

ACADEMIA DE STUDII ECONOMICE BUCUREŞTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

PROIECTAREA SISTEMELOR INFORMATICE

-INTRODUCERE-

BUCUREŞTI
2020-2021

OBIECTIVUL GENERAL AL DISCIPLINEI



Însușirea competențelor necesare utilizării conceptelor, teoriilor, principiilor, metodelor, proceselor și instrumentelor tehnologice în scopul realizării unor sisteme informatiche prin parcurgerea tuturor etapelor specifice ciclului de dezvoltare.

OBIECTIVE SPECIFICE ALE DISCIPLINEI



- Însușirea etapelor de realizare a unui sistem informatic.
- Identificarea, interpretarea și modelarea cerințelor pentru proiectare și dezvoltarea de noi sisteme informatiche.
- Folosirea noțiunilor economice în soluționarea de probleme prin dezvoltarea de subsisteme informatiche noi sau sisteme informatiche în organizație.
- Utilizarea metodelor de analiză și proiectare specifice dezvoltării de sisteme informatiche.
- Definirea cerințelor și caracteristicilor de actualizare a sistemelor informatiche din organizație.
- Elaborarea de studii de specificații pentru proiectarea și realizarea de componente ale sistemelor informatiche.

NECESITATE

Disciplină integratoare

Oferă o viziune de ansamblu asupra procesului de dezvoltare software

Ajută la clarificarea și înțelegerea etapelor necesare pentru realizarea unui sistem informatic

Folosește standarde în domeniu

Acoperă o varietate de activități specifice dezvoltării de software

Asigură competențe necesare pentru absolvenții cu specializarea în informatică

ACTIVITĂȚI ÎN DEZVOLTAREA DE SOFTWARE



Analizează nevoile utilizatorilor și apoi proiectează, testează și dezvoltă software pentru a satisface aceste nevoi

Creează o varietate de modele și diagrame care arată ce trebuie să facă sistemul

Recomandă actualizări pentru programele și sistemele existente ale clienților

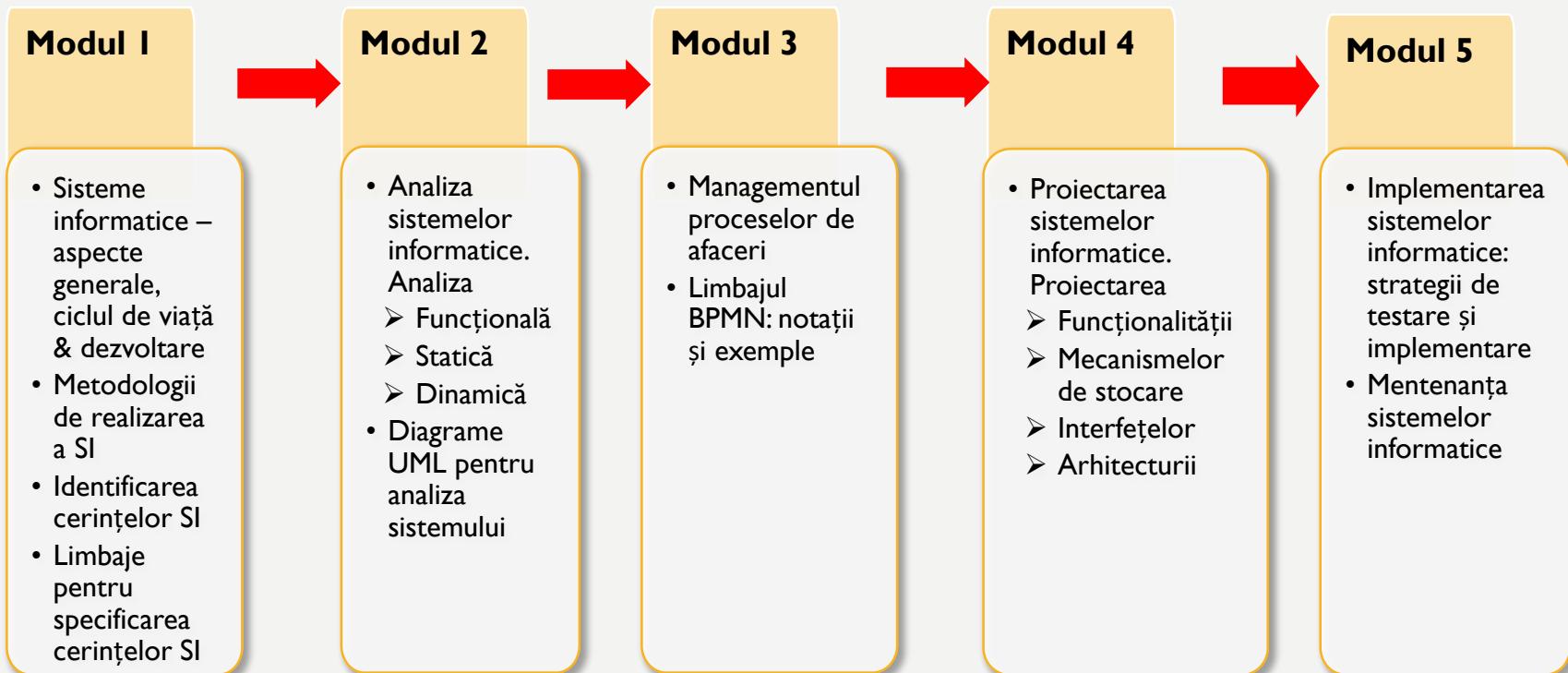
Proiectează fiecare componentă a unei aplicații sau a unui sistem și planifică modul în care componentele vor funcționa împreună

Asigură că un sistem continuă să funcționeze normal prin întreținerea și testarea software-ului

Documentează fiecare aspect al unei aplicații sau al unui sistem ca referință pentru întreținerea și actualizările viitoare

Colaborează cu alți specialiști în informatică pentru a crea software de calitate

STRUCTURA CURSULUI





MODALITATE EVALUARE

- Examen final – 50%
 - Subiecte tip grilă
- Seminar – 50%
- **Condiții de intrare în examen:**
 - susținerea proiectului, la seminar
 - minim nota 5 la seminar
- **Condiție de promovare a examenului:** minim nota 5
- **REEXAMINARE:** se susține examen + se reface proba de seminar nesusținute
- Punctaj **BONUS** – maxim 1,5p la nota de examen
 - 3 teste grile – la curs
 - 1 test = 0, 5p bonus
 - Punctajul bonus se obține pentru scor minim 70% per test.



RESURSE

- <https://online.ase.ro/>
- Ion Lungu, Gheorghe Sabău, Manole Velicanu, Mihaela Muntean, Simona Ionescu, Elena Posdarie, Daniela Sandu - *Sisteme informaticе. Analiză, proiectare, implementare*, Ed. Economică, Bucuresti, 2003
- Anca Andreeșcu - *Dezvoltarea sistemelor software pentru managementul afacerilor*, Ed. ASE, 2010
- A. Dennis, B. H. Wixom and D. Tegarden - Systems analysis and design: An object-oriented approach with UML, John Wiley & Sons, 2015. Disponibil la:
<http://www.arxen.com/descargas/PulzarCloud/Books/systems-analysis-and-design-with-uml-5th-edition.pdf>
- Specificațiile OMG pentru UML. Disponibil la
<https://www.omg.org/spec/UML/2.5.1/PDF/>
- Specificațiile OMG pentru BPMN. Disponibil la
<https://www.omg.org/spec/BPMN/2.0/PDF>

CADRE DIDACTICE TITULARE



- Conf. univ. dr. Alexandra CORBEA (FLOREA)
 - alexandra.corbea@ie.ase.ro
- Conf. univ. dr. Anca ANDREESCU
 - anca.andreescu@ie.ase.ro
- Prof. univ. dr. Ramona BOLOGA
 - ramona.bologa@ie.ase.ro
- Prof. univ. dr. Ion LUNGU
 - ion.lungu@ie.ase.ro



CUPRINS

Sistemul și componentele sale

Piramida datelor

Sisteme informatice

Ciclul de viață și de dezvoltare

Strategii de informatizare

CONCEPTUL DE SISTEM

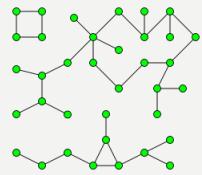


- Sistemul este un fenomen **omniprezent**.
- Putem spune că întâlnim sisteme **peste tot**.
- Dacă ceva este **organizat sistematic**, atunci se așteaptă să dea **rezultate calitative**.
- Cu toții am văzut și am experimentat sisteme din **viața reală**: sistemul educațional, sistemul politic, sistemul pentru prognoză meteo, sisteme pentru rezervări de bilete, sisteme de admitere în învățământ.

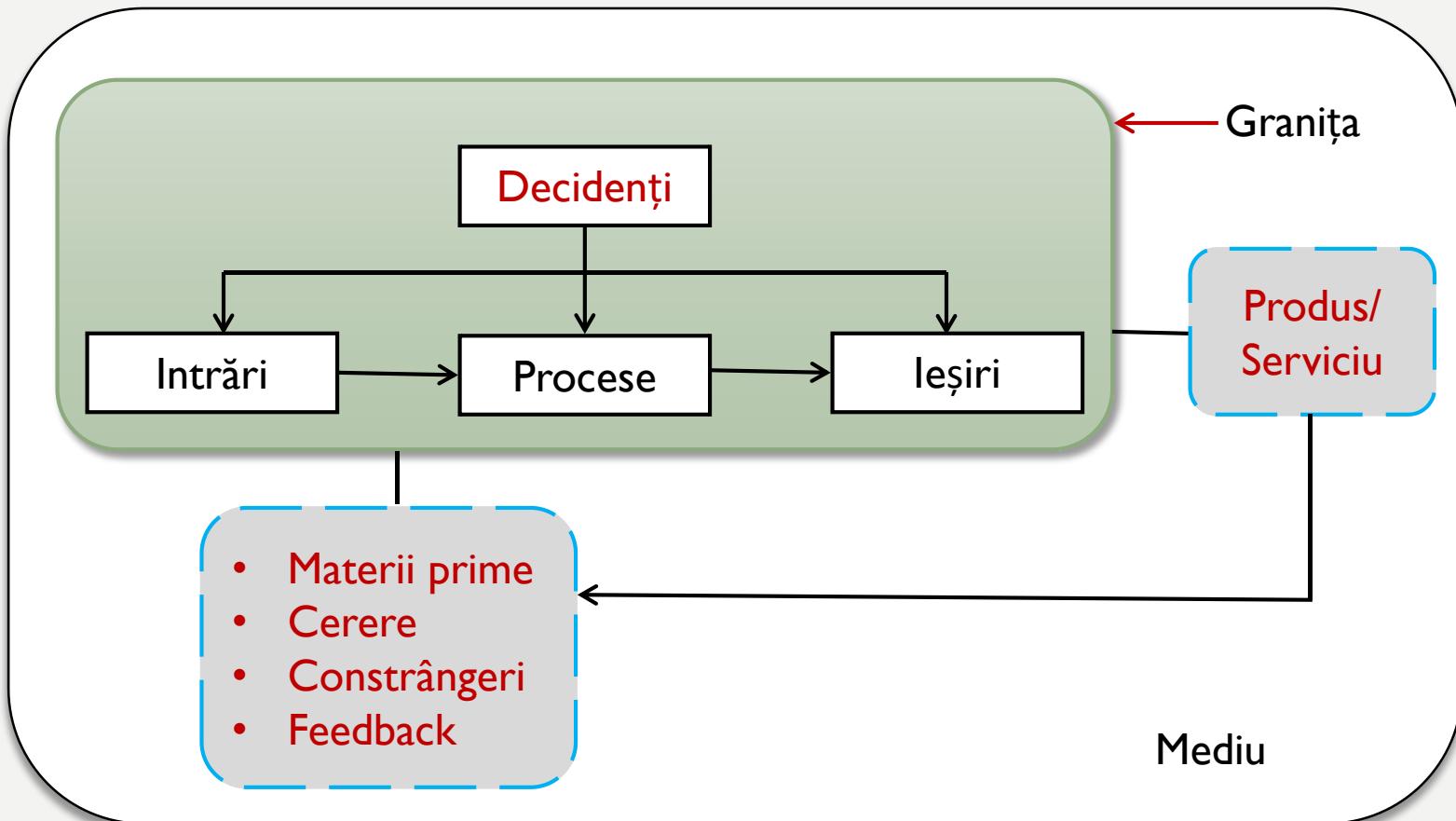
CONCEPTUL DE SISTEM

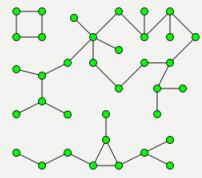


- Un sistem este un ansamblu de **componente intercorelate funcțional** care lucrează împreună pentru a realiza anumite obiective.
- Pentru sistemele informaticе, aceste obiective țin de **automatizarea obținerii informațiilor** necesare managementului în procesul de fundamentare și elaborare a deciziilor.



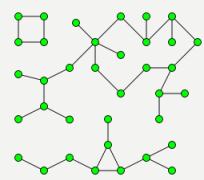
SISTEMUL ȘI COMPOENELE SALE





SISTEMUL ȘI COMPOENELE SALE

- **Componentele tipice** ale unui sistem sunt:
 - ✓ intrările
 - ✓ procesele (sau procesarea)
 - ✓ ieșirile
 - ✓ decidenții
- **Granița** este conceptul care separă sistemul de mediul său. Uneori, pentru un sistem, granița acționează ca o interfață cu mediul în care acesta operează.
- Un sistem care interacționează cu mediul său se numește **sistem deschis**.
- În contrast, un sistem care nu interacționează cu mediul său se numește **sistem închis**. Sistemele închise funcționează ca un fel de cutie neagră sau recipient izolat. Din moment ce nu acceptă intrări din mediul său și nu returnează nimic mediului, un sistem închis nu poate fi reprezentativ pentru o organizație reală.
- **Majoritatea** sistemelor din viața reală sunt sisteme deschise.

