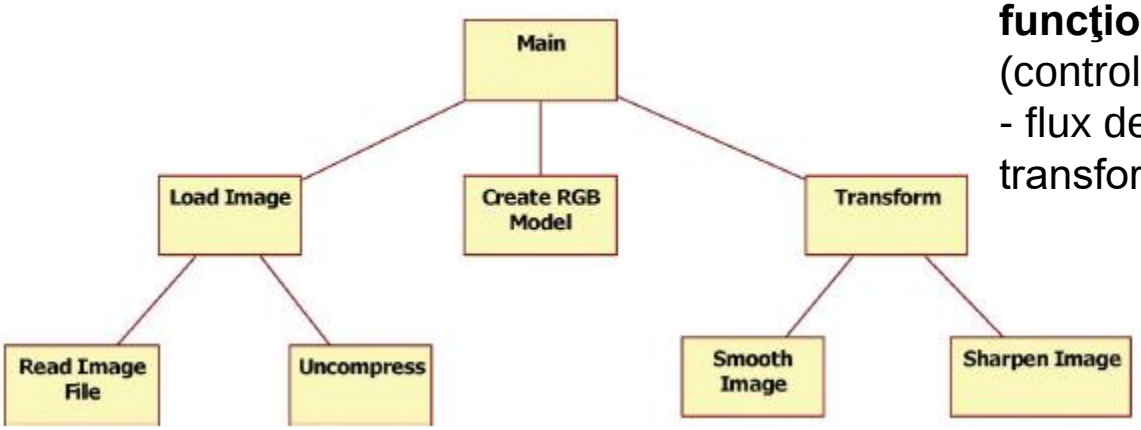
Sistemele evoluează de la simplu la complex, trecând prin forme intermediare stabile.

Obiecte considerate iniţial complexe devin primitivele din care se construieşte sistemul complex.

Obs. Primitivele pot fi validate doar după utilizarea lor în contextul sistemului. Ele vor evolua pe măsură ce sistemul se construieşte.

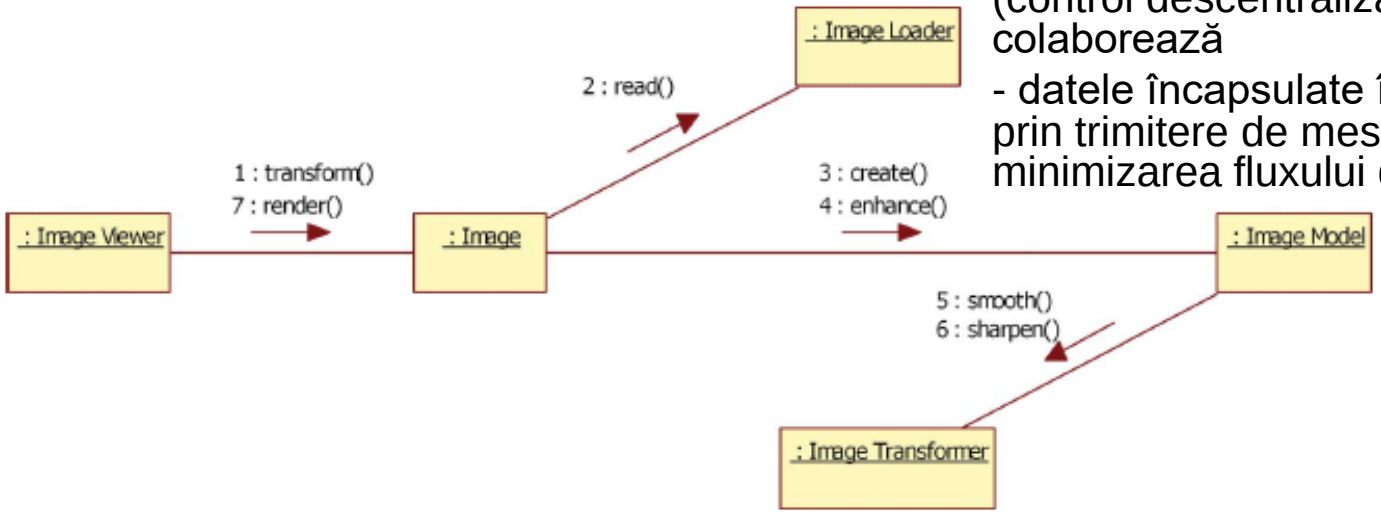
Gestionarea complexităţii

Descompunere



Funcţională (algoritmică) – elementele funcţionale şi relaţiile dintre acestea.
(control centralizat – ierarhie de funcţii
- flux de date supuse unei succesiuni de transformări).

Perspective ortogonale



OO – conform abstractizărilor cheie din domeniul problemei.
(control descentralizat – reţea de obiecte ce colaborează
- datele încapsulate în obiecte şi manipulate prin trimitere de mesaje la aceste obiecte ⇒ minimizarea fluxului datelor prin sistem).

