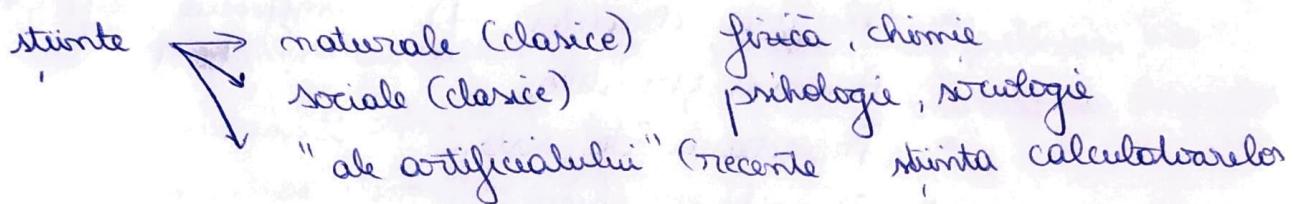


Introducere în Ingineria Sistemelor Soft

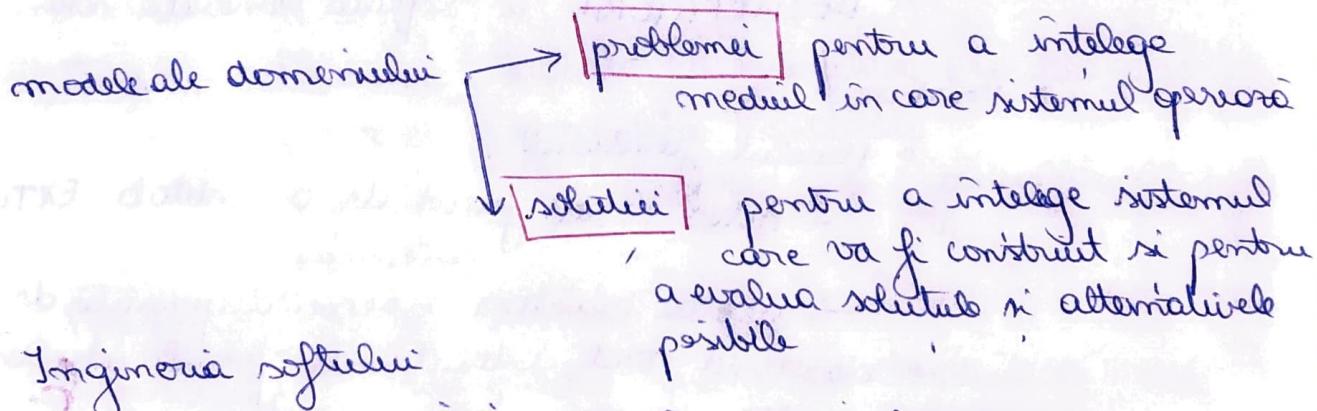
Ce presupune ISS-ul

- modelare
- rezolvare de probleme
- acumulare de cunoștințe
- argumentare



Modelarea → proces de reprezentare a elementelor sau evenimentelor unui sistem / fenomen

Modelul → reprezentare abstractă a unui sistem



Ingineria softului

activitate INGINEREASCĂ, nu algoritmice
experimentare + reutilizare de sabioane + evoluții multe

Bucket theory of the mind → modelul cascădă al procesului soft (acumulare seconală)

contă:

- funcționale
- nefuncționale

• utilizatorul va peteca achizițione bilete
 • constanțe legate GUI cu cadrile companiei
 • de operația sistemului

UML Unified Modeling Language

OCL Object Constraint Language

Analiza procesului de dezvoltare a softului

- Collecțarea cerințelor (1) inginerie soft orientată obiect
- Analiza cerințelor
- Proiectarea de sistem
- Proiectarea obiectuală
- Implementarea
- Testarea

(1) Collecțarea cerințelor definiree, de către CLIENT și DEZVOLTATORI a nevoilor / cerințelor sistemului

- Produse
 - Modelul funcțional
 - Actor: rol jucat de o entitate EXTERNA sistemului
 - Lini de utilizare: secvență generată de evenimente deservind toate interacțiunile posibile între un actor și sistem → funcționalități
 - Cerințe nefuncționale

(2) Analiza cerințelor

- descrierea cerințelor de utilizare

SDP: model COMPLET CONSISTENT NEAMBIGUU inconsistentă / ambiguități → rezolvate cu clientul

* model conceptual (diagramă de clase)

* model dinamic (diagramă de interacție)

ANALIZĂ → CE? me CM?

(3) Proiectarea de sistem

- definirea obiectivelor de proiectare și descompunerea sistemului în subsisteme
 - strategii utilizate pentru dezvoltarea sistemului
 - platforme
 - modalități de asigurare a persistenței datelor
 - fluxul de control global
 - politice de control a accesului

→ I proiectarea de sistem trece din domeniul problemei → domeniul soluției (cum? + ~~onde~~ introducerea conceptelor neformale clientului)

(4) Proiectarea obiectuală

- definirea de obiecte / clase din domeniul soluției
- se realizează legături dintre modelul obiectual de analiză și platforma de hardware / software de la (3)

Incluse: descrierea detaliată a interfețelor obiectelor / subsistemele selectate componente reziliante restructurare obiectuale → generalitate / extensibilitate optimizare

Rezultatul: model obiectual detaliat

(5) Implementare / translație modelului obiectual în cod

- rezultat: cod implementat
- Testare
 - identificarea diferențelor între înțelesul implementatorului și modelele sale, (cu seturi de date de test)
 - testare unitară (în timpul proiectarii obiectuale) (IV)
 - testare de integrare (proiectori de sistem) (III)
 - testare de sistem (analiză + calector) I, II
 - identificarea defectelor anterior predată sistemului.

Cours 2

Specificarea modelelor folosind UML

O notație (un limbaj) reprezentă o mulțime de reguli, textuale sau grafice, folosite pentru specificarea modelelor.

Intr-un proiect soft → instrument de comunicare (idei, decizii de analiză, proiecte)

Notație → trebuie să aibă o SEMANTICĂ bine definită

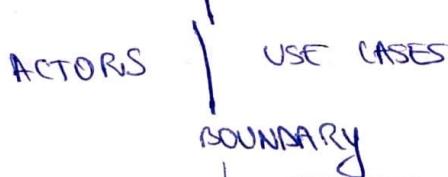
UML → limbajul standard adoptat de industrie pentru reprezentarea modelelor orientate obiect

Tipuri de modele ale sistemelor soft

- modelul FUNCTIONAL
 - + funcționalitatea sistemului din perspectiva utilizatorului
 - diagrame de cerință utilitate
- modelul OBIECTUAL
 - + structura sistemului în termeni de clase / atribută
 - diagrame de clase
- modelul dinamic
 - + comportamentul intern al sistemului
 - diagrame de introducere / activitate trimitute către

• USE CASE DIAGRAM

Identificarea actoarelor și a condițiilor de utilizare permit definirea frontierei sistemului (SYSTEM BOUNDARY) → diferențierea între sarcinile îndeplinite de sistem și cele îndeplinite de mediul său.