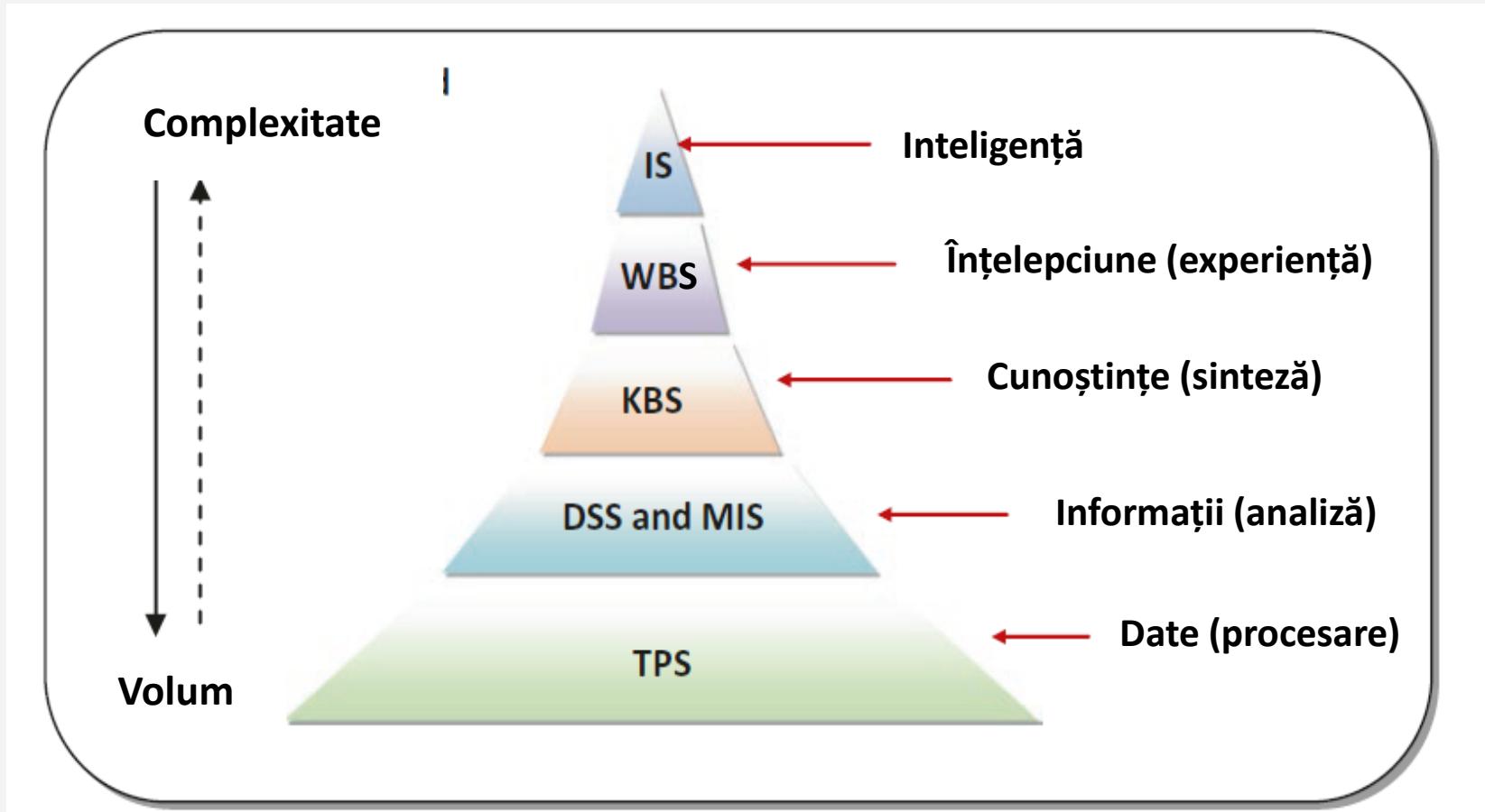


SISTEMUL ȘI COMPONELELE SALE

- **Intrările** includ acele elemente și informații care sunt furnizate sistemului. Intrările tipice pentru un sistem pot fi materii prime, informații legate de cerere de produse/servicii, reguli impuse de guvern sau autorități, feedback-ul clientilor și constrângerile de mediu sau specifice domeniului.
- **Procesarea** reprezintă un set de procese responsabile, în principal, de prelucrarea și **transformarea intrărilor în ieșiri**. Această componentă cuprinde principalele proceduri necesare pentru a obține produsele sau serviciile dorite.
- **Ieșirile** reprezintă cantitatea de informații, produsele sau serviciile (finalizate sau semi-finalizate), tangibile sau intangibile obținute de sistem după procesarea intrărilor. Ieșirile sunt destinate utilizatorilor sistemelor.
- **Feedback-ul** reprezintă informațiile furnizate înapoi în cadrul sistemului. De obicei, feedback-ul provine din mediu în cazul unui sistem deschis. Există două tipuri de feedback: feedback **pozitiv și negativ**, ambele putând să ducă la îmbunătățirea sistemului.



PIRAMIDA DATELOR



Piramida DIKW – Data Information Knowledge Wisdom

TPS - Transaction Processing Systems, DSS - Decision Support Systems,

MIS- Management Information Systems, KBS – Knowledge Based Systems,

WBS – Wisdom Based Systems, IS – Intelligent Systems



PIRAMIDA DATELOR

- **Piramida datelor** este un model care cuprinde diferite niveluri ierarhice de date, informații, cunoștințe, înțelepciune și inteligență, împreună cu tipurile de sisteme asociate acestora.
- **Datele** sunt entități elementare care formează baza piramidei. Datele sunt definite sub formă de observații brute.
- Odată ce datele sunt colectate și procesate, vor fi generate **informațiile**. Prelucrarea datelor este operația cheie care transformă datele în informații. Informațiile au asociate un factor de utilizare și de semnificație.
- **Cunoștințele** reprezintă colectarea adecvată de informații cu intenția de a deveni utile.
- Cunoștințele dobândite sunt evaluate în termeni de timp, experiență și etică pentru a genera **înțelepciune**.
- **Inteligența** se bazează pe abilitatea de a folosi cunoștințele și înțelepciunea dobândite.



PIRAMIDA DATELOR

Date: 7

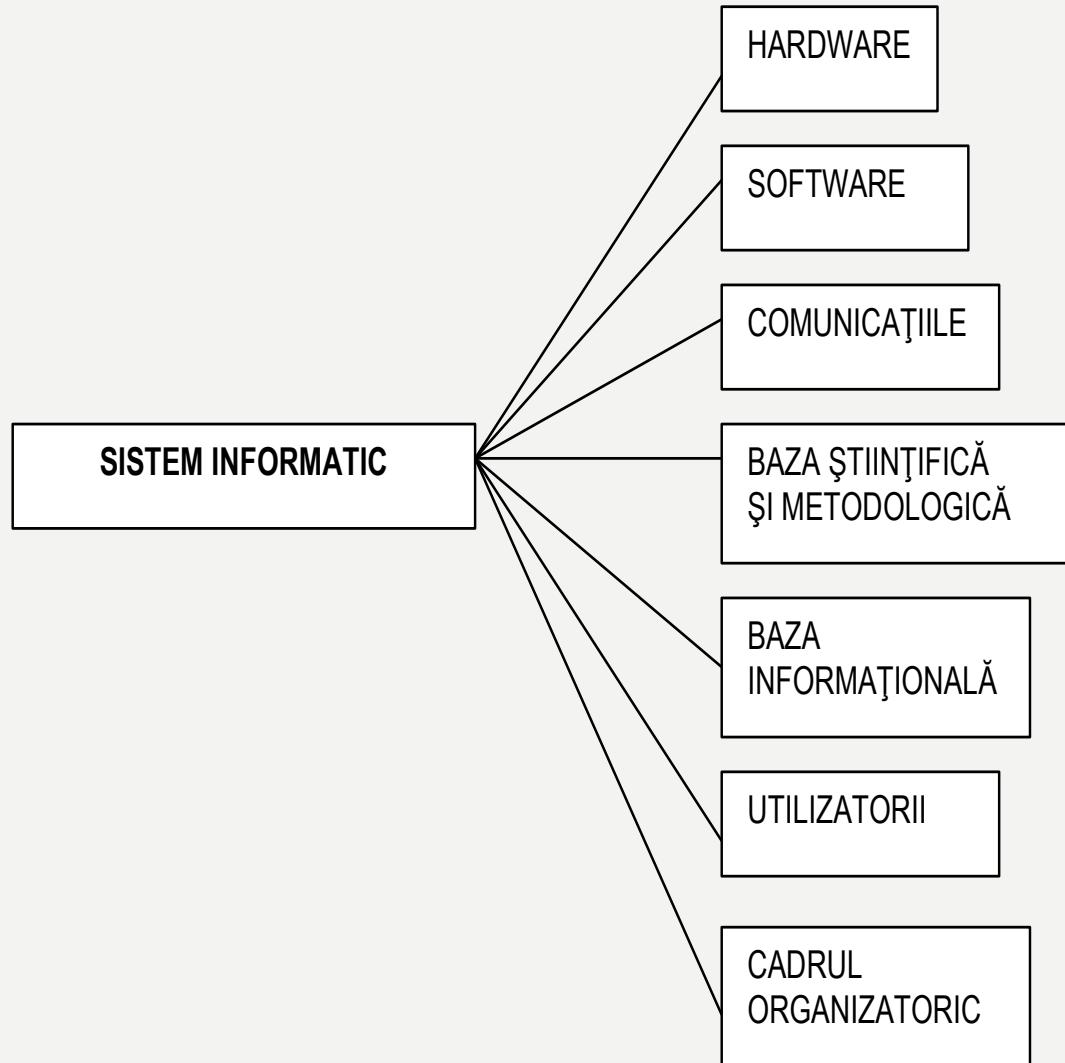
Informații: 7 grade C

Cunoștințe: 7 grade C, astăzi la ora 10.00 în București

Înțelepciune: Este nevoie să îmi pun o haină groasă când ies din casă

Inteligentă: Regulile implementate de o aplicație mobilă care mă sfătuiește ce anume să port în funcție de condițiile meteorologice prognozate (temperatură, umiditate, precipitații, vânt)

COMPONENTELE SISTEMULUI INFORMATIC



COMPONENTELE SISTEMULUI INFORMATIC

- **HARDWARE-ul** sistemului informatic este constituit din totalitatea mijloacelor tehnice de culegere, transmitere, stocare și prelucrare automată a datelor.
- **SOFTWARE-ul** sistemului cuprinde totalitatea programelor pentru funcționarea sistemului informatic, în concordanță cu funcțiile și obiectivele ce i-au fost stabilite.
- **COMUNICAȚIILE** se referă la totalitatea echipamentelor și tehnologiilor de comunicație a datelor între sisteme.
- **BAZA ȘTIINȚIFICO-METOLOGICĂ** este constituită din modele ale proceselor și fenomenelor economice, metodologii, metode și tehnici de realizare a sistemelor informatiche.
- **BAZA INFORMATIIONALĂ** cuprinde datele supuse prelucrării, fluxurile de date și nomenclatoarele de coduri.
- **UTILIZATORII** reprezintă personalul necesar funcționării sistemului informatic: utilizatori finali, analiști, proiectanți, programatori, testeri etc.
- **CADRUL ORGANIZATORIC** include regulamentul de organizare și funcționare a organizației în care funcționează sistemul informatic.

OBIECTIVE ALE SISTEMELOR INFORMATICE



- Obiective ce **afectează activitățile de bază** din cadrul organizațiilor economice, cum ar fi:
 - ✓ creșterea gradului de încărcare a capacitaților de producție existente și reducerea duratei ciclului de fabricație;
 - ✓ creșterea volumului producției;
 - ✓ optimizarea stocurilor;
 - ✓ creșterea profitului.
- Obiective ce **afectează funcționarea sistemului**, cum ar fi:
 - ✓ creșterea vitezei de răspuns a sistemului la solicitările beneficiarilor;
 - ✓ creșterea exactității și preciziei în procesul de prelucrare a datelor și informare a conducerii.

AVANTAJE OFERITE DE SISTEMELE INFORMATICE

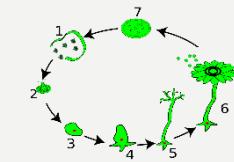


- Oferă posibilitatea **simulării facile** a proceselor și fenomenelor economice la nivel micro și macroeconomic.
- Asigură o **corelare** mai judicioasă a obiectivelor cu resursele.
- Prin implementarea unor modele matematice în cadrul sistemelor informatiche apare posibilitatea **alegerii ofertei (variantei) optime** în diferite domenii de activitate.
- Sistemele informatiche imprimă **valențe sporite de ordin cantitativ și calitativ** informațiilor furnizate decidenților sub aspectul exactității, realității, oportunității, viteze de răspuns, formei de prezentare, completitudinii informației și costului informației.

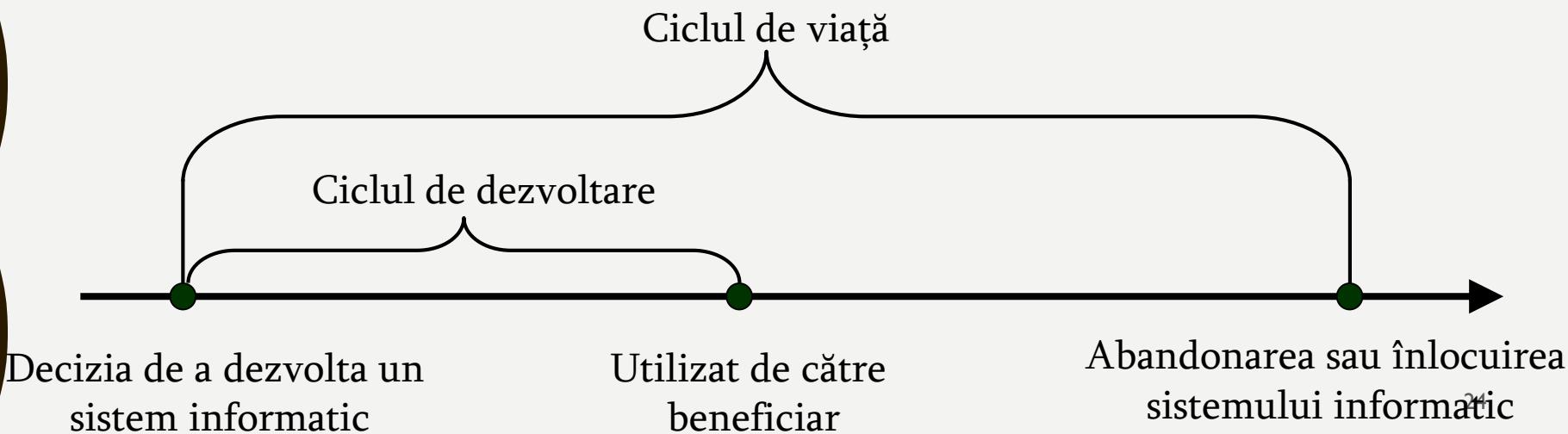
CONSTRUCȚIE/ DEZVOLTARE



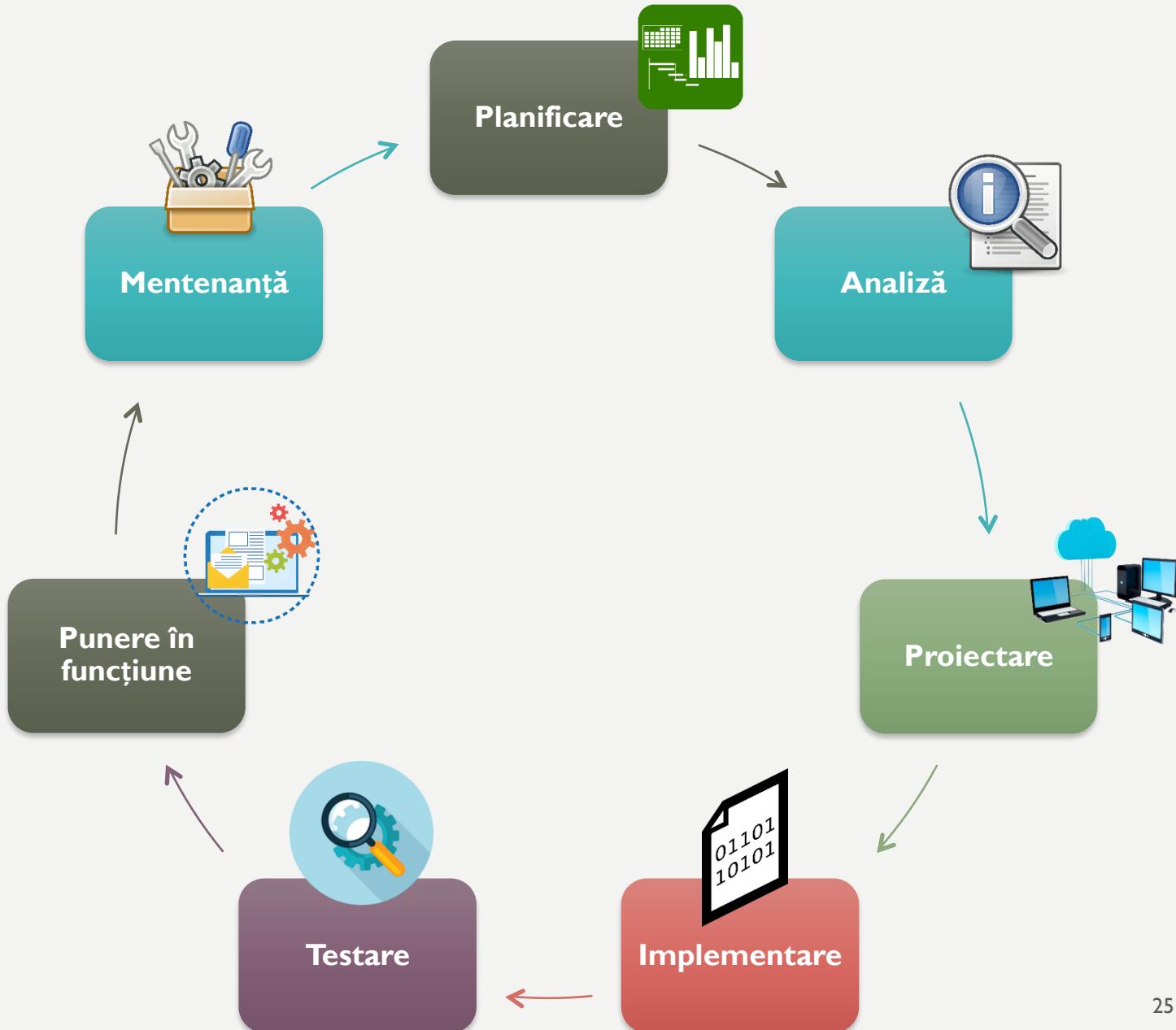
CICLUL DE VIAȚĂ ȘI DE DEZVOLTARE



- **Ciclul de viață al unui sistem informatic** este un şablon pentru ordonarea activităţilor de realizare a sistemului informatic, cuprinzând intervalul de timp care începe cu decizia de elaborare a unui sistem informatic şi se încheie cu decizia de abandonare a acestuia şi înlocuirea lui cu un nou sistem informatic.
- **Ciclul de dezvoltare al sistemului informatic** este cuprins în ciclul de viață al sistemului informatic. El include intervalul de timp de la luarea deciziei de realizare a unui sistem informatic până în momentul intrării sistemului în exploatare.