Model

- Model – descrie un anumit aspect al sistemului
- Construire model - descompunere, abstractizare şi ierarhie.
- Evaluare model :
  - În condiţii tipice
  - În condiţii neaşteptate
- Modificare corespunzător rezultatelor evaluării

Metodă şi metodologie

- Metodă (practică) de proiectare– *procedură disciplinată* pentru generarea unui *set de modele* ce descriu *diferite aspecte* ale unui sistem software în curs de dezvoltare folosind o *notaţie* bine definită.

- Metodologie – *colecţie de metode* aplicate de-a lungul ciclului de dezvoltare de software şi *unificate* prin *proces*, *practici* şi o *abordare* (filozofie) generală.

Metode pentru proiectare software

Elemente comune:

- **Notaţie**: limbajul pentru exprimarea fiecărui model
- **Proces**: activităţile ce conduc la construirea ordonată a modelelor sistemului
- **Instrumente**: artefactele utilizate în construirea modelului (maschează rutina, impun respectarea de reguli de modelare, detectează erori şi inconsistenţe).
- Procesul de proiectare *nu este deterministic*: diferiţi proiectanţi pot produce modele diferite pentru soluţia aceleiaşi probleme.
- O metodologie completă de proiectare se bazează pe un *fundament teoretic solid*, dar oferă *grade de libertate* pentru inovaţie.

Metodă

Categorii de metode:

- Proiectare *structurată descendentă* – structura sistemului bazată pe descompunere algoritmică
- Proiectare *condusă de date* – structura sistemului bazată pe corespondenţa intrare – ieşire
- Proiectare *orientată obiect* – structura sistemului bazată pe colecţii de obiecte cooperante

Evaluare formativă (1)

1. Enumerați etapele procesului de proiectare software.

2. Ce se urmărește la evaluarea modelelor ?

3. În procesul de proiectare software, ce rezultate are separarea problemelor ?

4. Care sunt elementele de care are nevoie orice metodologie pentru proiectarea software-lui ?

https://forms.gle/1YJFcfdsCZU8KWBM8

PLAN CURS

- Ingineria proiectării
- Proiectare în contextul ingineriei software
- Procesul de proiectare
- Calitatea proiectării
- Conceptele proiectării
- Principii ale proiectării OO (SOLID)

PROIECTARE ÎN CONTEXTUL INGINERIEI SOFTWARE

Nucleul tehnic al ingineriei software (indiferent de modelul procesului de dezvoltare de software).

Ultima etapă a modelării.

Definitivează baza pentru generarea şi testarea codului.

Ordinea de lansare a activităţilor fundamentale:

- definirea *interfeţei* cu elementele din context
- definirea *datelor*
- definirea *arhitecturii – componentele* și *interfețele* dintre acestea
- definirea *elementelor componente*

PROIECTARE ÎN CONTEXTUL INGINERIEI SOFTWARE

Proiectare date/clase

- Transformă clasele din modelul analiză în *realizările claselor* (de la la nivelul design-lui) şi în *structurile de date* necesare implementării.
- Se introduc noi detalii odată cu proiectarea componentelor.

Proiectare arhitectură – defineşte un cadru pentru sistem:

- *elementele* structurale majore ale software-lui şi relaţiile dintre ele
- *stilurile* arhitecturale şi şabloanele utilizate pentru îndeplinirea cerinţelor sistem,
- *constrângerile* ce afectează modul în care arhitectura va fi implementată.

Proiectare interfaţă – defineşte comunicarea (sub)sistemului cu utilizatorii şi cu alte (sub) sisteme

- flux informaţional (date şi/sau control).
- tip specific de comportament.

Proiectare componente – proiectarea interfeţelor şi descriere procedurală a componentelor software.

PLAN CURS

- Ingineria proiectării
- Proiectare în contextul ingineriei software
- Procesul de proiectare
- Calitatea proiectării
- Conceptele proiectării
- Principii ale proiectării OO (SOLID)

PROCESUL DE PROIECTARE

Proces ***iterativ*** de transformare a ***cerinţelor*** în ***plan*** de construire a software-lui.

Rafinări succesive ale reprezentărilor.

Reevaluare calitate după fiecare iteraţie.

Criterii de evaluare:

- Implementarea tuturor cerinţelor explicite conţinute în modelul analiză şi conformarea cu toate cerinţele implicite ale clientului.
- Ghid inteligibil pentru programatori şi testeri.
- Imagine completă a software-lui – datele, funcţiile şi comportamentul, din perspectiva implementării.