Contextul canalui de utilizare poate fi
descrierea textuală → sablon ce conține
climbaj natural

- Nume
- Actori participanți
- flux de evenimente
- precondiții
- postcondiții
- cerințe de calitate

Un caz de utilizare → o ABSTRACTIZARE ce acoperă toate scenariile posibile afișând funcționalități diverse

Un scenario → o INSTANȚĂ a unui caz de utilizare ce descrie o secvență ~~completă~~ CONCRETA de acțiuni

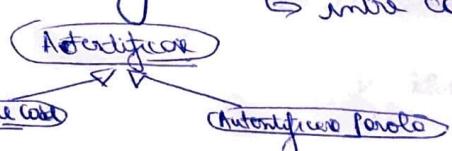
scenario → ilustrare situată tipică
INTEGRABILITATE

scenario de utilizare → exprimă toate situațiile posibile
COMPETITIVITATE

Scenario → sablon cu 3 compun.

- Nume scenario
- Instanțele actorilor participanți
- Flux de evenimente

Relații → comunicare (associere UML binară bidirecțională)
→ inclusivare (nu e opțională) include → inclusiv
→ extindere (nu este extensie de sine statutor)



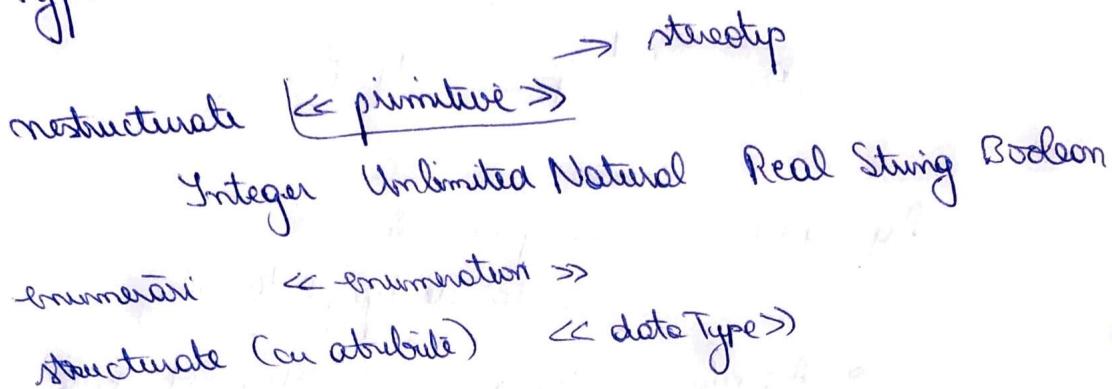
extensie ← - - - extensie
extends

CLASS DIAGRAM

→ utilitate pentru descrerea structurii de sistem

Clasele → abstracții ce specifică structura și comportamentul comunelor multimi de obiecte

Data Types



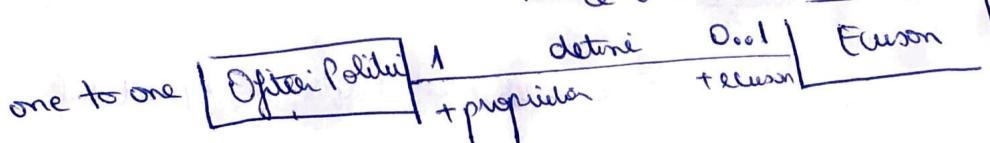
Close → litere mari

Object → litere mici

O legătură → o conexiune între 2 obiecte
 Asocieri → relații între clase și reprezintă mulțimi de legături

Multiplicitate în relații

număr de relații → clarifică scopul asocierei
 → permite diferențierea între asociere difuză și corectă a relației



one 1 or 0..1

one - to - one

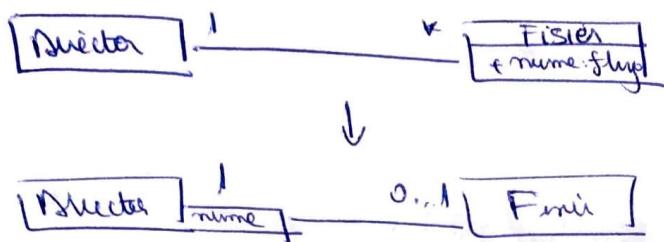
one - to - many

many to many

one 1 or 0..1

many 1..* or 0..* or *

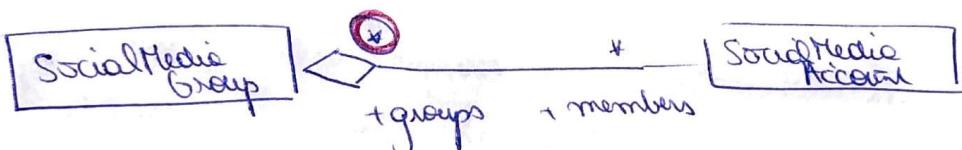
Calificarea → metoda de reducere a multiplitudinii primării cheii



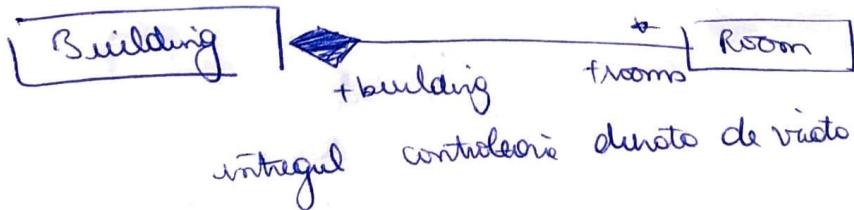
Agregare -

- cerc particular de asociere
- relație de tip parte întreg

- Agregare partajată (partile pot exista și independent) (◇) - pe capătul desigur întreg



- Componente
 - o parte poate fi continută în cel mult un întreg la un moment dat



Generalizare: factorii arbitrajii și opacitate

UML: operări → specificație comportamentului
metode → implementarea comportamentului

Diagramme de interaction → diagramme de séquentielle (1)
→ diagramme de communication (2)

(1), (2) are equivalent

Diagrama de secvență → primulă perspectivă temporală

1

adware (executie unei metode)

→ utilizate pentru descrierea unor interacțiuni specifice (SCENARIU) sau a unor generale (CAZURI de UTILIZARE)