CICLUL DE VIAȚĂ ȘI DE DEZVOLTARE



CICLUL DE VIAȚĂ ȘI DE DEZVOLTARE



Activitățile componente ale ciclul de viață al sistemelor informatic sunt grupate în mai multe moduri, în etape sau faze.

1. Planificarea

- Este fundamentală pentru a înțelege de ce un sistem informatic ar trebui să fie construit și pentru a stabili modul în care echipa de dezvoltare va planifica construirea acestuia.
- Include ca faze distințe:

A. Stabilirea oportunității:

- Identifică aspecte care necesită automatizare și explică modul în care un sistem va susține aceste nevoi și va genera valoare pentru companie.
- Se identifică și se formulează cerințele globale privind realizarea sistemului informatic.
- Se pot realiza analize de tipul cost/beneficiu sau studii de fezabilitate (Cum va reduce costurile sau va crește veniturile noul sistem?).
- În final, se decide dacă dezvoltarea sistemului este oportună pentru companie.

B. Inițierea managementului proiectului:

- După aprobare, are loc demararea activităților specifice managementului proiectului.
- Rezultatul este un plan de proiect, care descrie modul în care echipa se va ocupa de realizarea sistemului.



CICLUL DE VIAȚĂ ȘI DE DEZVOLTARE

2. Analiza sistemului este etapa în care:

- are loc investigarea sistemului/sistemelor actuale și identificarea oportunităților de îmbunătățire;
- se realizează identificarea cerințelor folosind tehnici precum interviurile sau chestionarele;
- se analizează cerințele funcționale și de calitate ale sistemului;
- se identifică, printre altele: ce funcții trebuie să îndeplinească sistemul, ce date trebuie prelucrate, ce rezultate trebuie să se obțină, ce tip de interfață va fi utilizată;
- se răspunde, în esență, la întrebările „**Cine va folosi sistemul?**”, „**Ce trebuie să facă sistemul?**”, „**Unde și când va fi folosit?**” ;
- nu trebuie să se țină cont de tehnologia care va fi aleasă pentru implementare;
- folosind diferite modalități de reprezentare textuală și grafică are loc dezvoltarea unui concept pentru noul sistem;
- calitatea rezultatelor acestei etape este deosebit de importantă, deoarece acestea reprezintă o punte de legătură între cerințele clientilor și modelele arhitecturale și de implementare care se vor realiza în etapele ulterioare.

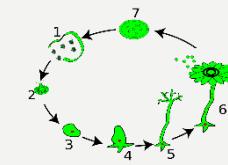
CICLUL DE VIAȚĂ ȘI DE DEZVOLTARE



3. Proiectarea sistemului este etapa care:

- răspunde, în esență, la întrebarea “**Cum vor fi realizate cerințele identificate în analiză?**”;
- decide cum va funcționa sistemul, în ceea ce privește componentele hardware, software și infrastructură de rețea;
- ia în considerare particularitățile tehnologiilor alese pentru implementare;
- în cele mai multe cazuri, sistemul va extinde sau va modifica infrastructura care există deja în companie;
- are în vedere o multitudine de aspecte tehnice, cum ar fi:
 - modularizarea și stabilirea arhitecturii sistemului;
 - modul de organizare și structurare a datelor;
 - proiectarea algoritmilor necesari pentru prelucrări;
 - proiectarea în detaliu a interfeței cu utilizatorul.
- livrabilele reprezentând modelul de proiectare al sistemului sunt predate echipei de programare pentru implementare.

CICLUL DE VIAȚĂ ȘI DE DEZVOLTARE



4. Implementarea sistemului:

- implică realizarea efectivă a aplicațiilor conform specificațiilor din etapa de proiectare;
- include activități specifice precum scrierea de cod, reutilizarea de cod, folosirea de instrumente de tipul IDE etc.

5. Testarea sistemului:

- pornește de la premisa că va implementa și se va testa separat fiecare modul al sistemului, iar integrarea și testarea de ansamblu presupune ca modulele implementate și testate în etapa anterioară să se integreze;
- ulterior, se testează sistemul în ansamblu pentru a verifica corectitudinea implementării relațiilor dintre module și funcționalitatea sistemului în ansamblu.

6. Punerea în funcțiune a sistemului:

- presupune instalarea sistemului și instruirea utilizatorilor;
- experimentarea în condiții reale este deosebit de importantă deoarece sistemul este validat folosind seturi de date reale și în condiții reale de funcționare.

7. Exploatarea și menținerea sistemului:

- pot duce la identificarea de cerințe incomplete sau incorecte, erori în sistem, posibilități de extindere etc.

STRATEGII DE INFORMATIZARE



- Strategiile, abordările și tehnicele de dezvoltare a sistemelor informatice s-au **reînnoit și perfecționat** în mod continuu.
- La început, dezvoltarea sistemelor informatice se concentra doar pe **utilizarea bazelor de date și a limbajelor de programare**.
- Treptat, **componentele și pachetele software comercializate**, precum și sistemele integrate de tip ERP realizate de producătorii de software și-au facut simțită prezența din ce în ce mai mult pe piață, oferind organizațiilor o alternativă la dezvoltarea integrală, de la zero, a sistemelor informatice.
- Recent, prin oferirea de **software sub formă de servicii** prin intermediul Internet-ului (SaaS – Software as a Service), organizațiile pot folosi software fără a avea instalate propriile aplicații.

STRATEGII DE INFORMATIZARE



- Strategiile, abordările și tehnicele de dezvoltare a sistemelor informatice s-au **reînnoit și perfecționat** în mod continuu.
- La început, dezvoltarea sistemelor informatice se concentra doar pe **utilizarea bazelor de date și a limbajelor de programare**.
- Treptat, **componentele și pachetele software comercializate**, precum și sistemele integrate de tip ERP realizate de producătorii de software și-au facut simțită prezența din ce în ce mai mult pe piață, oferind organizațiilor o alternativă la dezvoltarea integrală, de la zero, a sistemelor informatice.
- Recent, prin oferirea de **software sub formă de servicii** prin intermediul Internet-ului (SaaS – Software as a Service), organizațiile pot folosi software fără a avea instalate propriile aplicații.

STRATEGII DE INFORMATIZARE



- Există două strategii principale în informatizarea activităților organizațiilor :

1. **Achiziția sistemului**
2. **Construcția sistemului**

- în interiorul organizației
- externalizarea – de către un furnizor extern de servicii software

1. Achiziția Sistemului

- Este prima strategie de dezvoltare a unui sistem informatic ce trebuie luată în calcul.
- Presupune utilizarea de către organizație a unor produse existente, cu posibilitatea configurației și personalizării .
- Categorii de produse existente: pachete software comerciale, sisteme integrate de tip ERP, SaaS.

STRATEGII DE INFORMATIZARE



Pachete software comerciale

- Sunt disponibile pentru vânzare sau închiriere către publicul general
- Se adresează, în general, organizațiilor mici și mijlocii
- Deseori, au o capacitate limitată de personalizare pentru nevoi speciale

Sisteme integrate de tip ERP

- Facilizează integrarea tuturor proceselor de afaceri din unitățile departamentale ale organizației și gestionează conexiunile cu organizațiile externe
- Operează în timp real
- Au o bază de date comună pentru toate aplicațiile
- Sunt formate dintr-un set de module care pot funcționa și independent
- Implică un efort semnificativ de configurare și personalizare a soluției
- Se adresează tuturor tipurilor de organizații

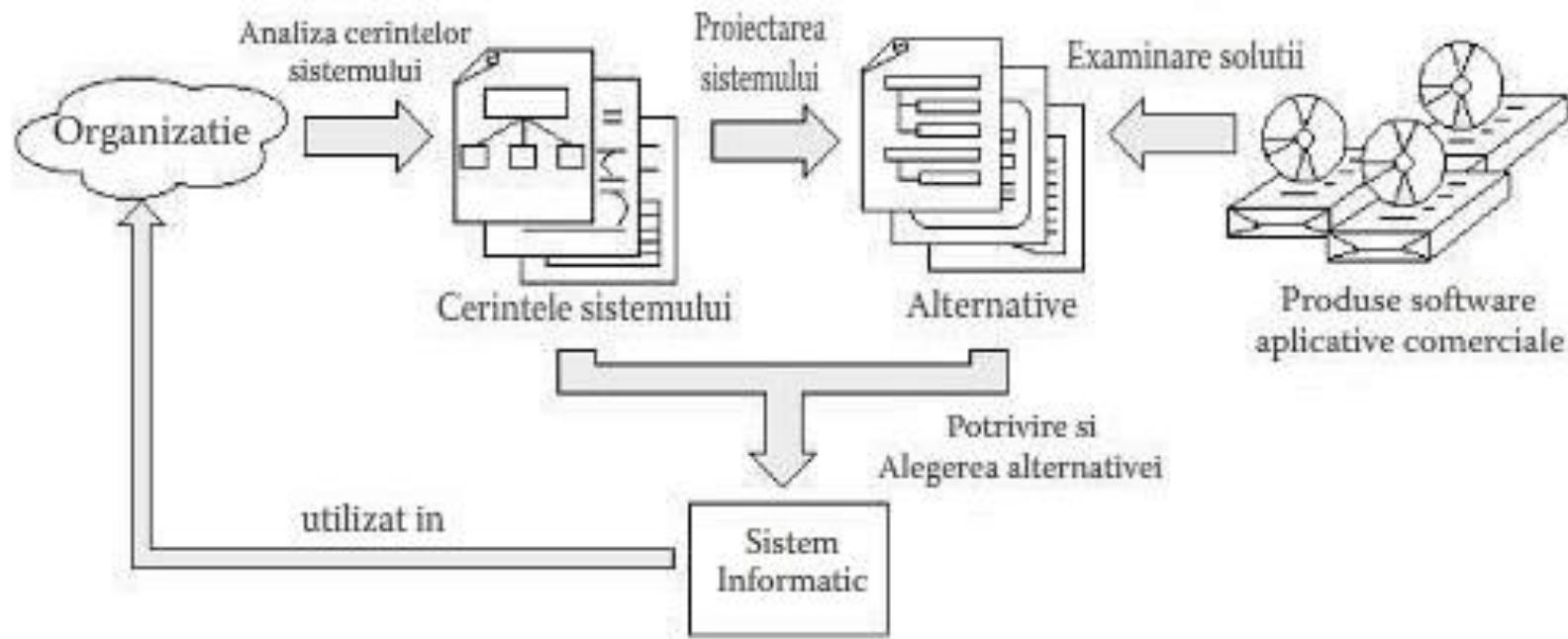
STRATEGII DE INFORMATIZARE



Software as a Service (SaaS)

- Constituie o modalitate de a oferi software în care aplicațiile și datele asociate lor sunt stocate centralizat de către furnizorul de servicii și sunt, tipic, accesate de către clienți prin intermediul Internet-ului, folosind un browser Web
- Pot suporta configurație, mai puțin personalizare
- Pot fi rapid actualizate
- Multe aplicații oferă utilizatorilor funcții de colaborare și partajare de informații
- Sunt găzduite în cloud, de aceea, timpul de răspuns și problemele de securitate constituie factori critici

STRATEGII DE INFORMATIZARE



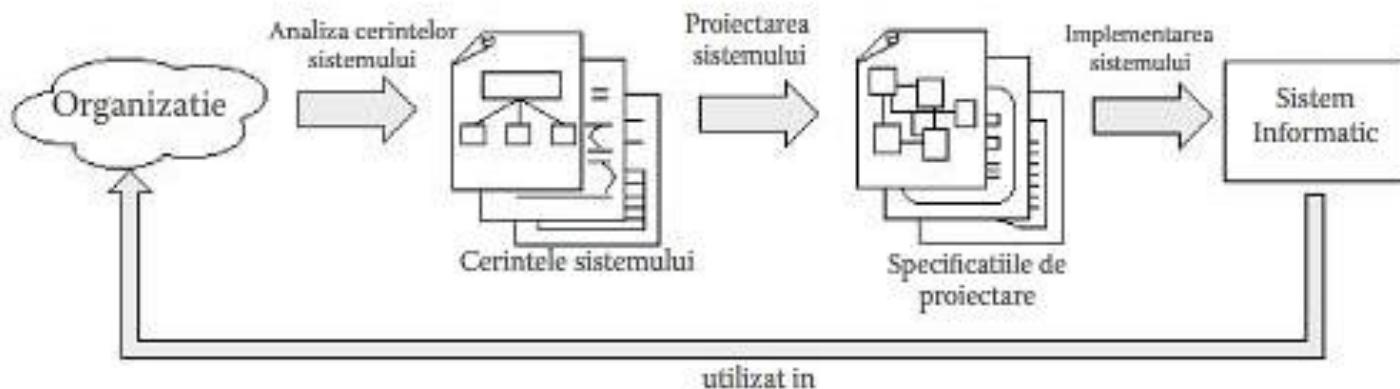
Informatizare prin achiziția de software

STRATEGII DE INFORMATIZARE



2. Construcția Sistemului

- Poate fi realizată în interiorul organizației sau externalizată
- Se folosește, spre exemplu, pentru cerințe specifice unice ale organizației
- Este metoda adoptată de către dezvoltatorii de software și de tehnologii informatiche
- Este o soluție consumatoare de timp și resurse
- Implică parcurgerea tuturor pașilor specifici ciclului de dezvoltare a unui sistem informatic



Informatizare prin realizarea de software

CURSUL 2...

- Identificarea cerințelor sistemelor informaticе
- Limbaje pentru specificarea cerințelor sistemelor informaticе

ACADEMIA DE STUDII ECONOMICE BUCUREŞTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

PROIECTAREA SISTEMELOR INFORMATICE

- CURS 2 -

BUCUREŞTI
2020-2021

IDENTIFICAREA CERINȚELOR SISTEMELOR INFORMATICE

Cuprins

- ✓ Problematica ingineriei cerințelor
- ✓ Categorii de cerințe ale sistemelor informaticice
- ✓ Cerințe non-funcționale
- ✓ Tehnici pentru identificarea cerințelor
- ✓ Caracteristici de calitate ale cerințelor



PREMIZE

- Slaba calitate a cerințelor este în mod constant **pe primele locuri** în ierarhia cauzelor care duc la eșecul proiectelor software.
- O explicație ar fi aceea că echipele de dezvoltare **alocă prea puțin timp** înțelegерii problemelor reale ale afacerii, a nevoilor utilizatorilor sau a naturii mediului în care va rula sistemul.
- De asemenea, dezvoltatorii încearcă să furnizeze **cât mai rapid** soluții tehnice, plecând însă de la **înțelegerea insuficientă** a cerințelor problemei.

IDENTIFICAREA CERINȚELOR

- Reprezintă procesul prin care **culegem** și **documentăm** cerințele sistemului.
- Identificarea și definirea detaliată a cerințelor se realizează de **comun acord** cu beneficiarul.
- Pot fi identificate:
 - ✓ Cerințe care **rezolvă** sau **reduc** deficiențele sistemului existent.
 - ✓ Cerințe care exprimă **noi facilități**. Ele nu sunt asigurate de vechea implementare a sistemului.