Modelele primare ale unui sistem software

Fairbanks, G. *Just Enough Software Architecture, A Risk Driven Approach,* Marshall&Brainerd, 2010

Modelul domeniului (modelul analiză)

- adevărurile durabile despre domeniu, relevante pt. sistemul de dezvoltat
- detalii ale domeniului independente de implementarea sistemului
- mod de a înţelege elementele ce vor fi esenţiale în procesul de proiectare
- reprezentat cu notaţii simple şi intuitive pentru o interacţiune eficientă cu experţii domeniului
- baza unui limbaj universal pentru domeniul respectiv (DSL – Domain Specific Language ; Domain Patterns)

Modelul proiect

- descrie sistemul ce va fi construit : decizii de proiectare, mecanisme, etc.
- mai multe nivele de abstractizare
- pentru fiecare componentă:
  - interfaţa publică vizibilă
  - detaliile interne

Modelul codului

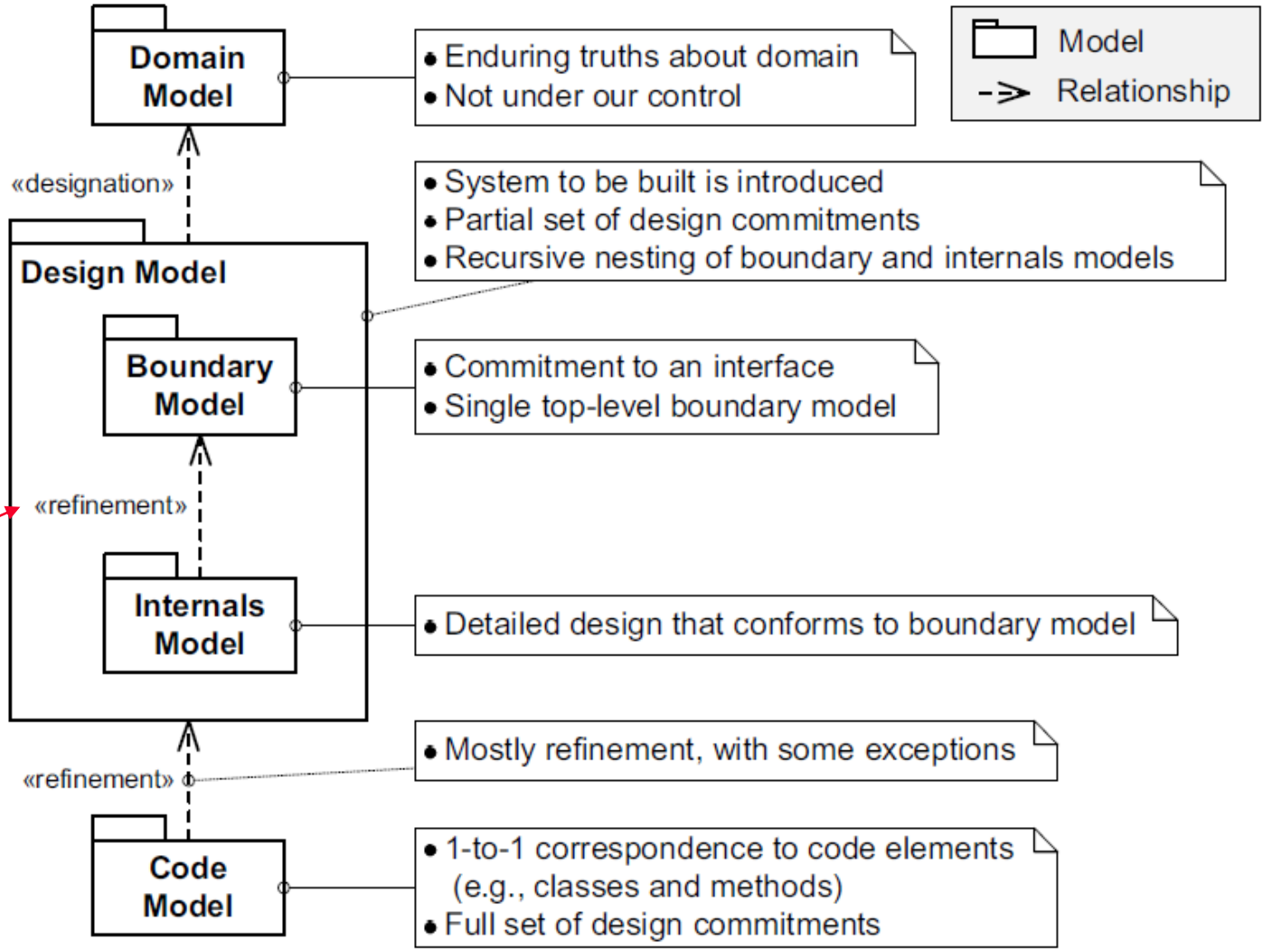
- descrie codul sursă al sistemului cu un model de nivelul limbajului de programare
- suficient pentru a fi executat automat

Structura canonică a modelelelor primare

Fairbanks, G. *Just Enough Software Architecture, A Risk Driven Approach,* Marshall&Brainerd, 2010

Corespondenţă între conceptele din modelul domeniului şi tipurile definite în modelul proiect.