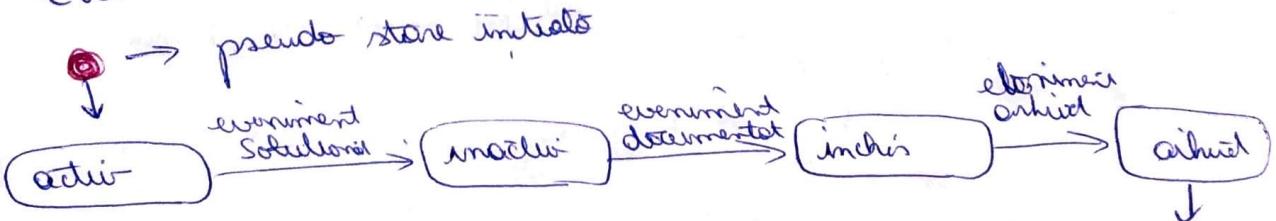
→ accentul e pe securitatea menajelor în timp,
pe vînd le diagnoza de comunicare aceste e pe
colaborările între obiectele participante

Diagramme de interaction → identificarea responsabilității
closelor deja existente + identificarea de noi cluse

Stări și tranziții → se cer condiții satisfăcătoare de valoare
(STARE) abordările unei ţări

fiziologie → o schimbare de mărime

Experiment



O marină cu stări VML

↳ notatice folosite pentru a descurca
succesivile de stat prin care trece imobil
sub actualele circumstanțe locului

vedere control

↑ personali

Diagrama de activități - descrie modul de realizare al unei

oameni comportament în termeni uneia sau a mai multor securitate obiectiv și a fluxului de obiecte necesare pentru coordonarea activității

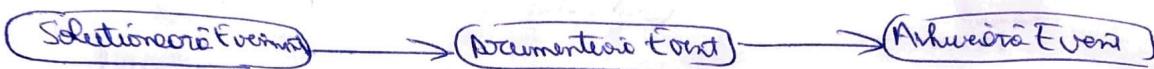
- sunt secvențiale activitate → o activitate

securitate

→ un graf de măblărie

- execuție activităților

SEQUENTIALĂ



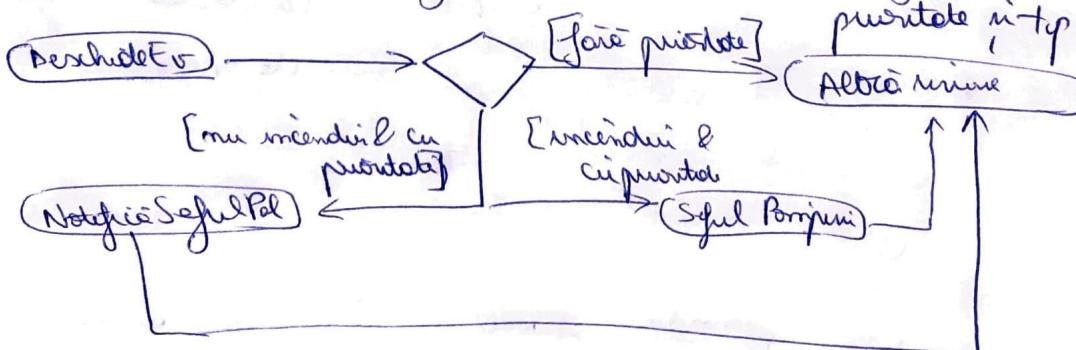
Elementele de control permit coordonarea flexibilă de control
mechanism de reprezentare a deciziilor / concurenții /
sincronizare

Tipuri elemente de control → moduri decizionale

fork

join

NOD DECIZIONAL - gestionarea evenimentelor în limitele de



NODURILE fork / join - reprezentarea concurenței

ni o sincronizare

↳ diversificarea flexibilității de control în threaduri
↳ sincronizarea thread-urilor și combinația
flexibilității într-un singur thread

Activitățile pot fi grupate pe partile → indică obiectul
subsistemul ce le va implementa

Cours 3

Colectarea cerintelor

Cerinte. Ingineria Cerintelor

- cerinte → un element de funcționalitate
- → o constrângere

Ingineria cerintelor → subdomeniul al ingineriei softului
→ se ocupă cu definirea cerintelor sistemelor și
↓ activități:

- colectare cerintele → specificație sistemului⁽¹⁾
- analiza cerintelor → modelul de ordere⁽²⁾

(1), (2) reflectă același informație
diferit pînă la ipostaze

specificația sistemului → limbaj natural → client
modelul de ordere → notatice (semi)formale → dezvoltator
dezvoltator
dezvoltator

Colectarea cerintelor bazate pe scenarii

- tipuri de scenarii
 - initial - scenariu descriind modelul curent de desfășurare a proceselor de lucru în mediu
 - ultim scenariu prescripție

Aktivități

- ale colectării cerintelor bazate pe scenarii
- identificarea actorilor
- identificarea scenariilor
- identificarea UC
- rafinarea UC
- identificarea relațiilor dintre UC
- identificarea cerintelor nefuncționale

Cerintele functionale descriu interacțiunile dintre sistem
și mediul aceluia INDIERENT de implementare

Cerinte nefuncționale cerințe

- Categorii → utilizabilitate
→ performanță
→ fiabilitate
→ reutilizabilitate

Inginerie Greenfield → procesul de dezvoltare începe de la 0
Re-inginerie → se apără același funcționalitate extinsă
Inginerie Integrator → reprezintă implementarea doar
a interfațelor

Metoda JAD

Joint Application Design

metoda de colectare a cerintelor

→ activitate de colectare a cerintelor → un workshop

SPECIFICAȚIA
COMPLETĂ A
SISTEMULUI

← părere finală → ~~PEPSI ALĂNGAȚĂ~~
prezentată într-un moderator de reuniune

Proiect definition → interviu manager + client
scop + obiective → reținute în

Management Definition Guide

Research → domeniul problemei
primărt UC → Scenarii Agenda
Preliminary Specifacții

Preparare → o primă versiune a documentului JAD
Working Document

Echipe PM + Client + Utilizatori + Devs

Scenarii → Modulul coordonator echipei în
elaborarea spec. sist (3-5 zile)

Scenarii + UC → analize → scurii documente

Final Document Preparare Se include scriere Form
1-2 debaturi → final vînătoare

de trienărie → comportamentul său să fie sănătos

Curs 4

Analiza cerintelor

Obiectie

- scopul analizei cerintelor → realizarea unui model al sistemului corect / complet / neambiguu
↳ de analiză
- accentul → structurarea cerintelor colectate în etapa formalizarei
↳ formalizarea → ameliorare
- formalizarea identifică ambiguități / inconsistente.
în descrierea cerintelor, soluții și discuții cu clientul → se perfectează
SPECIFICAREA SISTEMULUI