IDENTIFICAREA CERINȚELOR

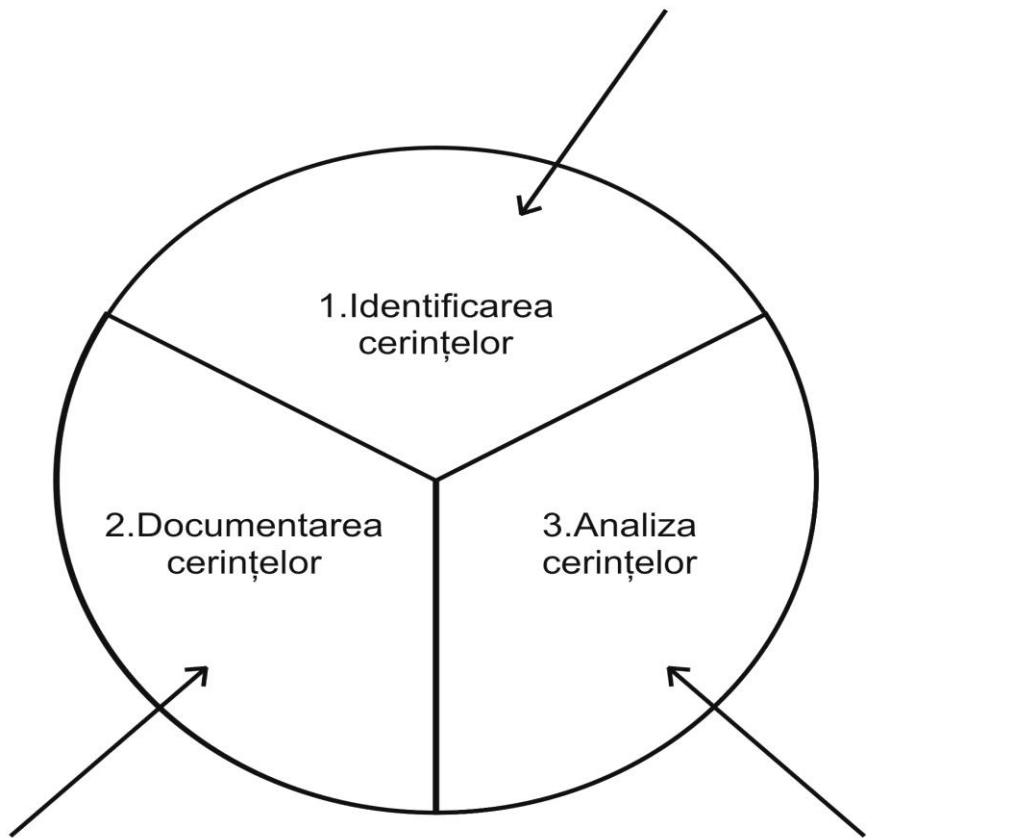
- Aspecte practice:
 - Cu cât descoperim **mai multe** cerințe, cu atât va fi **mai bun** sistemul construit.
 - Cerințele se **schimbă**.
 - Pot exista cerințe care sunt **deja implementate**.
 - Cerințele pot fi:
 - ✓ **explicite** – furnizate de beneficiar;
 - ✓ **implicite** – le descoperă analistul.

PROBLEMATICA INGINERIEI CERINȚELOR

- Pentru a fi siguri că un sistem informatic rezolvă în mod corect o anumită problemă, trebuie, mai întâi, să se înțeleagă în mod corect problema care trebuie rezolvată. În acest sens sunt necesare descoperirea, înțelegerea, specificarea și analiza următoarelor componente:
 - **CE** problemă trebuie rezolvată;
 - **DE CE** o astfel de problemă necesită rezolvare;
 - **CINE** este implicat și va fi responsabil de rezolvarea acestei probleme.
- În mare, aceste trei componente, stau la baza *Ingineriei Cerințelor Sistemelor Informaticice*.

PROBLEMATICA INGINERIEI CERINȚELOR

Cum comunicăm cu beneficiarii și utilizatorii pentru a stabili cerințele lor?



Cum documentăm cerințele
într-o manieră textuală sau
grafică?

Sunt cerințele neclare, incomplete
redundante sau contradictorii?
Cum rezolvăm aceste probleme?

Activități de bază pentru construirea unui model al cerințelor

PROBLEMATICA INGINERIEI CERINȚELOR

- Deseori, dificultățile în modelarea și analiza cerințelor provin din **înțelegerea insuficientă a părții de logică a sistemului/aplicatiei**, cunoscută și sub denumirea de logică a afacerii.
- *Logica afacerii* reprezintă elementul definitoar pentru o afacere aflată în proces de modelare și de automatizare, ea incluzând atât **regulile afacerii**, cât și **fluxul de lucru (procesele)** din cadrul afacerii prin care se descrie modul de transfer al documentelor sau al datelor de la un participant (persoană fizică sau sistem software) la altul.
- În dezvoltarea de sisteme informatici, logica afacerii are ca obiective:
 - modelarea **obiectelor afacerii** din lumea reală (precum stocuri, clienți, produse);
 - gestionarea **modului de stocare a obiectelor afacerii** (maparea obiectelor afacerii în tabelele bazei de date);
 - descrierea modului în care **obiectele afacerii interacționează între ele**.

CATEGORII DE CERINȚE ALE SISTEMELOR INFORMATICE

- În dezvoltarea sistemelor informaticice, o cerință **definește un deziderat** pe care acesta trebuie să-l îndeplinească, ca răspuns la necesitățile utilizatorilor săi.
- Ea reprezintă **o parte a scopului** pentru care se dezvoltă un proiect informatic.
- Cerințele dictează și modul în care **sistemul trebuie să răspundă la interacțiunea cu utilizatorul**.
- În linii mari, dezvoltatorii trebuie să aibă în vedere următoarele aspecte legate de cerințele unui sistem:
 - folosirea unui **limbaj tehnic**: cerințele trebuie întotdeauna specificate în limbajul utilizatorului, putând include aici și jargonul domeniului analizat;
 - **relația cu obiectivele afacerii**: fiecare cerință trebuie în mod clar legată de obiectivele oamenilor de afaceri.

CATEGORII DE CERINȚE ALE SISTEMELOR INFORMATICE

- Prin procesul de inginerie a cerințelor se urmărește culegerea, elaborarea, corectarea și adaptarea **unui volum mare de specificații**, care **diferă în ceea ce privește scopul și modul de exprimare**. O astfel de diferențiere se poate realiza între cerințele:
 - *descriptive*;
 - *prescriptive*.
- Specificațiile *descriptive* prezintă proprietăți pe care sistemul trebuie să le aibă **indiferent de modul în care acesta va funcționa**. Astfel de proprietăți sunt generate, de obicei, de legi ale naturii sau de constrângeri fizice. Exemple de specificații descriptive:
 - Aceeași carte nu poate fi împrumutată de doi abonați în același timp.
 - Dacă o cameră a hotelului este în renovare, atunci aceasta nu poate fi ocupată.

CATEGORII DE CERINȚE ALE SISTEMELOR INFORMATICE

- Specificațiile *prescriptive* descriu proprietățile pe care se dorește să le aibă sistemul informatic și pe care acesta le îndeplinește sau nu, acest lucru depinzând de modul în care sistemul va funcționa

Exemple de specificații prescriptive sunt:

- Un abonat nu poate să împrumute mai mult de trei cărți în același timp.
 - Un produs trebuie livrat în intervalul orar specificat de cumpărător.
- Distincția dintre specificațiile descriptive și cele prescriptive este foarte importantă în contextul ingineriei cerințelor, în sensul că **putem negocia, schimba sau găsi alternative pentru specificațiile prescriptive**, în timp ce pentru cele descriptive aceste lucruri nu sunt posibile.

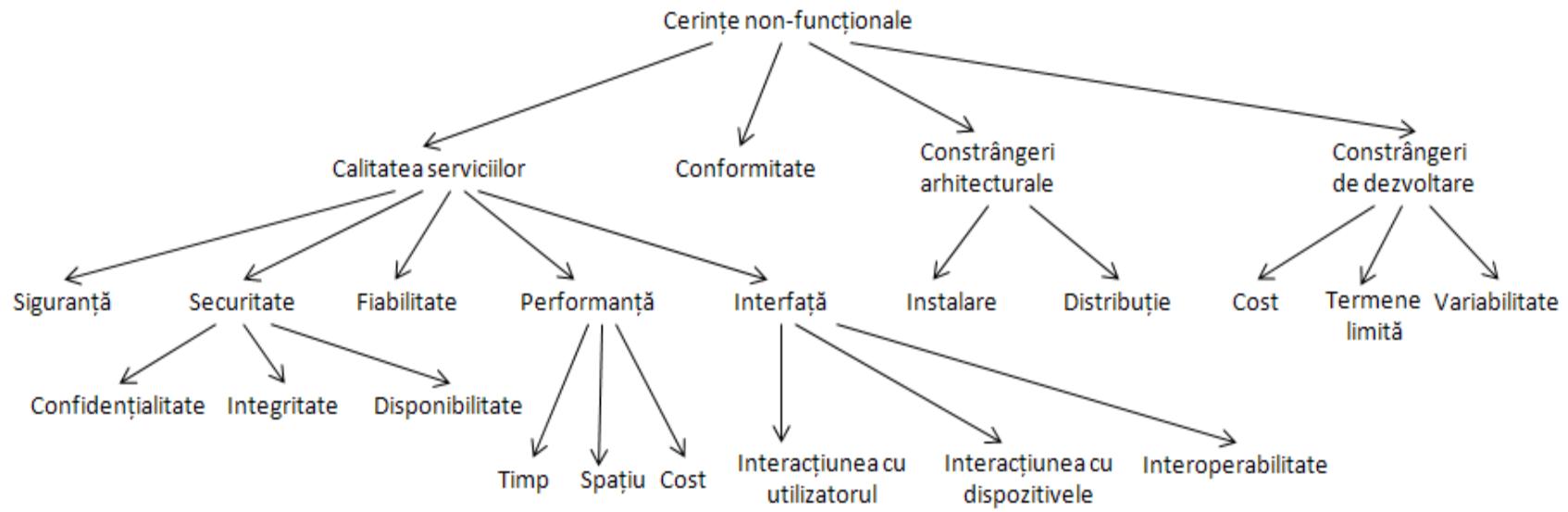
CATEGORII DE CERINȚE ALE SISTEMELOR INFORMATICE

- Prin prisma funcționalității, cerințele unui sistem sunt de două tipuri:
 - *funcționale* ;
 - *non-funcționale*.
- **Cerințele funcționale definesc funcțiile unui sistem informatic** sau ale componentelor acestuia, prin intermediul unui mulțimi de intrări, ale comportamentului și a ieșirilor. În această categorie putem încadra calcule, detalii tehnice, informații privind procesarea și manipularea datelor, precum și alte funcționalități care arată, de exemplu, cum trebuie realizat un caz de utilizare.

CATEGORII DE CERINȚE ALE SISTEMELOR INFORMATICE

- Cerințele non-funcționale surprind criteriile care pot fi folosite pentru a analiza aspectele legate de **operaționalitatea sistemului**, și nu de comportamentul acestuia.
- Acestea **impun constrângeri** la nivel de proiectare sau de implementare asupra cerințelor funcționale (de exemplu, cerințe legate de performanță, securitate sau fiabilitate). Cerințele non-funcționale sunt deseori referite prin termenul de *calități ale unui sistem*, dar pot fi menționate și ca *attribute de calitate*, *obiective de calitate*, *caracteristici de calitate* sau *constrângeri*.

CERINȚE NON-FUNCȚIONALE



Tipuri de cerințe non-funcționale

CERINȚE NON-FUNCȚIONALE

A. Cerințele de calitate definesc aspecte referitoare la “cât de bine” trebuie să funcționeze sistemul. Acestea includ specificații legate de:

- **Cerințele de siguranță** care sunt cerințe de calitate prin care se preîntâmpină producerea unor accidente sau degradarea mediului.
- **Cerințele de securitate** descriu măsuri de protecție a sistemului împotriva unor comportamente nedorite.
 - *Cerințele de confidențialitate* arată că anumite informații nu pot fi divulgăte părților neautorizate.
 - *Cerințele de integritate* arată că anumite informații pot fi modificate dacă acest lucru a fost realizat într-o manieră corectă și autorizată.
 - *Cerințele de disponibilitate* se referă la faptul că anumite cerințe sau resurse pot fi folosite de către persoanele autorizate atunci când este necesar.

CERINȚE NON-FUNCȚIONALE

A. Cerințele de calitate -continuare

- **Cerințele de fiabilitate** constrâng sistemul informatic să funcționeze aşa cum este de așteptat și de dorit, pentru perioade lungi de timp. Exceptând circumstanțele excepționale, sistemul trebuie să furnizeze servicii într-o manieră corectă și robustă.
- **Cerințele de performanță** impun condiții asupra modului în care operează sistemul, cum ar fi, spre exemplu, timpul maxim necesar pentru execuția unei operații.
- **Cerințele de interfață** arată cum interacționează sistemul cu mediul, utilizatorii, precum și cu alte sisteme.

CERINȚE NON-FUNCȚIONALE

- B. Cerințele de conformitate** impun condițiile necesare astfel încât sistemul să funcționeze în conformitate cu legile naționale, reglementările internaționale, normele sociale, constrângerile politice și culturale, standarde etc.
- C. Cerințele arhitecturale** impun constrângerile structurale asupra sistemului informatic, astfel încât acesta să poată funcționa corespunzător în mediul unde va fi implementat. Oferă dezvoltatorilor indicații privind tipul de arhitectură potrivită pentru sistem.
- D. Cerințele de dezvoltare** sunt cerințe non-funcționale care constrâng modul în care sistemul ar trebui dezvoltat, și nu felul în care acesta ar trebui să satisfacă cerințele funcționale. Sunt incluse aici cerințe privind costurile de dezvoltare, planul de livrare și instalare, detalii referitoare la menenanță, portabilitate etc.

EXEMPLE DE CERINȚE NON-FUNCȚIONALE

- *Cerințe de calitate*
 - *Siguranță*
 - Sistemul nu trebuie să mai funcționeze dacă temperatura exterioară scade sub 4 grade C.
 - Sistemul nu trebuie să mai funcționeze în caz de incendiu.
 - Sistemul nu trebuie să mai funcționeze în cazul în care se observă atacuri evidente.
 - *Securitate*
 - Permișunile de acces la date pot fi modificate numai de către administratorul de date al sistemului.
 - Toate datele de sistem trebuie să fie copiate la fiecare 24 ore și copiile de rezervă stocate într-un loc sigur, care nu se află în aceeași clădire ca și sistemul.
 - Toate comunicările externe între serverul de date și clienți trebuie să fie criptate.
 - *Fiabilitate*
 - Sistemul trebuie să aibă o disponibilitate de 999/1000 sau 99%. Aceasta este o cerință de fiabilitate care presupune ca din fiecare 1000 de cereri ale serviciului, 999 trebuie să fie îndeplinite.
 - *Performanță*
 - Sistemul va procesa un minim de 100 tranzacții pe secundă.

EXEMPLE DE CERINȚE NON-FUNCȚIONALE

- *Cerințe de conformitate*
 - *Cerințe legale și de reglementare:*
 - Toate modificările aduse datelor utilizatorilor trebuie păstrate minim 6 ani.
 - *Cerințe de licențiere:*
 - Sistemul este licențiat pentru a fi utilizat simultan de către maxim 100 de utilizatori.
 - Licențele pentru soluția de baze de date trebuie să permită lucrul cu până la 4 procesoare fără limită pentru lucrul cu memoria RAM și dimensiunea bazei de date.
- *Cerințe arhitecturale*
 - Persistența datelor va fi asigurată printr-o baze de date relațională.
 - Această baze de date va fi Oracle 18c.
 - Sistemul va rula 24 de ore pe zi, 7 zile pe săptămână.
- *Cerințe de dezvoltare*
 - Modulul de raportare inclus în soluția de bază de date trebuie să conțină un instrument de dezvoltare de rapoarte adiționale.
 - Modulul de raportare inclus în soluția de bază de date trebuie să aibă o interfață web de configurare și acces proprie.