Relaţie între modele ale aceluiaşi element aflate pe nivele de detaliu diferite.



PLAN CURS

- Ingineria proiectării
- Proiectare în contextul ingineriei software
- Procesul de proiectare
- Calitatea proiectării
- Conceptele proiectării
- Principii ale proiectării OO (SOLID)

CALITATEA PROIECTĂRII (1)

Ghid de calitate pentru proiectare:

- Arhitectura:
  - Creată utilizând şabloane sau *stiluri arhitecturale*
  - Componente bine proiectate
  - Implementabilă în *manieră evolutivă* $\Rightarrow$facilitare implementare şi testare. ?
- Modularitate: *partiţionare software* în elemente sau subsisteme.
- *Reprezentări distincte* pentru date, arhitectură, interfeţe şi componente.
- *Structuri de date* pentru clasele ce vor fi implementate, ce corespund unor *şabloane recognoscibile*.

CALITATEA PROIECTĂRII (2)

Ghid de calitate pentru proiectare:

- Componente caracterizate prin *independenţă funcţională*.
- Interfeţe care *reduc complexitatea conexiunilor* dintre componente şi a celor cu mediul extern.
- Condusă utilizând o *metodă repetitive,* pe baza informaţiilor obţinute în cursul analizei cerinţelor software.
- Modelul proiect reprezentat utilizând o *notaţie care comunică eficient semnificaţia*.

CALITATEA PROIECTĂRII

Atributele de calitate: ***FURPS***

Funcţionalitate:
- setul de capabilităţi
- generalitatea funcţiilor oferite
- securitatea sistemului în ansamblu

Utilizabilitate:
- estetică
- consistenţă
- documentare

Fiabilitate (***R***eliability):
- frecvenţa şi severitatea defectelor
- acurateţea rezultatelor la ieşire
- timpul mediu între defectări (MTTF)
- abilitatea de recuperare din defect
- predictibilitatea programului

Performanţă:
- viteza de procesare/timpul de răspuns
- consumul de resurse
- volumul procesărilor (throughput)
- eficienţa

Suportabilitate:
- mentenabilitate (extensibilitate, adaptabilitate, serviabilitate)
- testabilitate
- configurabilitate
- instalare simplă
- localizare rapidă a problemelor

CALITATEA PROIECTĂRII

Atributele de calitate: ***FURPS***

Accentul ponderat, funcţie de aplicaţie.

Ex.

- Funcţionalitate, cu accent pe securitate.
- Performanţă, cu accent pe viteza de procesare.
- Fiabilitate.

Trebuie avute în vedere de la începutul procesului de proiectare !

PLAN CURS

- Ingineria proiectării
- Proiectare în contextul ingineriei software
- Procesul de proiectare
- Calitatea proiectării
- Conceptele proiectării
- Principii ale proiectării OO (SOLID)

CONCEPTELE PROIECTĂRII

- Abstractizare
- Arhitectura
- Șabloane
- Separare probleme
- Modularitate
- Încapsulare (ascunderea informaţiilor)
- Independenţă funcţională
- Rafinare
- Aspecte
- Refactorizare

CONCEPTELE PROIECTĂRII

ABSTRACTIZARE = Extragerea, într-un *model generalizat/idealizat*, a *aspectelor esenţiale* funcţie de perspectiva de abordare a problemei de rezolvat.

Principiu de proiectare: Utilizarea de *nivele multiple de abstractizare* în exprimarea soluţiei.

- Nivelul superior: termeni largi, limbaj specific contextului problemei.
- Nivele inferioare: descriere în detaliu a soluţiei; limbaje tehnice.

Scop: crearea de abstractizări:

- procedurale,
- ale datelor.

Abstractizare *procedurală* = separarea *proprietăţilor logice* ale unei *acţiuni* de *detaliile de implementare* ale acesteia.

⇓

Secvenţă de operaţii ce au o funcţie specifică şi limitată.

Abstractizare *date* = separarea *proprietăţilor logice* ale *datelor* de *detaliile de reprezentare* a datelor.

⇓

Colecţie de date care descriu un obiect informaţional.

CONCEPTELE PROIECTĂRII

ARHITECTURĂ

Arhitectură software : *structura* de ansamblu a software-lui şi modalităţile în care aceasta oferă integritatea conceptuală pentru un sistem.

- Structura şi organizarea elementelor sistemului software (unităţilor de cod, unităţilor de execuţie).
- Modalităţile de interacţiune a acestor elemente (cu contextul sistemului şi între ele).
- Structura datelor utilizate de sistem.

Entităţile majore ale sistemului şi interacţiunile dintre acestea.

Cadru, bazat pe şabloane, pentru conducerea activităţilor ulterioare de proiectare.

CONCEPTELE PROIECTĂRII

ŞABLOANE (PATTERNS)

Şablon de proiectare: descrie o ***structură de proiectare*** ce rezolvă o anumită ***problemă de proiectare*** în cadrul unui ***context specific*** şi în mijlocul unor "***forţe***" care pot avea impact asupra manierei în care este aplicat şi utilizat şablonul.