Colectarea + Analiza cerintelor

↳ Activități concomitente, ce se desfășoară
ITERATIV - INCREMENTAL

Concepte

Modelul obiectual de analiză

- concepte manipulabile de sistem (proprietăți + relații)
 - se reprezintă cu ajutorul CLASS DIAGRAM-ului
- o clasa din modelul de analiză → abstracție pentru una sau mai multe clase din actualul monște

Modelul dinamic

- surprinde comportamentul sistemului
- DIAGRAME de SECVENȚĂ + TRĂZITIE

Diagramme de secvență → surprind interacțiunile dintre o mulțime de obiecte, în cadrul UC-ului
de tranzitie → comportamentul unui singur obiect

Modelul dinamic → permite identificarea responsabilităților entităților / claselor existente, precum și identificarea de clase / obiecte / resurse noi

Modelele de analiză surprind doar concepte / abstracții (relații) comportamente percepute de utilizatori (domeniu problemelor)

modelul obiectual de analiză constă din clase: entity
boundary
control

Clase **«entity»** responsabile de informație persistenta din sistem

«boundary» responsabile de interacțiunea exterior cu sistemul

«control» responsabile de realizarea comandelor utilizatorilor

Avantajele: date / obiecte mai mici, specificitate
! Model mai ușor de modificat

Generalizare + Specializare → permit organizarea conceptelor în arhitectură

Generalizare = activitate de modelare, având drept scop identificarea unor concepte deschise, permisă de la mulți de nivel și

Specializare = activitate de modelare, având drept scop identificarea unor concepte specifice, permisă de la mulți de nivel împreună

Obiectele control → responsabile de coordonarea obiectelor boundary + entity

Realizarea diagramelor de secvență

- realizarea legătură dintre clase / obiecte și coziile de utilizare / scenariile de uz

Azori → cunoscute
→ roluș
→ multiplicitate

Modelarea comportamentului cu diagrame de stări

Diagrama de secvență: comportamentul din perspectiva unei singure UC

de tranzitie a stărilor: perspectiva unei singure obiecte

Rel: identificarea UC-ului omului

Rezumatul modelului de analiză

- 1) Asigurarea corectitudinii
- 2) completitudinii
- 3) consistenței
- 4) realității

Curs 5

Proiectarea de sistem (I)

→ procesul de transformare a modelului rezultat din inginerie civilă → model arhitectural al sistemului

→ proceșe: [Obiective de proiectare
Arhitectura sistemului]

Obiective de proiectare:

Persevere din cerințele nefuncționale

Calitate ale sistemului pe care dezvoltatorii trebuie

să le optimizeze

Arhitectura sistemului:

- sub sisteme componente + responsabilități și dependențe
- maparea subsistemelor la hardware

Subsisteme și clase

Subsistem = parte încloasă a unui sistem,
caracterizată de interfețe bine definite
ce încapsulează starea și comportamentele
claserelor componente

descompunerea sistemului în subsisteme (relativ) independente

↳ se permite dezvoltarea concurentă

Sistemele complexe le corespund mai multe nivele de
descompunere COMPOSITE PATTERN

Componente UML

- componentă logică - subsistem ce nu are
un echivalent la nivelul fizic
- fizic - ce are

Serviciu = multime de operatii intrinseci (definite cu acelasi scop)

- subsistemele sunt caracterizate de serviciile oferite altor subsisteme
- serviciile se identifică în timpul proiectarii de sistem unic

Interfață = multime de operații intrinseci, complet specifică și definite în timpul proiectarii obiectului

API = specificare a unei interfețe subsistem într-un lanț de proiectare și implementare

Cuplare = măriție a dependenței dintre 2 subsisteme
Coacere - măriție a dependențelor - slabă sau nici de dependență
din interior unui subsistem - strânsă

coacere înaltă - un mare de cloșe pe teren
relativistic

Stratificare

și Partitionare

Stratificare: multime ordonată de straturi

grup de subsisteme ce oferă servicii intrinseci

Arhitectură → inclusă servicii din stratul IMENAT anterior
distrusă din toate straturile anterioare

Partitionare: grup generat de sisteme, le acelasi nivel

Descompunere

Stratificare + Partitionare

Descompunerea sistemului

identificarea subsistemelor, a serviciilor și a relațiilor dintre acestea

Stil arhitectural = rețea de descompunere a sistemelor
Arhitectura software = existența unui stil arhitectural

Stil Arhitectural → MVC (tip particular de Repository)
Subsistemele mode → ne depind de noul subsistem view / controller

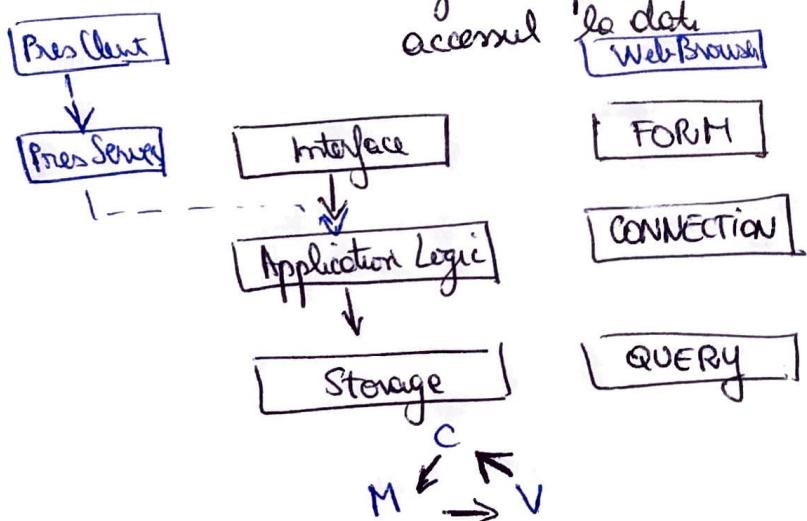
Peer-to-Peer (generalizare a stilului arhitectural client server)

↪ un subsistem → client și sau server

Fiecare subsistem poate solicita / oferi servicii

Three-tier - subsistene pe interfață utilizator logic aplicatiei accesul la date

3 layuri
SOUNDARY
ENTITY + CONTROL
gestionarea și lucru cu acasă
la date cu caracter permanent



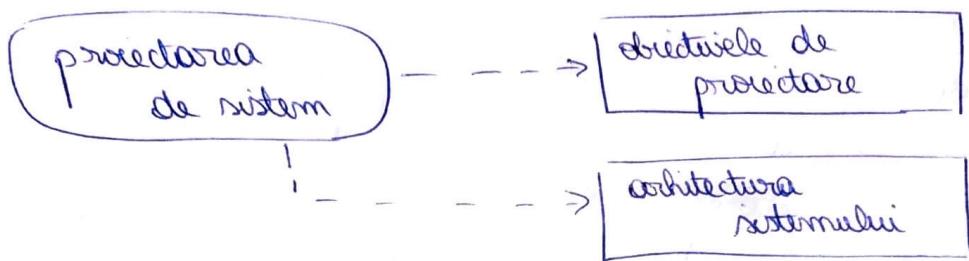
Four-tier Interfață → prezentare client
prezentare server