SURSE PENTRU IDENTIFICAREA CERINȚELOR

- Cerințele le identificăm prin două modalități de bază:

1. De la **beneficiar** prin:

- Interviuri
- Ședințe comune
- Observații directe
- Prototipizare

2. Prin **consultarea artefactelor**

- Documente de intrare
- Rapoarte de ieșire
- Legislație
-

ARTEFACTE

- Artefactele conțin orice **informații existente** despre sistem care pot fi culese din diverse surse:
 - Documente preexistente despre funcționarea sistemului
 - Eșantioane de date
 - Chestionare
 - Scenarii
 - Prototipuri de interfețe
 - Scheme conceptuale
 -
- În primul rând trebuie **înteleasă organizația**, chiar și la nivel de bază:
 - Cu ce se ocupă
 - Care sunt fluxurile de lucru
 - Ce politici a adoptat
 - Cine sunt beneficiarii sistemului

TIPURI DE ARTEFACTE

- **Organizația**

- ✓ Organigrame
- ✓ Planuri de afaceri
- ✓ Politici
- ✓ Rapoarte financiare
- ✓ Procese verbale ale întâlnirilor
- ✓ Fișe de post

- **Domeniu**

- ✓ Cărți, studii, articole
- ✓ Reglementările din domeniu
- ✓ Rapoarte privind sisteme similare

- **Sistemul existent**

- ✓ Rapoarte privind fluxul de informații, proceduri de lucru, reguli de afaceri
- ✓ Rapoarte care conțin defecte, reclamații, solicitarea unor schimbări
- ✓ Manuale de utilizare

ANALIST

- Responsabilități legate de cerințe:
 - Culege
 - Documentează
 - Analizează
 - Validează
- În cadrul echipei:
 - Reprezintă legătura dintre client și dezvoltator
 - Rol important în evoluția proiectului
 - Gestionează activități în contextul schimbării cerințelor, luării unor noi decizii de dezvoltare sau apariției unor riscuri

ANALIST

- Documentează specificațiile cerințelor
 - Modelează cerințele
 - Reprezentări grafice
 - Ecuații
 - Diagrame
 - Tabele
 - Prototipuri de interfețe
- Gestionează procesul de validare a cerințelor
 - Cerințele culese îndeplinesc caracteristicile de calitate necesare pentru ca sistemul descris să satisfacă nevoile beneficiarului

PROBLEME LEGATE DE IDENTIFICARE CERINȚELOR

- ✓ Surse distribuite și conflictuale
- ✓ Dificultatea de a accesa sursele
- ✓ Cunoștințe tacite/ascunse (Cunoștințe care sunt implicate din punct de vedere al beneficiarului sau cu care sunt atât de familiarizați încât le consideră de bun simt și nu simt nevoie să dea explicații suplimentare)
- ✓ Condiții instabile
- ✓ Factori sociopolitici
- ✓ Competiții între departamente
- ✓ Competiții între indivizi/angajați
- ✓ Interese divergente
- ✓ Priorități diferite
- ✓ Documente neactualizate
- ✓ Rezistență la schimbare

ABORDĂRI TRADIȚIONALE VS. AGILE

- Abordările tradiționale

- ✓ Încercăm să documentăm cât mai multe cerințe de la început
- ✓ Planifică și documentează
- ✓ Uneori sute sau mii de pagini
- ✓ Definește totul
- ✓ Înregistrează toate presupunerile
- ✓ Explică raționamente, alternative
- ✓ De lungă durată

- Abordările agile

Agile	Tradiționale
Indiviși și interacțiuni	Procese și instrumente
Software care funcționează	Documentație exhaustivă
Colaborare cu clientul	Negocierea contractului
Reacție la schimbare	Urmărirea unui plan

TEHNICI PENTRU IDENTIFICAREA CERINȚELOR

1. Interviurile

- Interviewarea beneficiarului este **cea mai uzuală metodă** de identificare a cerințelor. Succesul său depinde de **implicarea tuturor părților interesate**, incluzând aici manageri, acționari angajați, etc. pe care în continuare îi vom plasa sub titulatura generală de utilizatori.
- Un interviu, în mod normal, se va concentra asupra **activității desfășurate** de utilizator în cadrul organizației, a modului în care implementarea sistemului **va influența munca sa și a problemelor** pe care acesta le-a identificat în procesele actuale care au loc în organizație. Interviul poate scoate la iveală **cerințe care nu au fost inițial identificate** ca făcând parte din obiectivele proiectului, dar și **cerințe contradictorii**.

TEHNICI PENTRU IDENTIFICAREA CERINȚELOR

1. Interviurile - continuare

- Apariția cerințelor contradictorii poate fi derutantă, în sensul că nu pare firesc ca diferite persoane care lucrează în aceeași organizație să aibă viziuni contradictorii asupra aceluiași aspect analizat.



Analistul își poate pune următoarea întrebare

pertinentă: *"Cum poate o afacere să fie competitivă și profitabilă dacă nu există un consens, cel puțin în interiorul ei, privind modul în care aceasta funcționează?"*



Un posibil răspuns: *"...nivelul de detaliu necesar pentru construirea unei aplicații informative este mai mare decât nivelul de detaliu necesar pentru a conduce cu succes o afacere."*

INTERVIURI – BUNE PRACTICI

- ✓ Identificați **eșantionul interviewat** potrivit pentru acoperirea completă a problemelor – nu este nevoie de mai mulți angajați care au aceleasi responsabilități sau de toți managerii.
- ✓ Centrați interviul pe **munca interviewaților** și pe preocupările acestora.
- ✓ Luați în considerare și **persoana**, nu numai rolul acesteia.
- ✓ Oferiți **motivare**, puneti întrebări cât mai ușor de înțeles.
- ✓ Fiți **deschiși, flexibili** în cazul în care apar răspunsuri neașteptate
- ✓ Puneti frecvent întrebarea “**DE CE?**” pentru a clarifica aspectele neînțelese.
- ✓ Evitați întrebările care exprimă **opinii** sau sunt **interpretabile** sau al căror răspunsuri sunt evidente sau imposibile pentru interviewat.
- ✓ **Adaptați** întrebările la **nivelul de expertiză** al fiecărui interviewat (trebuie să cunoașteți responsabilitățile acestora) .

TEHNICI PENTRU IDENTIFICAREA CERINȚELOR

2. Sesiunile comune pentru stabilirea cerințelor

- Sesiunile comune pentru stabilirea cerințelor (engl. Join Requirement Planning- JRP) pot fi asimilate cu efectuarea interviurilor tuturor utilizatorilor (sau cel puțin a unei părți semnificative a acestora) în același timp și în aceeași încăpere.
- Toate persoanele care vor influența direcția în care se dezvoltă sistemul sunt reunite într-un singur loc pentru a furniza detalii despre ceea ce trebuie să facă sistemul.
- Este necesară existența unui mediator care să modereze aceste sesiuni, dar și a unei persoane care să documenteze propunerile utilizatorilor, ajutându-se, de exemplu, de un proiectoare și de un produs software pentru modelarea vizuală.

TEHNICI PENTRU IDENTIFICAREA CERINȚELOR

3. Cazurile de utilizare

- Cazurile de utilizare descriu **interacțiunile** dintre utilizatori și sistem și reflectă ceea ce trebuie să facă utilizatorii cu sistemul prin identificarea funcțiilor importante.
- Un caz de utilizare surprinde **secvența de interacțiuni dintre sistem și actorii externi** (cum ar fi persoane, dispozitive hardware sau alte sisteme software), incluzând și variante sau extensii ale comportamentului de bază al sistemului.
- Cazurile de utilizare pot constitui una dintre **primele forme de reprezentare a cerințelor unui sistem informatic**, motiv pentru care ele sunt potrivite pentru stadiile incipiente ale procesului de dezvoltare software. Analistii, împreună cu beneficiarii sistemului, vor trebui să examineze și să valideze fiecare caz de utilizare propus

TEHNICI PENTRU IDENTIFICAREA CERINȚELOR

4. Observațiile și analizele sociale

- Metodele bazate pe observații presupun existența unui **observator** care să urmărească utilizatorii sistemului și să noteze informații referitoare la activitatea pe care aceștia o desfășoară.
- Observațiile pot fi **directe**, implicând prezența observatorului pe parcursul execuției activităților sau **indirecte**, atunci când activitatea este vizualizată prin intermediul altor mijloace, cum ar fi o înregistrare video.
- Metoda prezintă avantajul de a facilita culegerea unor date de calitate și este utilă pentru analiza activităților și proceselor reale, prin care **se pot evita eventualele omisiuni sau erori** furnizate de o descriere a fluxului de lucru de către beneficiar .

TEHNICI PENTRU IDENTIFICAREA CERINȚELOR

5. Prototipurile

- **Prototipul** este util pentru utilizatorul final care va înțelege mai bine ce vrea sau ce se așteaptă de la produs și este util dezvoltatorului pentru a testa unele tehnici, algoritmi folosiți, interfețe realizate etc.
- Referitor la prototip există două abordări :
 - **prototip de încercare** care trebuie să fie **rapid**, chiar dacă nu perfect și care poate fi utilizat pentru **validarea interfeței**, pentru particularizarea arhitecturii pentru a cuprinde cerințele cât mai bine, sau pentru a valida algoritmi particulari;
 - **prototip evolutiv** care, dimpotrivă, dezvoltă produsul final astfel încât să se poată aprecia **toate caracteristicile de calitate ale produsului software** final; în general acest prototip nu poate fi rapid, dar permite îmbunătățirea modelului prin asigurarea unei calități ridicate a produsului software.

CARACTERISTICI DE CALITATE ALE CERINȚELOR

Literatura de specialitate face referire la o multitudine de proprietăți “dorite” ale specificațiilor. În ansablu lor, se urmărește ca cerințele să fie:

- **Complete:** Documentația cerințelor include toate informațiile necesare referitoare la cerințe: cerințe funcționale și non-funcționale, constrângeri, cerințe referitoare la interfețele externe, cerințe ale datelor etc.
- **Consistente:** Între cerințele de la același nivel, nu există conflicte interne care să ducă la contradicții. De asemenea, nu trebuie să existe contradicții între cerințele de la niveluri diferite. Se va urmări și consistența terminologiei, astfel încât: a) un cuvânt să aibă același sens de fiecare dată când este folosit; b) două cuvinte diferite nu se folosesc pentru a desemna același lucru.
- **Modificabile:** Documentația care include cerințele este organizată și scrisă într-o manieră care facilitează modificările ulterioare și, în acest sens, o cerință trebuie să fie:
 - **Neredundantă:** Fiecare cerință este specificată o singură dată;
 - **Schimbabilă:** Fiecare cerință poate fi schimbată fără a avea un impact major asupra altor cerințe.

CARACTERISTICI DE CALITATE ALE CERINȚELOR

Totodată, este de dorit ca fiecare cerință individuală să fie:

- **Neambiguă:** Fiecare specificare a unei cerințe trebuie să aibă o singură interpretare și să fie descrisă într-o manieră coerentă și ușor de înțeles.
- **Concisă:** Fiecare cerință trebuie să fie scurtă, folosind un limbaj orientat spre acțiune și evitându-se detaliile inutile.
- **Măsurabilă:** Dacă este necesar, pentru fiecare cerință se precizează limite sau valori măsurabile specifice. Caracteristica este utilă în special pentru cerințele non-funcționale.
- **Fezabilă:** cerința poate fi implementată folosind tehnici, tehnologii, instrumente și resurse existente, cu ajutorul personalului disponibil și în limite cunoscute de cost și timp.
- **Testabilă:** Din perspectiva costurilor, există o modalitate acceptabilă de a stabili dacă sistemul informatic satisface acea cerință.
- **Poate fi urmărită:** Pentru fiecare cerință, pot fi identificate sursa acesteia, precum și formele sub care este reprezentată la diferite niveluri de specificare. De asemenea, cerința trebuie specificată într-o manieră care permite urmărirea acesteia în etapele următoare ale dezvoltării sistemului, și anume proiectare, implementare și testare.

ASPECTE PRACTICE

Combinarea tehnicielor

- Datorită limitărilor, o singură tehnică nu poate fi suficientă
- Sunt necesare atât identificarea pornind de la artefacte, cât și identificarea pornind de la beneficiar
- Exemple:
 - Interviuri + observații directe + prototipizarea
 - Rapid Application Development: sesiuni JAD + prototip evolutiv

Implicare

- Este necesară implicarea activă a beneficiarului, deoarece:
 - ✓ Cerințele se schimbă
 - ✓ Obținem feedback în mod frecvent
 - ✓ Rezolvăm aspecte neînțelese
 - ✓ Încercăm să eliminăm ambiguități

NU PIERDEȚI DIN VEDERE!!

- Analiza **riscurilor** - beneficiarii pot vorbi despre soluții care nu sunt fezabile datorită:
 - Mediului
 - Bugetului
 - Tehnologiei
- Cere sunt cerințele **reale** ale clientului
- **Prioritizarea** cerințelor
- Întotdeauna fii pregătit pentru **schimbare**

LIMBAJE PENTRU MODELAREA INFORMAȚIONALĂ

Cuprins

- ✓ Modelarea în realizarea sistemelor informaticice
- ✓ Limbaje pentru modelarea informațională
- ✓ Categorii de limbaje pentru modelarea informațională
- ✓ Limbajul UML



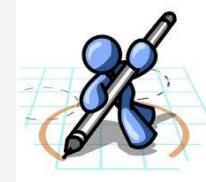
MODELAREA ÎN REALIZAREA SISTEMELOR INFORMATICE

- Modelarea - una dintre cele mai importante **tehnici de concepere** a sistemelor informaticе.
- **Model** – abstractizare a unei părți a realității înconjurătoare
- Trăsăturile caracteristice ale modelării:
 - *simplificarea*,
 - *subordonarea la un scop*,
 - *reprezentarea unei realități*,
 - *divizarea*,
 - *ierarhizarea*
 - *comunicarea*.



MODELAREA ÎN REALIZAREA SISTEMELOR INFORMATICE

- Analiza și proiectarea orientată-obiect folosește trei tipuri diferite de modele pentru descrierea unui sistem informatic:
 - ***modelul static*** care descrie obiectele și relațiile lor în sistem;
 - ***modelul dinamic*** ce descrie interacțiunile între obiecte în cadrul sistemului ;
 - ***modelul funcțional*** care descrie transformarea valorii datelor în sistem.



LIMBAJE PENTRU MODELAREA INFORMAȚIONALĂ



- La modul general, limbajele pentru modelarea informațională fac posibilă **descrierea concisă și exactă** a proprietăților sistemelor la diferite nivele de abstractizare.
- Sunt o parte **integrantă** a procesului de dezvoltare a sistemelor informaticice în cadrul căruia pot fi folosite pentru:
 - a face **legătura** între etapa de analiză a cerințelor și cea de implementare;
 - a **verifica proprietățile critice** ale unor sisteme;
 - a ajuta la **generarea** automată de cod și de cazuri de test.