Oferă o ***descriere a soluţiei***.

Din descriere se pot deduce:

- aplicabilitatea şablonului la problema curentă,
- reutilizabilitatea şablonului,
- dacă şablonul poate fi folosit ca ghid pentru dezvoltarea unui şablon similar, dar diferit din punct de vedere funcţional sau structural.

CONCEPTELE PROIECTĂRII

SEPARARE PROBLEME (*separation of concerns*)

Divizare problemă în subprobleme ce pot fi *rezolvate independent*.

Problemă (*concern*) = caracteristică sau comportament specificat în modelul cerințelor.

Separarea problemelor este în relație cu conceptele de modularitate, aspecte, independență funcțională și rafinare.

CONCEPTELE PROIECTĂRII

MODULARITATE - Atribut esenţial în gestionarea sistemelor software complexe.

Divizare software în componente separate: *identificate* prin nume, *adresabile*, *integrate* pentru rezolvarea unei probleme.

Discuţie:

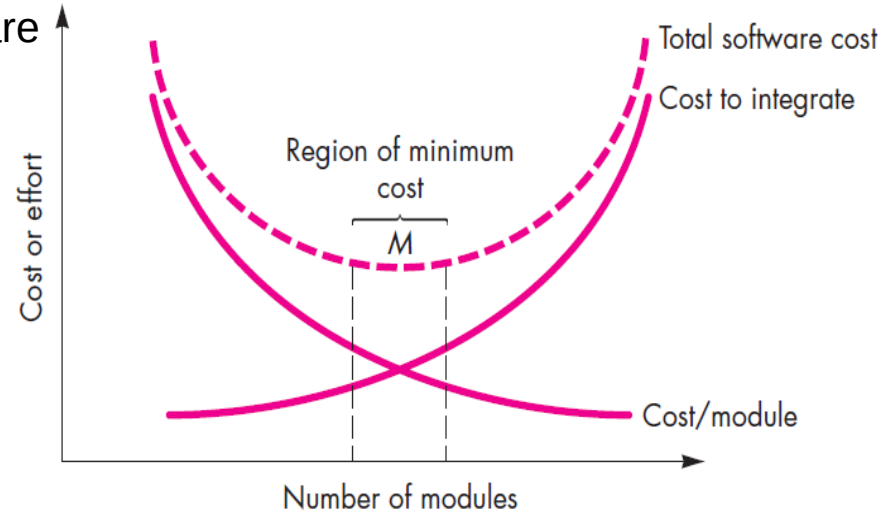
Creştere număr module => scădere cost per modul & creştere cost pentru integrare module

Criterii identificare *nivel optim* de modularizare:

- planificarea uşoară a procesului de dezvoltare
- posibilitatea de a defini şi livra software

în incremente

- simplitatea adoptării modificărilor
- eficienţa procesului de testare şi depanare
- simplitatea mentenanţei pe termen lung



CONCEPTELE PROIECTĂRII

ÎNCAPSULARE

Principiul încapsulării este criteriu esenţial în definirea **MODULELOR**.

Încapsularea impune constrângeri de acces la:

- detaliile procedurale,
- structurile de date locale.

Acestea sunt *ascunse* în spatele unei **INTERFEŢE**.

Deziderat: module *independente* ce *comunică minimal* între ele pentru realizarea în comun a unei funcţii de nivel mai înalt.

Oferă avantaje la adoptare modificări:

- localizarea intervenţiei în cod,
- evitarea propagării erorilor.

CONCEPTELE PROIECTĂRII

INDEPENDENŢĂ FUNCŢIONALĂ

Consecinţă directă a:

- modularizării
- abstractizării
- încapsulării

Modul:

- o (sub)funcţie specifică
- interfaţă simplă

Avantaje:

- simplifică dezvoltarea, testarea, întreţinerea
- reduce propagarea erorilor
- reutilizare

CONCEPTELE PROIECTĂRII

INDEPENDENŢĂ FUNCŢIONALĂ

Criterii calitative de evaluare:

- ***Coeziunea***

Indicator al **puterii funcţionale** relative a modulului.

Coeziune mare = task unic + interacţiune slabă cu alte module.

- ***Cuplarea***

Indicator al **independenţei** relative între module.

Depinde de:

- complexitatea interfeţei dintre module,
- punctul de intrare în modul,
- datele transferate prin interfaţă.

INDEPENDENŢA FUNCŢIONALĂ:

•Direct proporţională cu coeziunea

•Invers proporţională cu cuplarea.