Pipes and filters → stil arhitectural

Pipe - conexiune între 2 pasi de procesare
Filter - subsistem ce efectuează un pas de procesare input-ul altuia

(wv) 6

Proiectarea de sistem 2



- Procesul de transformare a modelului rezultat din ingineria contorelor într-un MODEL ARHITECTURAL al sistemului

Activități ale proiectării de sistem

- identificarea obiectivelor de proiectare
(dezvoltatorii trebuie să optimizeze anumite calități ale sistemului)
- descompunerea inițială a sistemului
(pe baza modelului funcțional + de analiză)
- reafinarea descompunerii inițiale pentru a răspunde obiectivelor de proiectare
 - + maparea hardware - software
 - + gestionarea datelor cu caracter persistent
 - + definirea politicilor privind controlul accesului
 - + stabilitatea fluxului global de control
 - + desemnarea controlierilor limita

Obiective de proiectare = criterii de calitate pe care sistemul trebuie să se concentreze (specificate explicit)

- Categorii {
- performanță
 - dependabilitate
 - cost
 - întreținere
 - criterii utilizator

Sablonul de proiectare Facade (structural)

- permite reducerea dependențelor (cupluri) dintre un subsistem și clientul său

Facade oferă o interfață simplificată a unei librării, a unui framework sau a unui set complex de clase

Interfața cuprinde doar serviciile "utile" pentru clienti

Ex: comandă pizza → operatorul "facade-ul" tuturor serviciilor / comportamentelor magazinului

+ diminuarea numărului de dependențe între client și clasele de implementare ale unei abstractii

Participanți: Facade + Clasele Subsistemu

Clasele subsistemu → nu stie de existenta facadelor (nu pot avea referinte spre ea)

UML deployment diagram (O diagrame de depunere UML)

ilustrează relația dintre componentele suntem și modul

componentă = unitate autoconținută, care oferă servicii altor componente / echipă

mod = dispozitiv fizic (calculator) sau mediu de execuție pentru componente

nodurile → 

Maparea subsistemelor la hardware

- Strategie de allocație a subsistemelor pe echipamentele hardware și proiectare infrastructurii de comunicare între acestea → influențează performanța + complexitatea

Sablonul de proiectare Proxy (structural)

Ex



- sablonul asigură, pentru un obiect, un surrogat sau un intermediar, în scopul controlului accesului la acesta

- Proxy păstrează o referință ce îi permite să acceseze subiectul real

proxy $\xrightarrow{\text{la distanță}}$ virtual $\xrightarrow{\text{proxy}}$ reprezentant local pentru-un un obiect dintr-un spațiu de adresă difuz

crează la cerere obiecte constitutive (pot depărtă informații suplimentare)

Gestirea datelor persistente

- fizice \rightarrow mai mulți cititori, un scriitor
- DB \rightarrow date acasăte de cititori și scriitori concurenți

Strategii de memorare

- fizice (viteză + durată) probleme concurență + pierderi de date

- DB relaționale (lente)

- DB orientată obiect (foarte lente)

Definirea politicii privind controlul accesului
cum modelăm accesul?

Analiză (UC le difuză)
Procedură (criptare "clorii")

Matrice de acces

- liniile → actoare
- coloanele → clase ale căror drepturi de acces doar nu și modelate

Variabili

- tabel de acces global (actor clase operate)
- lista de control (actor operate)
- capabilități (clase operate) → actor

Flux de control → sevenirea activității

- flux de control → procedural și ~~este~~ este apel de la act
- thread-uri variente concurențiale a controlului procedural

Descreerea controlorilor liniști (date corante, căderi de retea)

CAZURI DE UTILIZARE LIMITE

Pornire / Oprire

Gestionarea excepțiilor

→ jocul exec → UC excepțional
(dintr-un alt UC existent)

Cours 7-8

Proiectare obiectuală Reutilizare

- identificarea de noi obiecte din domeniul soluției
- reafinarea celor existente
- adaptarea componentelor reutilizate
- specificarea rugozășă a interfețelor subsistemelor / claselor componente

Activități ale proiectării obiectuale

- REUTILIZARE (noutăți de proiectare / componente de bibliotecă pentru structuri de date)
- SPECIFICAREA INTERFETELOR (specificație completă)
- RESTRUCTURAREA MODELULUI OBIECTUAL
- OPTIMIZAREA MODELULUI OBIECTUAL
(asigurarea criteriilor de performanță)

Etapile identificării obiectelor din domeniul problemei / soluției

ANALIZĂ - obiectele din domeniul problemei + cele obiecte din domeniul soluției vizibile utilizatorului

PROIECTAREA DE SISTEM - obiectele din domeniul soluției în termenii platformei hardware / software

PROIECTAREA OBIECTUALĂ - reafinarea obiectelor din ambele domenii identificate anterior + identificarea obiectelor noi din domeniul soluției

Mostenirea specifică / implementării

- Mostenirea implementării

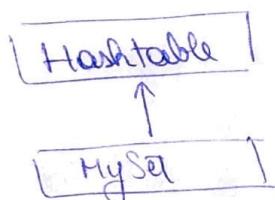
' ↳ utilizarea mostenirii având drept urmă scop

REUTILIZAREA CODULUI

- Mostenirea specifică / interfeței → clasificare ^{conceptele} codului _{în ierarhii}

Delegarea - alternativă la mostenirea implementării atunci când se doară implementarea unei funcționalități prin reutilizare

- MODIFICABILITATE
- SUBTIPARE



MySet are în constructor un
Hashtable

Față de delegare → MySet EXTENSAHashtable

Principiul Liskov al substituției
DEFINITION FORMALĂ a conceptului de mostenire
specifică

SOLID

Single Responsibility Principle

Open/Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

SOLID

↳ proprietatea claseri

• PRINCIPIUL RESPONSABILITĂȚII UNICE

Nu trebuie să existe mai mult de o responsabilitate per clasa

• PRINCIPIUL DESCRISSIEI / ÎNCRISSEI

Entitățile software (clase, module, funcții) trebuie să fie DESCRISE pentru extindere și ÎNCRISSE pentru modificare

• PRINCIPIUL SEPARĂRII INTERFETELOR

Clienții nu trebuie constrâni să depindă de interfețe pe care nu le utiliză

• PRINCIPIUL INVERSĂRII DEPENDENȚELOR

Modulele de nivel înalt nu ar trebui să depindă de modulele de nivel jos, modulele ar trebui să depindă de ABSTRACTIZĂRI