CATEGORII DE LIMBAJE PENTRU MODELAREA INFORMAȚIONALĂ



În funcție de nivelul de formalizare pe care îl impun, limbajele pentru modelarea informațională se împart în:

- Limbaje **informale**
- Limbaje **semi-formale**
- Limbaje **formale**

Reguli de sintaxă	NU	DA	DA
Reguli de semantică	NU	NU	DA
Limbaj de specificare a cerințelor	Informal (limbajul natural)	Semi-formal (UML)	Formal (Z,OCL)

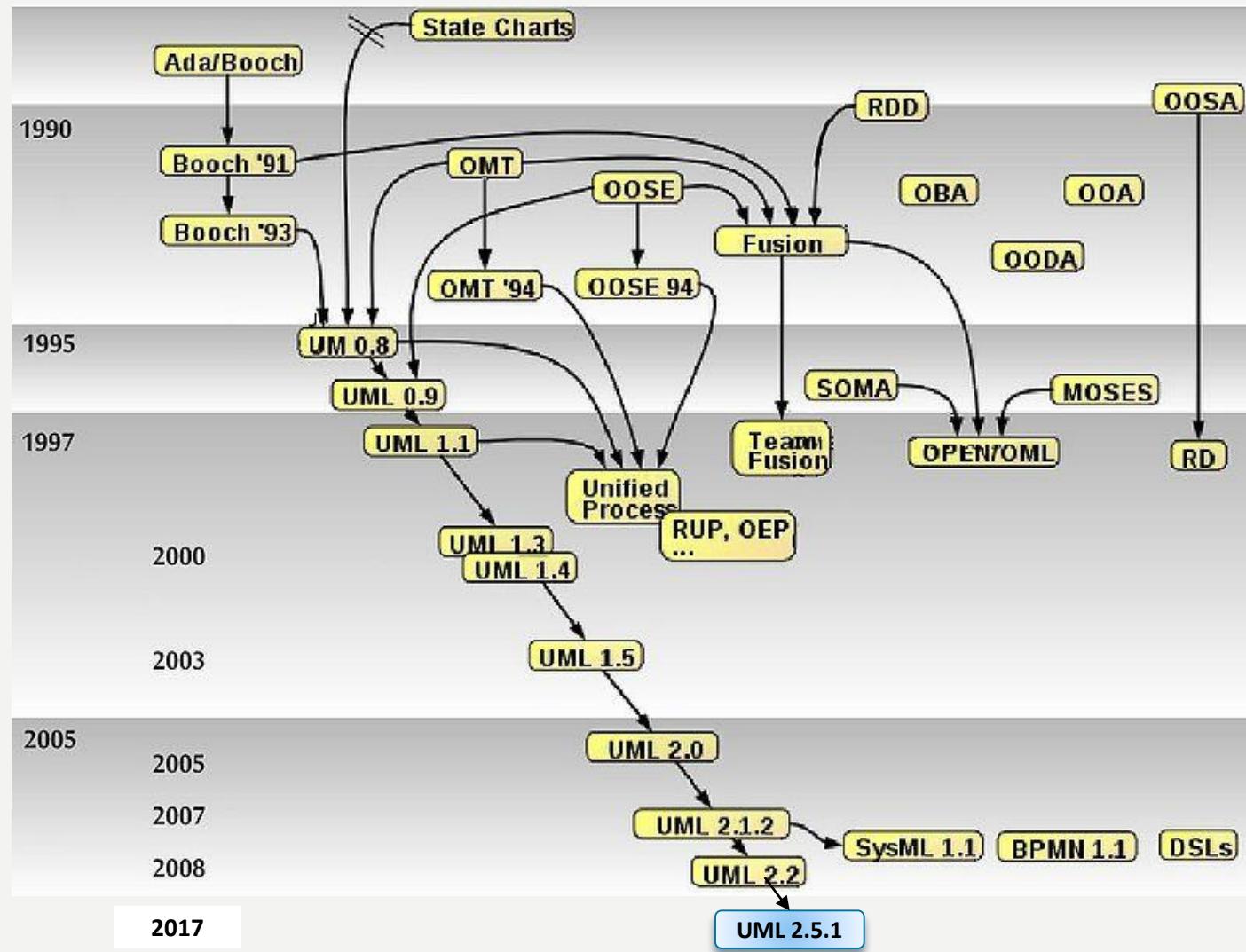
LIMBAJUL UML

- Limbajul standardizat *UML (Unified Modeling Language)* a apărut din necesitatea unei **standardizări** a tipologiei, semanticii și a reprezentării rezultatelor.
- În prezent, **UML** este un standard de modelare recunoscut de către **OMG (Object Management Group)**, standardizarea fiind realizată în noiembrie 1997, iar până în prezent realizându-se o perfecționare continuă a acestuia.
- UML poate fi definit ca fiind un **limbaj de vizualizare, specificare, construire și documentare a modelelor**. Valoarea lui constă în faptul că este un standard **deschis**, realizează **întreg ciclul de dezvoltare al software-ului**, acoperă multe tipuri de aplicații, este bazat pe marea experiență a celor care l-au realizat și poate fi implementat de multe produse de tip CASE.

LIMBAJUL UML

- UML are **notații standard** și semantică adekvată modelării sistemelor orientate obiect.
- Odată cu apariția acestui limbaj, proiectanții de sisteme pot **înțelege** mai ușor documentațiile de sistem.
- Înaintea acestei standardizări, un proiect orientat-obiect putea fi descris folosind **una din multele metode orientate-obiect disponibile**, iar dacă era necesară o revizuire, se pierdea un timp important cu analiza notațiilor și semanticii metodei folosite, înainte de a începe logica proiectării.

ISTORIA UML



ELEMENTELE DE BAZĂ ALE UML



1. Metamodel pentru modelarea orientată obiect

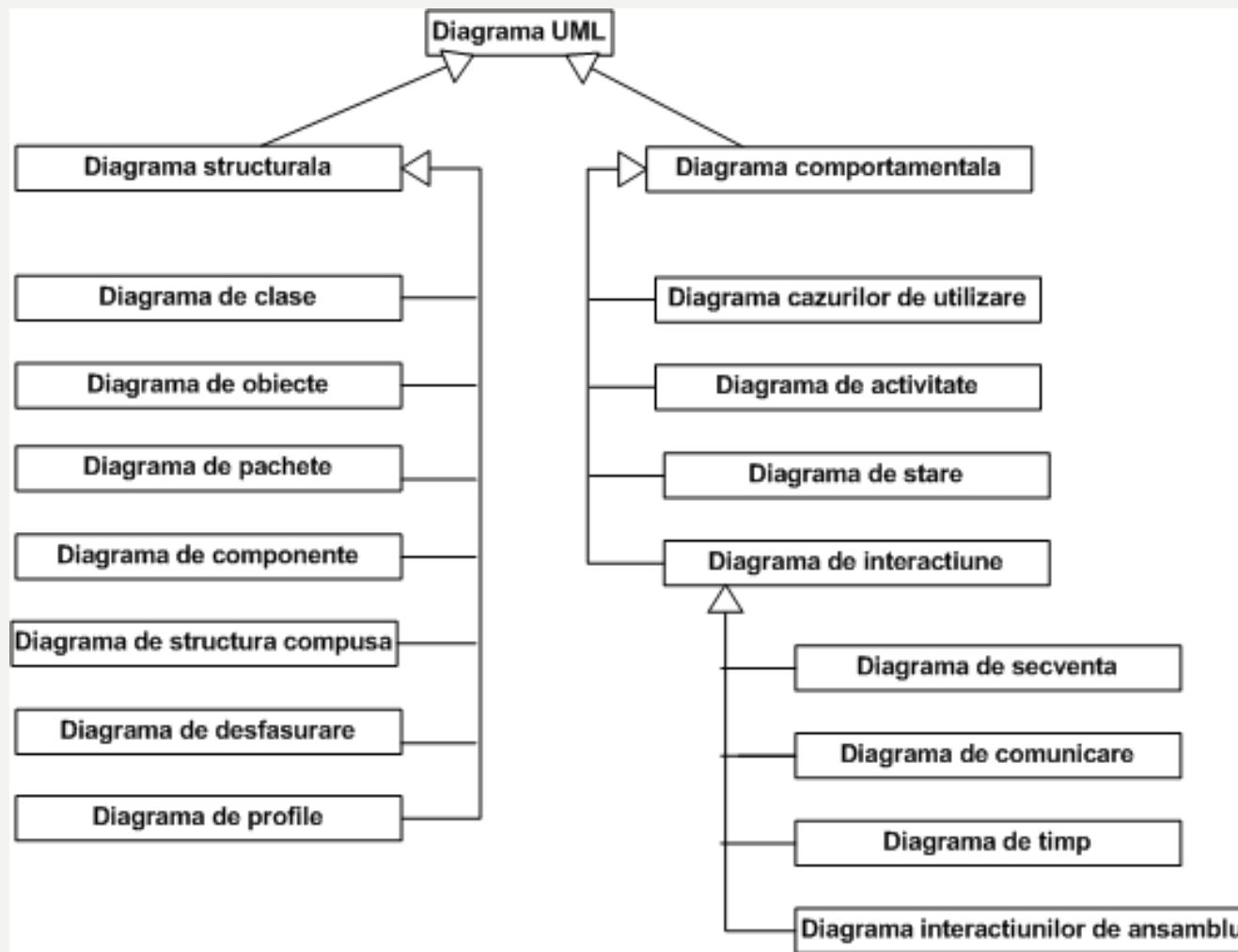
- Set coherent de definiții ale unor concepte și a relațiilor dintre ele;
- Se definește, folosind o sintaxă precisă, fiecare element utilizat în modelare (exemplu: definirea unei clase);
- Limbaj suport pentru transmiterea modelelor vizuale între diferite instrumente;
- Are o arhitectură pe patru niveluri.

Strat	Descriere	Exemplu
meta-metamodel	Definește limbajul pentru specificarea metamodelelor.	Concepte abstracte din care este derivat metamodelul.
metamodel	Definește limbajul pentru specificarea modelului.	Concepte: Clasă, Atribut, Operație, Componentă
model	Definește limbajul folosit pentru descrierea domeniului analizat.	Concepte: Student, Materie, Client, Produs, Comandă
obiectele utilizatorului	Definesc informații despre obiectele domeniului analizat.	Exemple: Student #3456, Materie #0512

ELEMENTELE DE BAZĂ ALE UML



2. Tipuri de diagrame



ELEMENTELE DE BAZĂ ALE UML



3. Mecanisme de extensie

- *Stereotipurile* caracterizează un element din model sau o relație între elemente (există stereotipuri predefinite).
- *Comentariile (notele)* descriu suplimentar un element din model.
- *Contrângerile* limitează utilizarea unui element din model.
- *Valori etichetate* reprezintă atrbute definite pentru un stereotip.
- *Profilele* personalizează metamodelul prin construcții care sunt specifice unui anumit domeniu, platformă sau metodă de dezvoltare.

MODELAREA CU AJUTORUL DIAGRAMELOR UML



- *Modelarea proceselor de afaceri* se realizează folosind **diagrama cazurilor de utilizare**. Această diagramă dirijează întreg procesul de dezvoltare a sistemului în cazul metodelor orientate pe cazuri de utilizare.
- *Modelarea structurii statice* se face cu ajutorul **diagramei claselor** (pentru modelarea structurii statice a claselor sistemului) și **diagramei obiectelor** (pentru modelarea structurii statice a obiectelor sistemului). **Diagrama pachetelor** este un mijloc de grupare a elementelor diagramelor în pachete.

MODELAREA CU AJUTORUL DIAGRAMELOR UML



- *Modelarea dinamicii* este realizată utilizând diagrame de interacțiune și diagrame care descriu comportamentul. UML utilizează două diagrame de interacțiune, una pentru modelarea circuitului mesajelor între obiecte, numită **diagrama de secvență** și alta, pentru modelarea interacțiunilor între obiecte, denumită **diagrama de comunicare**. Ca diagrame ce descriu comportamentul, se utilizează **diagrama de stare** pentru modelarea comportamentului obiectelor din sistem și respectiv **diagrama de activitate** pentru modelarea comportamentului cazurilor de utilizare, obiectelor sau a operațiilor.
- *Modelarea implementării* se face cu ajutorul a două diagrame. Modelarea componentelor se realizează utilizând **diagrama componentelor**, iar modelarea distribuirii sistemului se obține folosind **diagrama de desfășurare**.

CURSUL 3...

- Diagrama Cazurilor de Utilizare
- Instrumente CASE

ACADEMIA DE STUDII ECONOMICE BUCUREŞTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

PROIECTAREA SISTEMELOR INFORMATICE

- CURS 3 -

BUCUREŞTI
2020-2021

Diagrama de cazuri de utilizare

*Modelarea
funcțională*

Cuprins

- Introducere
- Cazuri de utilizare
- Actori
- Relații între cazuri de utilizare și actori
- Relații între cazuri de utilizare
- Relații între actori
- Descrierea textuală a unui caz de utilizare
- Bune practici
- Erori frecvente
- Rezumat notații



Introducere

- Cazul de utilizare reprezintă un concept fundamental al multor metodologii de dezvoltare orientate-obiect.

Diagramele de cazuri de utilizare:

- Au rolul de a reprezenta într-o formă grafică **funcționalitățile** pe care trebuie să le îndeplinească sistemul informatic în faza sa finală.
- Exprimă **așteptările** clienților/beneficiarilor:
 - esențial pentru o proiectare detaliată
- Sunt folosite în întreg procesul de analiză și proiectare.
- Modelul realizat de diagramele de cazuri de utilizare alături de documentele de descriere succintă a fiecărui caz de utilizare identifică poartă numele de **model al cerințelor**.



Introducere

- Diagramele de cazuri de utilizare sunt formate din **actori** și **cazuri de utilizare**, pe de o parte, și relațiile între acestea, pe de altă parte.
- Putem folosi diagrama de cazuri de utilizare pentru a răspunde la următoarele întrebări:
 - Ce este descris? (Sistemul)
 - Cine interacționează cu sistemul? (Actorii)
 - Ce pot face actorii? (Cazuri de utilizare)



Exemplu: Sistemul de administrare a studentilor

■ Sistem

(Ce este descris?)

- Sistemul de administrare a studentilor

■ Actori

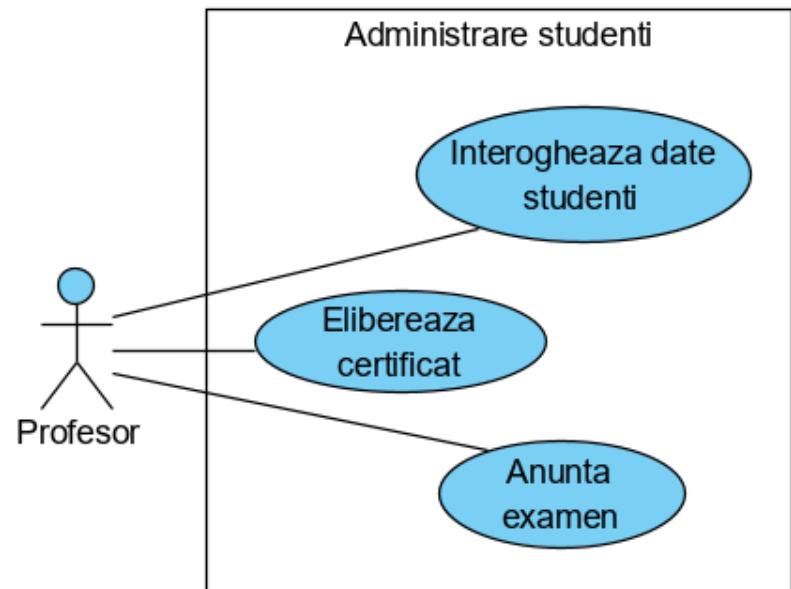
(Cine interacționează cu sistemul?)

- Profesorul

■ Cazuri de utilizare

(Ce pot face actorii?)