Deziderat : **independenţă funcţională mare.**

CONCEPTELE PROIECTĂRII

RAFINARE

- Strategie de proiectare descendentă (top-down).
- Dezvoltare model proiect prin rafinarea, cu detalii, a nivelelor superioare.
- Proces de elaborare:
  - Iniţial : definiţii conceptuale pentru date / funcţii,
  - Ulterior: rafinări succesive cu adăugare de noi detalii.
- Proces complementar cu procesul de abstractizare:
  - *abstractizare* : suprimare detalii.
  - *rafinare* : dezvăluire detalii.

CONCEPTELE PROIECTĂRII

ASPECTE

Separarea problemelor este esențială în proiectarea sistemelor software complexe.

DAR - Unele probleme sunt greu de separat deoarece au *anvergură la nivelul sistemului.*

Ex. de cerințe care travesează (*crosscut*) mai multe componente ale sistemului: securitate, tranzacții, etc.

Aspect = reprezentarea unei probleme transversale (*crosscutting concern*)

Ex. Validare utilizator înregistrat înainte de a folosi diferite funcții ale aplicației.

Pe platforme orientate pe aspecte - mecanism ce permite :

- definire aspect în modul separat
- legare implementare aspect cu toate modulele pe care le traversează

CONCEPTELE PROIECTĂRII

REFACTORIZARE

Tehnică de reorganizare care simplifică modelul sau codul unei componente fără a-i schimba funcţionalitatea şi comportamentul (îmbunătăţeşte structura internă).

Exemple:

- eliminare redundanţe din modelul existent,
- înlocuire algoritmi ineficienţi sau inutili,
- reorganizare structuri de date prost construite sau nepotrivite,
- divizare componentă slab coezivă.

Alte exemple:
http://industriallogic.com/xp/refactoring/catalog.html

CONCEPTELE PROIECTĂRII

CLASE DE PROIECTARE

Clase de analiză : set de clase definite în procesul de analiză, ce descriu elemente din domeniul problemei; nivel înalt de abstractizare.

Categorii de clase de proiectare :

- clase care *rafinează clasele de analiză* cu detalii tehnice de proiectare ce ghidează implementarea clasei,
- un nou set de clase ce *implementează o infrastructură software* pentru a sprijini soluţia business.

CONCEPTE ALE PROIECTĂRII OO

CLASE DE PROIECTARE

Tipuri (pe diferite nivele ale arhitecturii):

- Clase la *interfaţa cu utilizatorul*
  - definesc abstractizările necesare HCI
  - reprezentări vizuale ale elementelor metaforei HCI
- Clase ale *domeniului business*
  - rafinările claselor de analiză
- Clase de *proces*
  - abstractizări business de nivel inferior necesare gestionării claselor de domeniu
- Clase suport *persistență*
  - Clase care implementează mecanisme de acces la date persistente
- Clase *sistem*
  - funcţii de management şi control necesare operării sistemului şi comunicărilor sale interne şi externe.

CONCEPTE ALE PROIECTĂRII OO

CLASE DE PROIECTARE

Criterii de revizuire (clasă **bine formată**):

- **completă şi suficientă**

Încapsulează toate şi numai atributele şi metodele necesare pentru modelarea conceptului în spaţiul soluţiei.

- **primitivă**

Fiecare metodă oferă un serviciu distinct.

- **puternic coezivă**

Set redus şi concentrat de responsabilităţi.

- **slab cuplată**

Colaborările cu alte clase reduse la un minim acceptabil.

(eventual valori mai mari în cadrul unui subsistem).

Evaluare formativă (2)

1. Enumerați câteva recomandări pentru creșterea calității proiectării.

2. Conform definițiilor prezentate, ce au în comun abstractizarea procedurală și abstractizarea datelor ?

3. Descrieți pe scurt principiul încapsulării.

4. Care sunt criteriile calitative de evaluare a independenței funcționale și care este relația independenței funcționale cu fiecare dintre acestea ?

5. Dați exemple de tipuri de clase de proiectare care nu sunt rafinări de clase de analiză.

https://forms.gle/teFvSC9CNjW5usas5

PLAN CURS

- Ingineria proiectării
- Proiectare în contextul ingineriei software
- Procesul de proiectare
- Calitatea proiectării
- Conceptele proiectării
- Principii ale proiectării OO (SOLID)

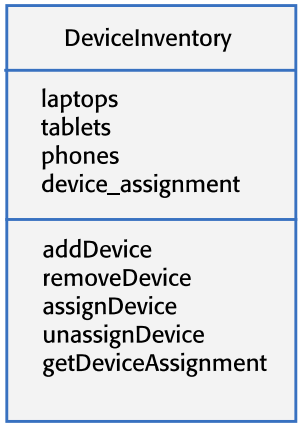
PRINCIPII ALE PROIECTĂRII OO

SRP (Single Resposability principle)

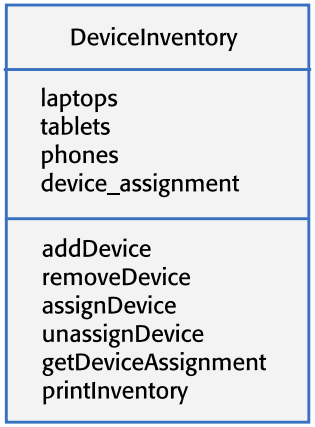
"O clasă trebuie să aibă un singur motiv de modificare, deci o singură reponsabilitate"

Toate elementele unei clase sau ale unui modul trebuie să aibă afinități funcționale între ele ⇒ coeziune internă puternică.