Abstractizările nu ar trebui să depindă de detaliu, modulele ar trebui să depindă de ABSTRACTIZĂRI

Salvarea de proiectare

• Obiectivul proiectării de sistem:

Gestionarea complexității, prin crearea unei arhitecturi stabile, descompunând sistemul în subsisteme cu interfețe bine definite și slab cuplate (UN GRAN DE RIGORITATE)

- Obiectivul proiectării obiectuale: FLEXIBILITATE

Proiectarea unui soft modificabil și extensibil

Sablonul de proiectare Adapter

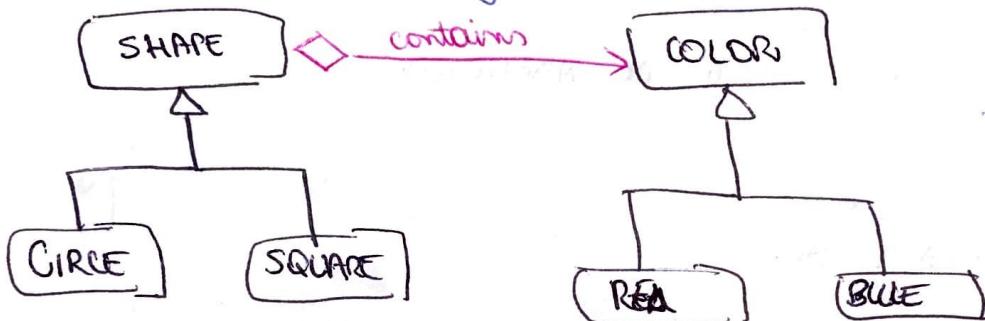
- încapsularea componentelor existente

- structural
- permite colaborarea dintre obiecte cu interfețe incompatibile

Scop: Transformă interfața unei clase existente (legacy class) într-o altă interfață astfel încât clientul să poată colabora fără nicio modificare la nivel abstract

Sablonul de proiectare Bridge

- decouplează abstractiile de implementare



- structural

Scop: Decouplează o abstractiune de implementare și, astfel încât cele 2 să poată varia independent. ~~Abstracție~~ Abstracție și implementare pot fi reținute independente

Sablonul de proiectare Strategy

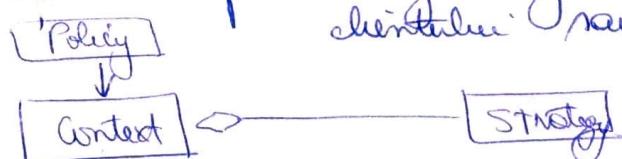
- behavioral
încapsularea algoritmilor

Apare atunci când există un număr de algoritmi diferiți pentru rezolvarea aceluiași probleme și neexistă de o interacțiunea în mod direct algoritmul utilizat

Soluție: Clasa care implementează varianta default + clase derivate oferind ceva specific algoritmul de lărgă

Algoritmul variabil independent de client și îutilizări (Interfață comună pentru algoritmi similari)

Consecință: se pot adăuga algoritmi noi fără modificarea clientului sau contextului



configurarea în funcție de policy

Sablonul de proiectare Composite

incapsularea jerarhiei

- structural

- Prin intermediul unei interfețe comune, se reprezintă jerarhia de întregie și adunarea variabilelor (resurse) astfel încât funcțiile și agregațiile să fie tratate uniform

Sablonul de proiectare Abstract Factory

incapsularea platformelor

- creational

- furnizare o interfață pentru crearea formulei de obicei variabile sau dependențe, fără specificație directă în cadrul

- clientul este decuplat de clasa produs concreta

Sablonul de proiectare Command

incapsularea fluxului de lucru

- comportamental

- incapsularea o șase ca un obiect, permitând parametrizarea clientului cu diverse cerințe precum și formarea unei cozi + organizarea reacțiilor pe bază

Sablonul de proiectare State

- comportamental

- oferă claselor noile pentru fiere "state" și unui obiect (clase cu comportamente specifice)

- eliminarea numărului mare de condiții

- state-urile sunt de exitate celorlalte (faza de Strategy)

Sablonul de proiectare ~~Factory~~ Singleton

- creational

Singleton

aceas

- O clasă are 3 singure instanțe

(global la ea)

- constructor privat →

new

(înloc.: metode de creare statică)

Ex: generatul unei tablă

Curs 9

Proiectarea obiectuală: Specificarea interfețelor

Produse existente pînă în momentul etapei de specificare a interfețelor

- modelul obiectual de analiză
(entități din domeniul problemei
 - + obiecte boundary, control întreagile utilizatorul
- descompunerea sistemului
 - + obiecte noi din domeniul soluției
- mapeaza hardware / software
- salboane de proiectare reutilizate + componente de bibliotecă reutilizate pentru structuri de date și servicii de bază

Subactivități

în specificarea interfețelor

- identificarea atributelor + operațiilor lipsă
- specificare vizibilitate și semnaturi operațiilor
- specificare pre/post-condițiilor pentru operații și a invariabilor de tip

OCL:

Object Constraint Language

- limbaj formal, folosit pentru definirea de expresii pe modelele UML

- nu e limbaj de scriere statelor, complementar UML-ului

- limbaj declarativ → evaluarea expresiilor OCL nu modifică starea modelului UML asociat

- limbaj ce susține principalele caracteristici OOP

* invariant - tip de constrângere

- se formează în contextul unei clasificări

context X

inv invX1 : self. $q \geq 0$

Exemplu de invariant cu iterator pe colectii

toate programările la un client au loc în timpul programului

context DentistProgram

inv: self.appointments → for All (a: Appointments |
a.start >= self.start) and a.end <= self.end)

Mostenirea contractelor

* clientul se asteaptă ca un contract formulat relativ la clasa de baza, să fie respectat și de clasele derivate

Reguli privind mostenirea contractelor

(consecință a principiului Liskov al substituției)

PRECONDITII:

Unei metode dintr-o subclasa, li este permis să slabescă precondiția unei metodelor pe care o SUPRADEFINESTE
(o metodă care supraredefinește poate gestiona mai multe cazuri decât cea supraredefinită)

POSTCONDITII:

O metodă care SUPRADEFINESTE trebuie să angaje o postcondiție îll propri la fel de puternică precum cea supraredefinită

INVARIANTI:

O subclasa trebuie să respecte toți invariantii superclaselor sale; poate, eventual, introduce invarianti mai puternici decât cei mosteniți.