- Interoghează date student
- Emite certificat
- Anunță examen



Caz de utilizare

- “Specifică un set de acțiuni executate de către un sistem sau un subiect și care conduc la un anumit rezultat. Rezultatul, în mod normal, este important pentru un actor sau un beneficiar. “ (OMG)
- Oferă beneficii tangibile pentru unul sau mai mulți actori care comunică cu acest caz de utilizare.
- Derivă din cerințele identificate de la clienți.
- Multimea tuturor cazurilor de utilizare descrie funcționalitățile pe care sistemul trebuie să le ofere.
 - Documentează funcționalitățile pe care le oferă acel sistem.
- Notații alternative:

Anunta examen

Anunta examen

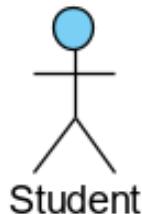
Anunta examen



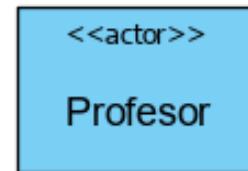


Actor (1/3)

- Actorii interacționează cu sistemul:
 - **prin folosirea** cazurilor de utilizare, spre exemplu, actorii inițiază execuția unui caz de utilizare
 - **find folosiți** de către cazurile de utilizare, spre exemplu, actorii oferă funcționalități pentru execuția unui caz de utilizare
- Actorii reprezintă roluri pe care le adoptă utilizatorii.
 - Anumiți utilizatori pot juca mai multe roluri în același timp.
- Actorii nu reprezintă o parte din sistem, ei se găsesc **în afara granițelor** sistemului.
- Notații alternative:



Student



Profesor

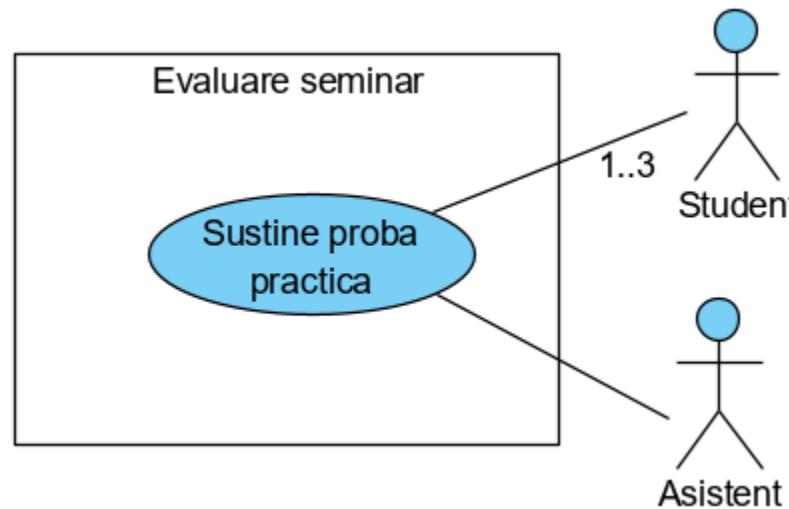


E-mail Server



Actor (2/3)

- De obicei datele utilizatorului sunt administrate în cadrul sistemului. Aceste date sunt modelate sub forma obiectelor și claselor. Exemplu: actor **Asistent**
 - Actorul **Asistent** interacționează cu sistemul **Activitate Seminar** prin utilizarea acestuia.
 - Clasa **Asistent** descrie obiectele care reprezintă datele utilizatorului (spre exemplu, nume, poziție etc.).

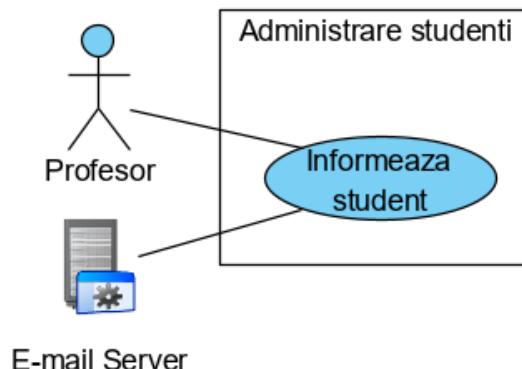


Actor (3/3)

- **Umani**
 - Exemple: **Student, Profesor**
- **Non-umani**
 - Exemple: **Server E-Mail , . . .**
- **Primar**: este principalul beneficiar al execuției unui caz de utilizare
- **Secundar**: nu primește niciun beneficiu direct
- **Activ**: inițiază execuția unui caz de utilizare
- **Pasiv**: oferă funcționalitate pentru execuția unui caz de utilizare
- **Exemplu**:

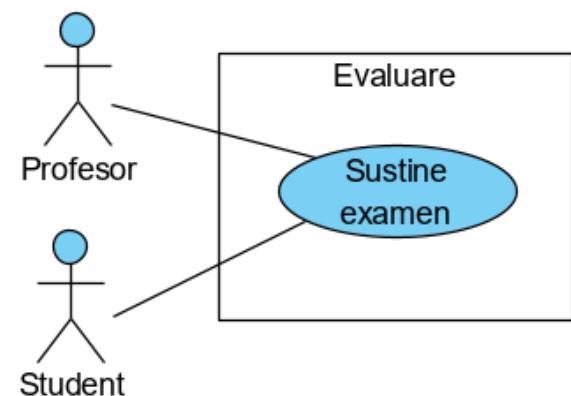
*Uman
Primar
Activ*

*Non-uman
Secundar
Pasiv*



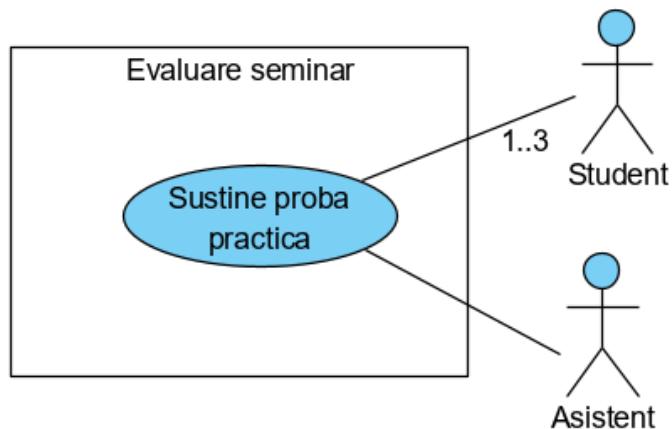
*Uman
Secundar
Activ*

*Uman
Primar
Activ*



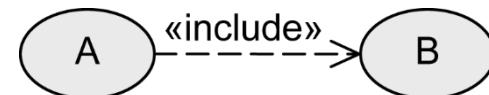
Relații între cazuri de utilizare și actori

- Asocierile simple sunt folosite pentru a **conecta** un actor cu un caz de utilizare. Aceasta reprezintă o **cale de comunicare** între cei doi.
- Fiecare actor trebuie să comunice cu cel puțin un caz de utilizare.
- Asocierea este întotdeauna binară.
- Pot fi specificate **multiplicități**. Multiplicitatea mai mare decât unu la capătul:
 - corespunzător CU \Rightarrow actorul este implicat în mai multe cazuri de utilizare de acel tip și poate iniția cazuri de utilizare: în paralel (concurrent), la diferite momente de timp sau mutual exclusiv în timp.
 - corespunzător actorului \Rightarrow mai multe instanțe ale actorului sunt implicate în inițierea cazului de utilizare putând realiza acțiuni simultane sau succesive.

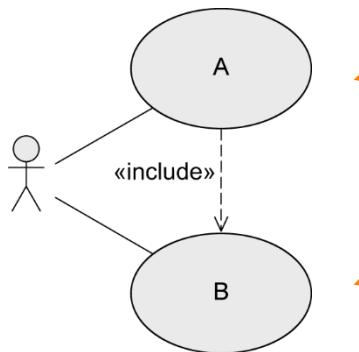


Relații între cazuri de utilizare

Relația de includere



- Comportamentul unui unui caz de utilizare (cazul de utilizare inclus) este integrat în comportamentul altui caz de utilizare (cazul de utilizare de bază).
- Cazul de utilizare care îl include pe un altul nu este complet.

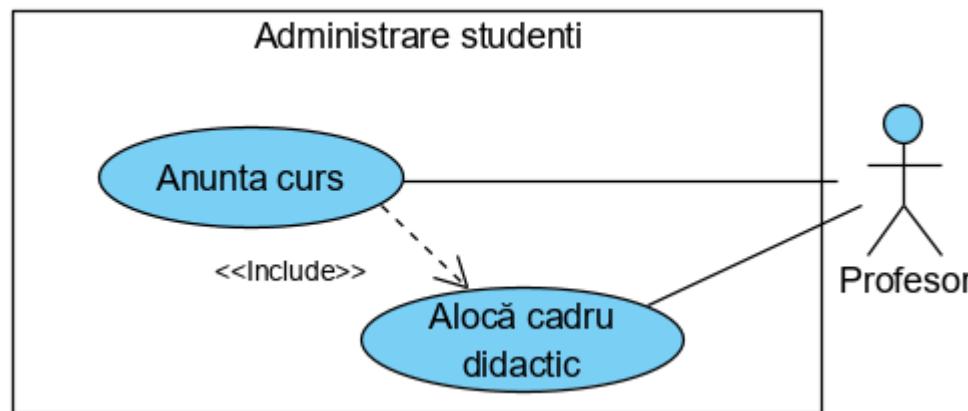


Caz de utilizare de bază

cere comportamentului cazului de utilizare inclus să îi permită să includă funcționalitatea sa

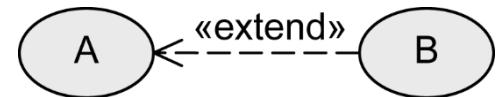
Cazul de utilizare inclus

este independent din punct de vedere al execuției

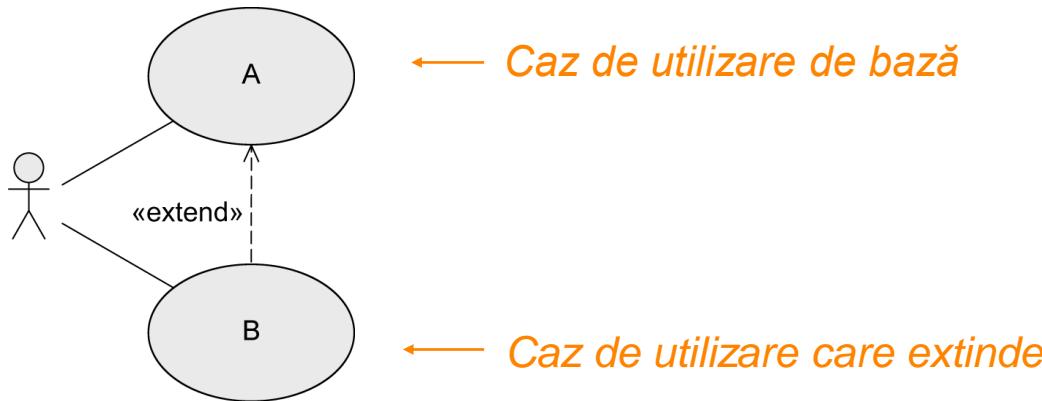


Relații între cazuri de utilizare

Relația de extindere



- Comportamentul unui caz de utilizare (cel care extinde) poate fi integrat în comportamentul altui caz de utilizare (cazul de utilizare de bază), dar acest lucru nu este obligatoriu.
- Ambele cazuri de utilizare pot fi executate în mod independent.

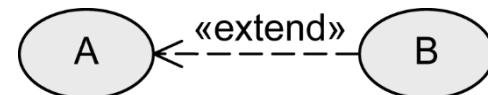


- A decide dacă B este executat.
- Punctele de extindere definesc punctul în care comportamentul este integrat.
- Condițiile definesc care sunt circumstanțele în care comportamentul este integrat.

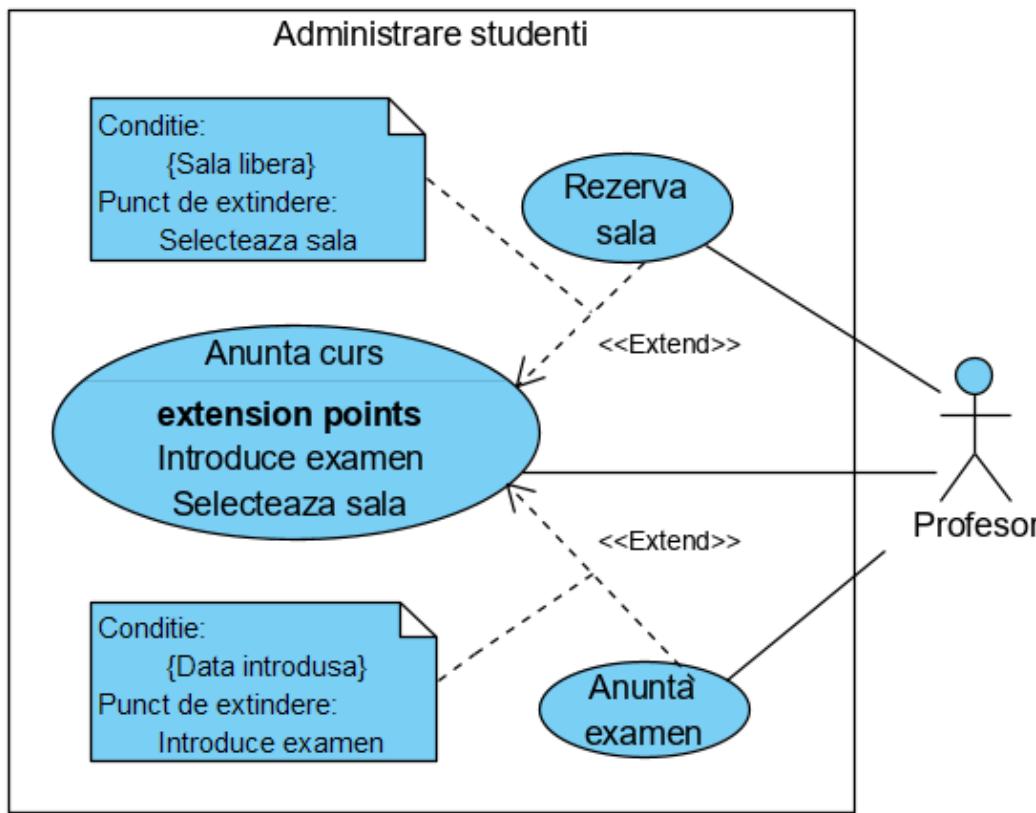


Relații între cazuri de utilizare

Relația de extindere: puncte de extindere



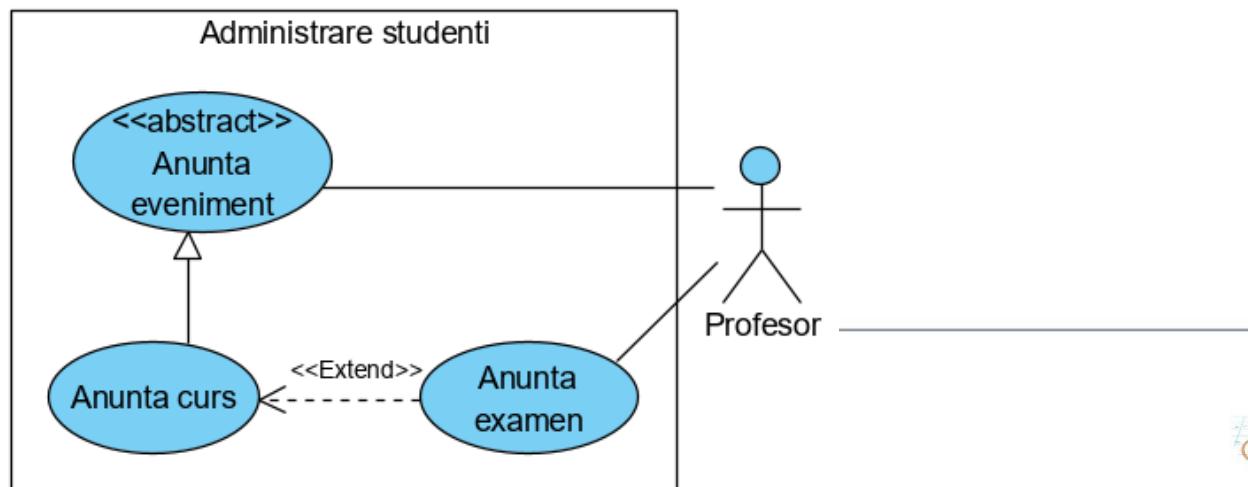
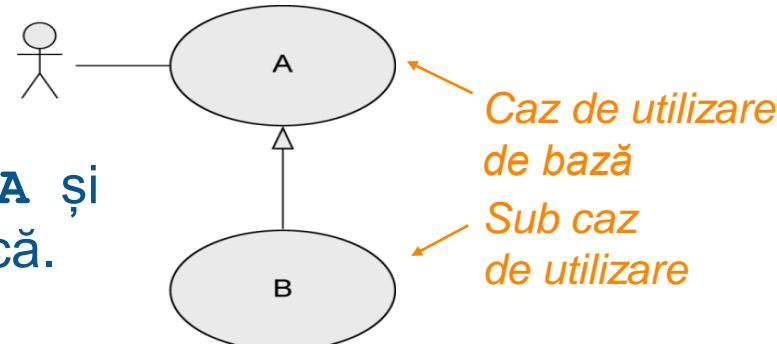
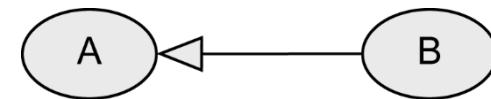
- Punctele de extindere sunt scrise direct în cazul de utilizare.
- Este permisă specificarea mai multor puncte de extindere.