Metodă : Gruparea elementelor care se modifică din aceleași motive. Separarea elementelor care se modifică din motive diferite.



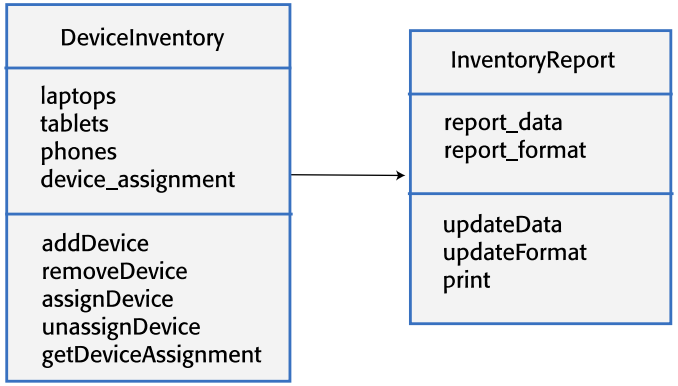
(a)



(b)

(a) Motiv de modificare = schimbare fundamentală în inventar (ex.înregistrare informații referitoare la cine folosește telefonul personal în interes de serviciu.)

(b) Un alt tip de date(un raport) este asociat cu clasa ⇒ 'Motiv de modificare' suplimentar = modificarea formatului raportului imprimat.



Soluție : adăugarea unei noi clase care să reprezinte raportul imprimat.

PRINCIPII ALE PROIECTĂRII OO

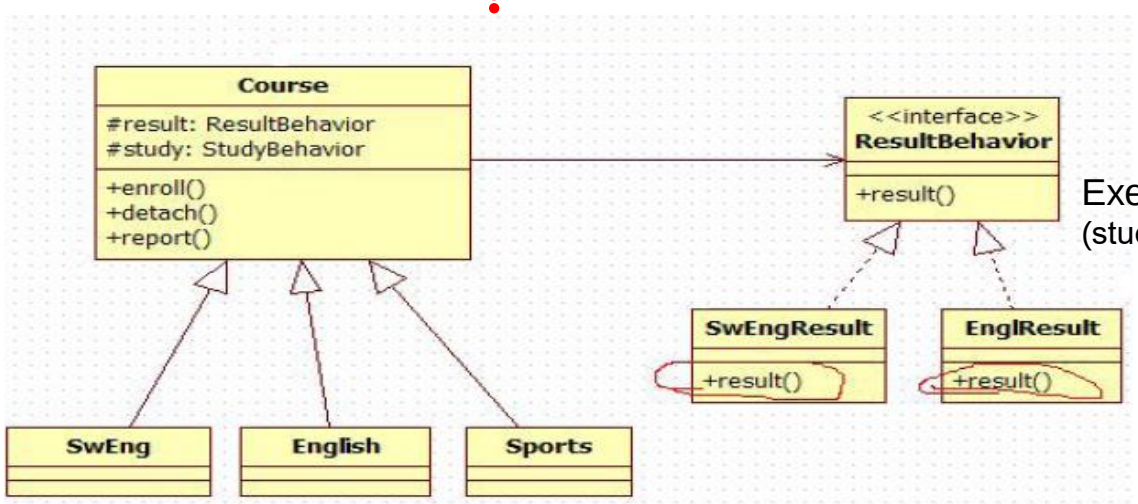
OCP (Open-Closed principle)

"O componentă (modul) trebuie să fie *deschisă pentru extensii* şi *închisă la modificări.*"

Extensibilă în domeniul funcţional căruia i se adresează, fără a necesita modificări interne (la nivel de cod sau de logică) ale componentei. ⇒ Separarea elementelor ce variază de ceea ce este invariabil.

Proiectantul va crea abstractizări (de regulă interfeţe) ce se interpun între funcţionalitatea extensibilă şi clasa proiectată pentru extinderea funcţionalităţii.

?



Exemplu de aplicare a şablonului Strategy.
(studiaţi Lab_DesignPatterns_1.pdf)

PRINCIPII ALE PROIECTĂRII OO

LSP (Liskov Substitution principle) (Barbara Liskov 1988)

"Subclasele trebuie să fie substituibile superclasei."

O componentă ce utilizează o superclasă trebuie să poată funcţiona corect dacă i se trimit obiecte instanţe ale oricărei subclase a acesteia.

Orice subclasă trebuie să onoreze orice contract dintre superclasă şi componentele ce o utilizează.