Transformarea modelelor în cod

Modele și transformări

transformare → imbunătățirea unui model,
cu păstrarea celorlalte proprietăți ale acestuia
(într-o succesiune de pași mărunți)

proiectarea structuralei + implementarea sistemului

↳ transformări: • Optimizare

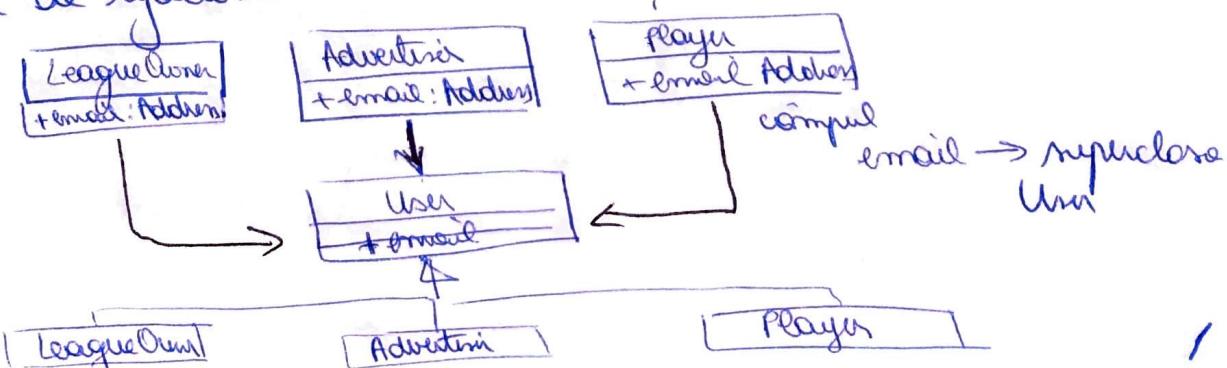
- Reprezentare Asocierilor
- Reprezentare contractele
- Reprezentare entităților persistente

atribute derivate (temp de acces între noduri)
reducere multiplicitatii asocierilor
asocieri redundante (eficiente)

Tipuri de transformări → simplificare / detaliere sau optimizare modelului initial

- transformări la nivelul modelului
 - string adresa → doar adresa → string strâns
- refactoring (imbunătățește aspectul sistemului)
 - intervenție numărătore
 - Nu afectează funcționalitatea (⊕)
- inginerie directă (model → cod susținut)
 - concepte de modelare → transformate în cod susținut
 - (atribute, asocieri, signature de operații)
- inginerie inversă (cod susținut → model)
 - * când modelul și codul au evoluat desincronizat

(*) pași de refactoring sunt intercalati cu teste (incremental)



- Pull Up Field atributul comun "email" → în superclasa
- Pull Up Constructor Body User (email) → subclasele primește super (email)
- Pull Up Method metoda notify () trece în Uml și dispără din subclase

Scopul Ingineriei directe este de a întreține o CORRESPONDENȚĂ între modelul obiectual de proiectare și cod, diminuând efortul de implementare (cost)

Scopul Ingineriei inverse este de a recrea modelul aferent unui sistem, la urmări a inexistenței sale sau a lipsii de informații a acesteia cu codul său

pentru inginerie directă → se pierd informații din model →
→ inginerie inversă nu produce modelul identic

Principii de transformare:

- 1) o transformare trebuie să vizere optimizarea din perspectiva unui SINGUR CRITERIU
- 2) fiecare transformare trebuie să fie locală
- 3) fiecare transformare trebuie aplicată mult de alte schimăriri
- 4) fiecare transformare trebuie urmată de validare aferente (teste + diagrame modificate)

Surse de influență la nivelul unui model obiectual

- traversarea unui număr mare de obiecte și alelor cu obiecte MANY

- plasarea exponențială a unor abstracții

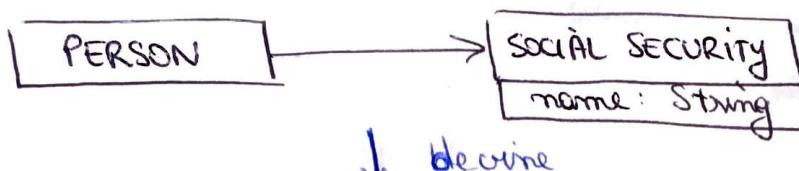
Traversarea mulților obiecte → Soluții: obiecte directe, Redundanță

Traversarea obiectelor "MANY" ↗ EXTRA

Plasarea exponențială: Soluție: ordonarea / mărirea obiectelor

Plasarea exponențială: Soluție: relocarea atributului în clasa aferentă

Transformarea unor clase în attribut
(comenzi spre începutul implementării)



// caz particular de REFACTORIZARE

Inlined Class

Gestionarea operațiilor Constituire

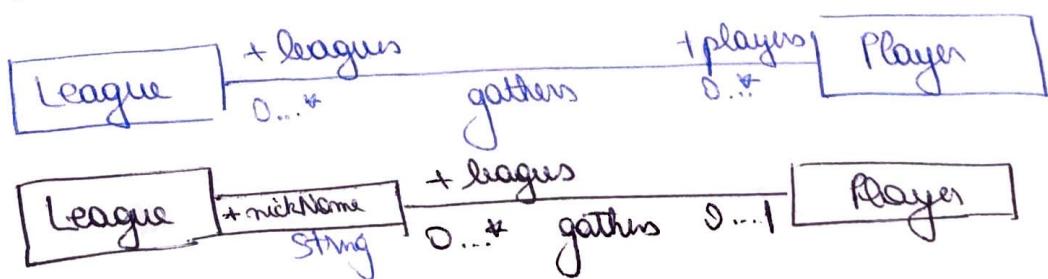
- * operațiile 'constituționale', de tipul 'încarcării' unei obiecte grafice, pot fi amânate până în momentele în care e necesară vizualizarea acestora
- * Rezultatul operațiilor complexe, apelate frecvent, din schimbările 'nor' → memorate în atrbute 'privale'
(se optimizează timpul de răspuns, dar e nevoie de memorie implementată)

Multiplicitatea ONE necesită o referință, ceea ce MANY necesită

o colectie de referințe

Asociările calificate → utilizate pentru a "reduce"
multiplicitatea unui capăt MANY

Calificatorul asociării este un atribut al clasei din capătul MANY ce se dovedește să fie redus, atribut cu valori UNICE în contextul asociării, nu neapărat și la nivel global



Reprezentarea contractelor

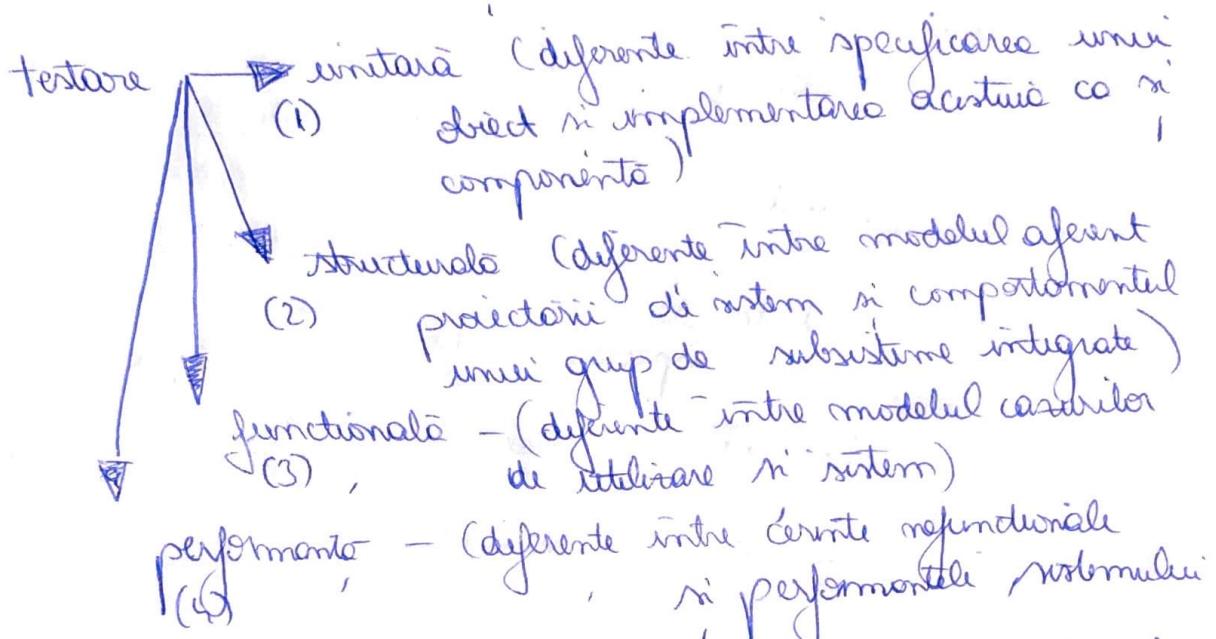
- Verificarea precondițiilor
(la începutul jocului metode)
dacă nu e adevarat → excepție
- Verificarea postcondițiilor
(la sfârșitul jocului metode)
contractul e închis → excepție
- Verificarea invariantei
(se verifică odată cu POSTCONDITIILE)
- Monitorizare contractul
codul de verificare al asertorilor trebuie
încapsulat în noulul următor metode specifice
pentru apelarea lor din subclase

Reprezentarea asociențelor

- one - to - one / one - to - many - folosind chei străine
- many - to - many - folosind table de legături

Testarea sistemelor soft

Testarea = procesul de identificare a diferențelor dintre comportamentul dorit / așteptat al sistemului și cel observat



* încercare de a demonstra că implementarea sistemului e inconsistentă cu modelele acăzute

Stare de eroare = manifestare a unei defecte (greșală de proiectare ce poate determina un comportament anomal)

Test stub = implementare parțială a unei componente, de care depinde componenta de test

Test driver = -II- care depinde de componenta de test

Testarea sistemelor soft - ACTIVITĂȚI

- Planificarea testării
- Inspectarea componentelor (inspectare manuală a codului sursă)
- Testarea utilizabilității (detecarea defectelor în proiectarea interfeței)
- (1) - Testarea uriașă (defecte la nivelul obiectelor)
- (2) - Testarea de integrare
- (3)(4) - Testarea de sistem (testare sistemul integral în ansamblu)

Inspectarea componentelor

Metoda inspectării a lui Fagan

- primul proces structurat de inspectare
- echipă de dezvoltare → autorul componentei
- moderator
- recensori

Pași:

- 1 Overview: autorul prezintă sumar scopul și obiectivele inspectării
- Preparare: recensori se familiarizează pe codul componentei
- (*) Scenariu de inspectare: o persoană parafonată codul nr. și membrii echipei semnaliză probleme
- Revisiuni: autorul revizuează codul componentei conform observațiilor
- Verifică: moderatorul verifică varanțe revizuite

Active Design Review → Bucările (*)

- Problemele sunt numălate în cadrul fazei de pregătire
- la finalul etapei de pregătire recensorul prezintă că a înțeles din componentă
- autorul colectează feedback în urma unei întâlniri individuale cu membrii

Testarea utilizabilității

- diferențele dintre sistem și antrenorul utilizatorului, cu privire la comportamentul acestuia

Tipuri:

- teste bazate pe scenarii
- teste bazate pe prototipuri
- teste pe baza sistemului real

Testarea unitară

- se axează pe componentele elementare ale sistemului soft (OBJECTE + SUBSISTEME)
- paralelismul e posibil (componentele pot fi utilizate testate independent și în același timp)

Tehnici de testare unitară

- testare bazată pe echivalente
- testare frontierelor
- testare căile de execuție
- testare bazată pe stări
- testare perimetrului

* Testare bazată pe echivalente - tehnice blackbox
- număr de cazuri de testare minim

Pasi:
1. Identificarea clorilor de echivalente
2. Selectarea intrărilor pentru test

- Acoperire: fiecare intrare ∈ unei clase de echivalente
- Dugund: o intrare nu poate apartine mai multor clase de echivalente
- Reprezentanță: intrarea unei membre → eroane → al unei clase de echivalente
→ aceeași stare va putea fi detectată de orice alt membru al clasăi

(Ex:  metoda ce returnează nr. de zile dintr-o lună
clase de echivalente → pentru lună

(se alege un reprezentant arbitrat)

(31 de zile)
(30 de zile)

28/29

pentru an
(bisiști)
(non-bisiști)

* Testare frontierelor

(un particular al testării unitare)

- se explorează corurile limite

- devotătorii sunt corurile speciale

* Testare căilor de execuție

- metodă de tip "white box"
 - defecți identificate prin testarea tuturor componentelor de execuție
 - punctul de start îl constituie conținutul unei reprezentări logice **Flow CONTROL**
 - nodurile corespund instrucțiunilor
 - arcele corespund fluxului de control
- (testare completă prevede proiecția calelor de test a.i. fiecare ARE al diagramei de activități să fie traversat cel putin o dată)

* Testare bazată pe stări

- generează caiete de test pe baza diagramei UML de tranzitii a stărilor asociate respectivelor
- $\text{tf stare} \rightarrow$ set reprezentativ de stările apărute tranzitile posibile
- după aplicarea fiecărui stări nu se compun stări curenții a componentelor cu ce indică diagrame
- caiete de test trebuie să aplică sevențe de stările ce aduc componente în stare doar, înainte de a se putea face o anumită transiție
- + potențial de automatizare

* Testare polymorfismului

- cum să testeze metodele care utilizează polymorfism, trebuie luate în calcul toate legăturile posibile
- ↳ necesitatea de a expune metode pentru a aplica algoritmul sănă de test