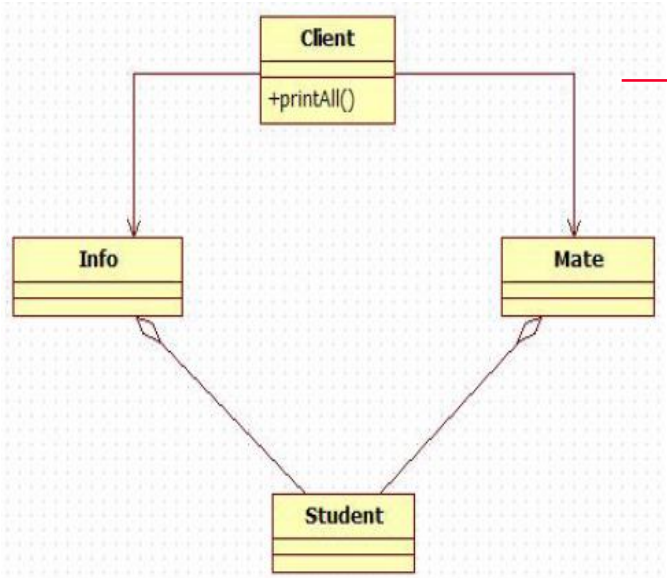
Contract:
precondiţie – adevărată înainte de utilizare
postcondiţie – adevărată după utilizare
Invarianţi – adevăraţi întotdeauna în contextul clasei

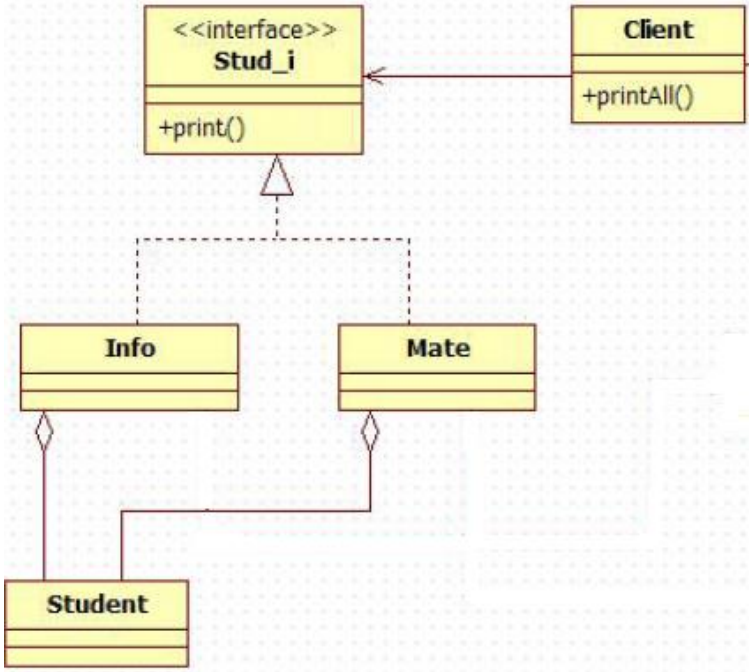
PRINCIPII ALE PROIECTĂRII OO

DIP (Dependency Inversion principle)

"Dependenţele vor fi definite faţă de abstractizări, nu faţă de elemente concrete."

Abstractizările: puncte de extindere fără mari complicatii.





(studiaţi Lab_DesignPatterns_1.pdf şi Lab_DesignPatterns_2.pdf)

1. *"High-level modules should not depend on low-level modules. Both should depend on abstractions."*
2. *"Abstractions should not depend on details. Details should depend on abstractions.„ Robert C. Martin*

PRINCIPII ALE PROIECTĂRII OO

ISP (Interface Segregation principle)

"Clients should not be forced to rely on interfaces they don't use."

Robert C. Martin

“Mai multe interfeţe, specifice diferitelor tipuri de clienţi, sunt mai bune decât o singură interfaţă cu scop general.”

Pentru acelaşi server se vor proiecta interfeţe specializate pe categorii de clienţi.

Fiecare interfaţă va prezenta operaţiile relevante categoriei.

Obs. Pot exista operaţii ce apar în mai multe categorii.

PRINCIPII ALE PROIECTĂRII OO

Principiile SOLID sunt strâns corelate.

Exemple :

- SRP bine implementat conduce la ISP

- OCP+LSP oferă suport pentru DIP

Bibliografie

Roger S. Pressman, **Software Engineering. A Practitioner's Approach**, ed.7, McGraw-Hill, 2010.

capitolele 6 - 14

Fairbanks, G. **Just Enough Software Architecture, A Risk Driven Approach**, Marshall&Brainerd, 2010

Proiectare vs.XP
https://www.martinfowler.com/articles/designDead.html

Principiile SOLID
https://www.techtarget.com/searchapparchitecture/feature/An-intro-to-the-5-SOLID-principles-of-object-oriented-design