Relații între cazuri de utilizare

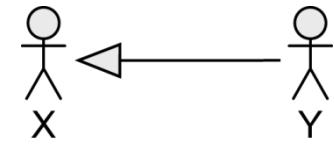
Relația de generalizare

- Cazul de utilizare **A** generalizează cazul de utilizare **B**.
- **B** moștenește comportamentul lui **A** și îl poate extinde sau suprascrie.
- **B** moștenește și toate relațiile lui **A**.
- **B** adoptă funcționalitățile de bază ale lui **A** și decide ce părți din **A** execută sau modifică.
- **A** poate fi etichetat ca **{abstract}**
 - Nu poate fi executat în mod direct
 - Numai **B** este executabil

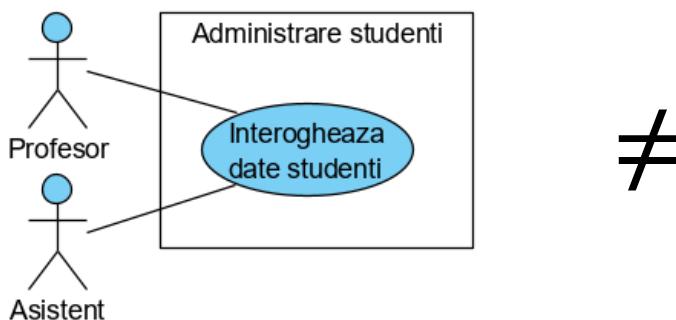
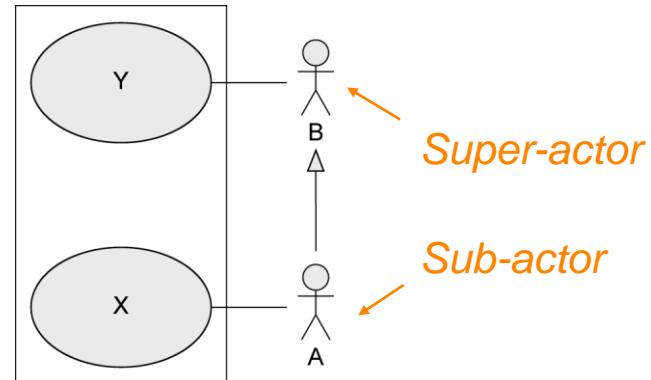


Relații între actori

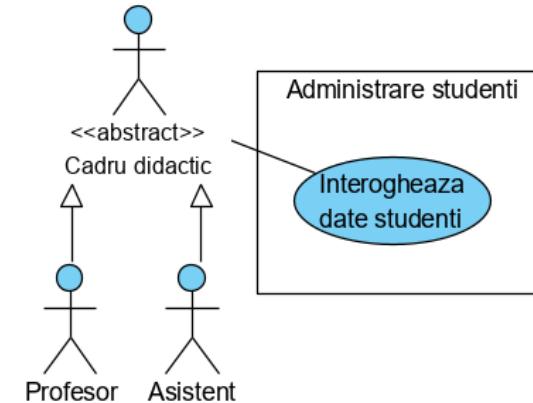
Relația de generalizare între actori



- Actorul **A** moștenește de la actorul **B**.
- **A** poate comunica cu **X** și **Y**.
- **B** poate comunica doar cu **Y**.
- Este permisă *moștenirea multiplă*.
- Pot exista actori *abstracți*.



Profesorul și Asistentul sunt necesari pentru execuția Interoghează date student



Profesorul SAU Asistentul sunt necesari pentru execuția Interoghează date student



Descrierea sub formă de şablon a unui caz de utilizare

Element al cazului de utilizare	Descriere
Cod	Un identificator unic asociat cazului de utilizare
Stare	Stadiul de finalizare în care se găsește, de exemplu: schiță, finalizat sau aprobat
Scop	Sistemul (parte a sistemului) sau aplicația căreia îi aparține
Nume	Numele cazului de utilizare, cât mai scurt și reprezentativ
Actor principal	Beneficiarul care inițiază cazul de utilizare și care urmărește un anumit scop
Descriere	Prezentare scurtă, în text liber, a cazului de utilizare
Precondiții	Ce condiții trebuie satisfăcute pentru ca scenariul să poată începe
Postcondiții	Ce condiții trebuie îndeplinite pentru a garanta un final reușit al scenariului
Posibile erori	Erori relevante pentru domeniul problemei
Starea sistemului în caz de eroare	În ce stare se găsește sistemul după apariția erorii
Declanșator	Un eveniment sau o succesiune de evenimente care inițiază cazul de utilizare
Flux de bază	Descrie însiruirea evenimentelor atunci când totul se petrece conform unui scenariu prestabilit; nu există excepții sau erori
Fluxuri alternative	Cele mai semnificative alternative și excepții ale scenariului de bază
Relații	Ce relații are cu alte cazuri de utilizare (de tipul include sau extend)



Descrierea unui caz de utilizare - Exemplu

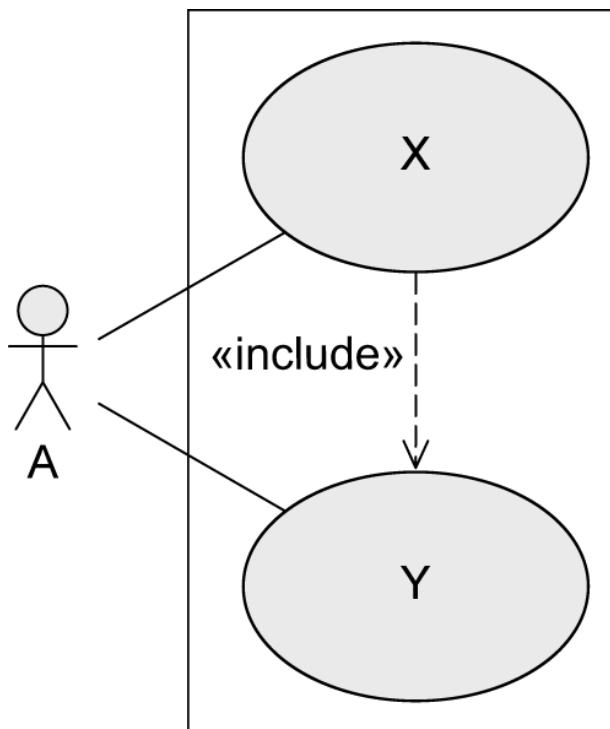
Element al cazului de utilizare	Descriere
Cod	CU01
Stare	Schiță
Scop	Gestiunea sălilor pentru desfășurarea activităților universitare
Nume	Rezerva sala
Actor principal	Angajat
Descriere	Angajatul rezervă o sală a universității pentru un eveniment
Precondiții	Angajatul este autorizat să rezerve săli
Postcondiții	Sala este rezervată
Declanșator	Angajatul solicită rezervarea unei săli
Posibile erori	Nu există sală liberă.
Starea sistemului în caz de eroare	Angajatul nu a rezervat sala.
Flux de bază	<ol style="list-style-type: none">1. Angajatul se autentifică în sistem.2. Angajatul selectează sală.3. Angajatul selectează data și ora.4. Sistemul confirmă că sala este liberă. [Curs alternativ A: Sala nu este liberă]5. Angajatul confirmă rezervarea.
Fluxuri alternative	A: 1. Sistemul propune săli alternative. 2. Angajatul selectează o sală alternativă și confirmă rezervarea.
Relații	Extinde cazul de utilizare CU06 Anunta curs.



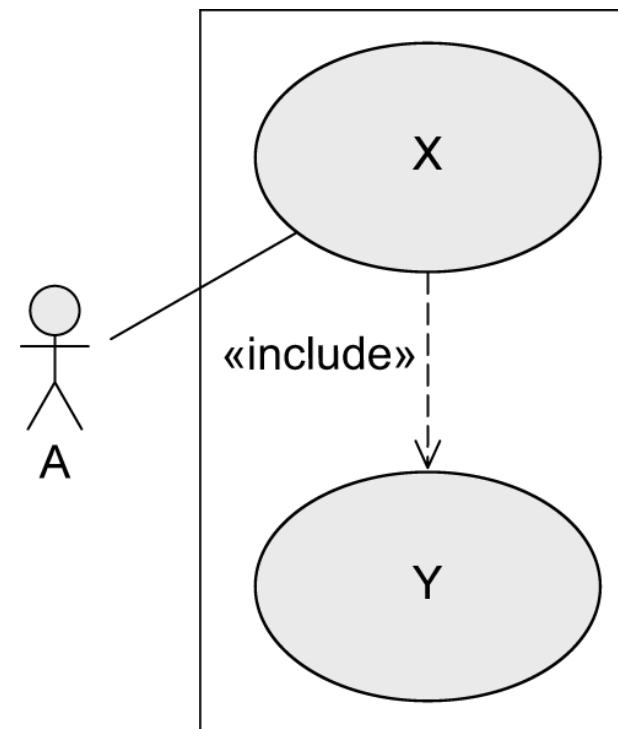
Bune practici

«include»

Standard UML



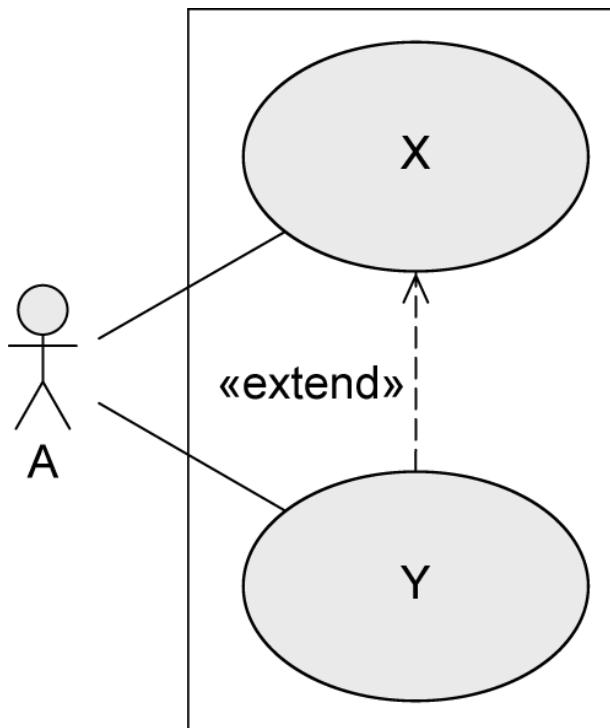
Bune practici



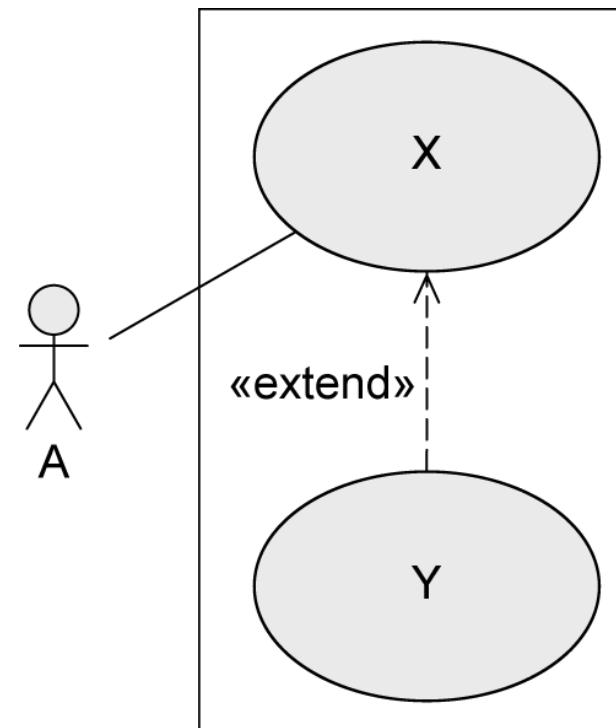
Bune practici

«extend»

Standard UML



Bune practici



Bune practici

Identificarea actorilor

- Cine folosește principalele cazuri de utilizare?
- Cine are nevoie de suport pentru munca lor zilnică?
- Cine este responsabil pentru administrarea sistemului?
- Care sunt dispozitivele externe/sistemele (software) cu care sistemul trebuie să comunice?
- Cine este interesat de rezultatele oferite de sistem?

Identificarea cazurilor de utilizare

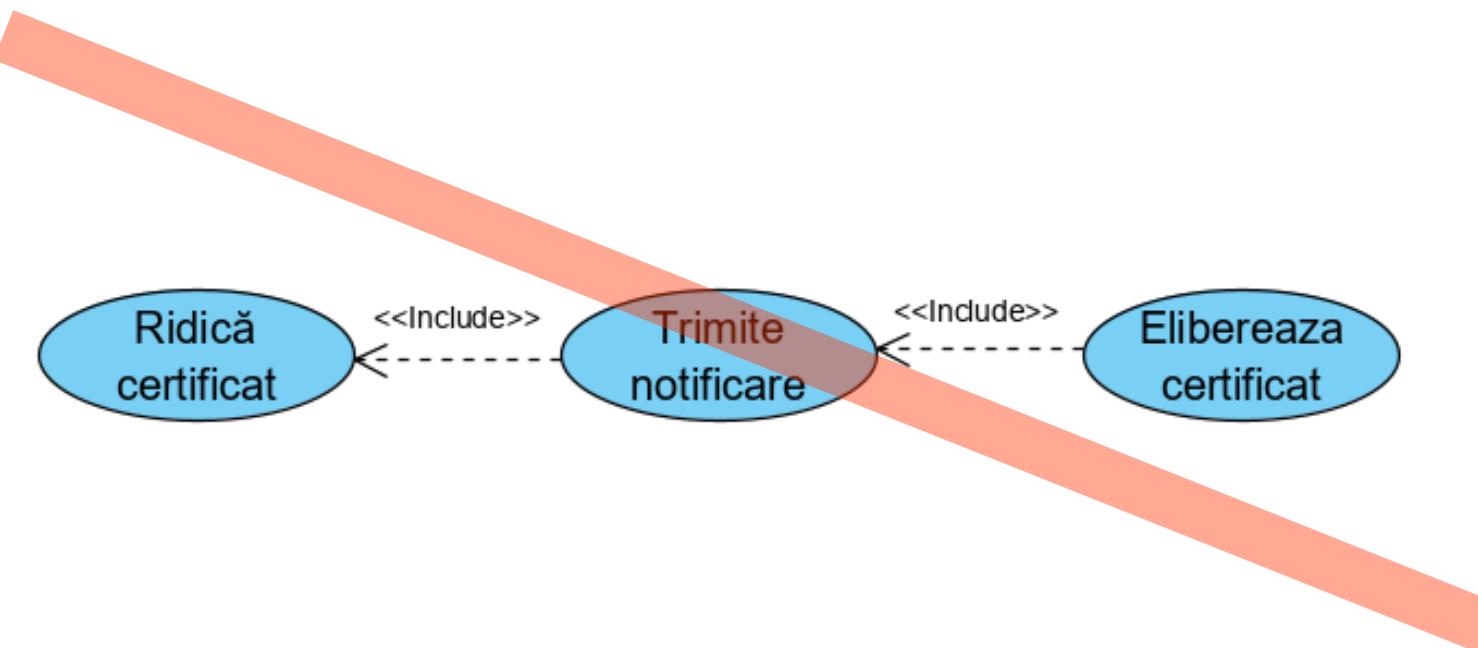
- Care sunt cele mai importante activități pe care trebuie să le realizeze un actor?
 - Dorește un actor să interogheze sau să modifice informațiile conținute în sistem?
 - Informează un actor sistemul despre schimbările din alte sisteme?
 - Ar trebui ca un actor să fie informat despre rezultatele neașteptate apărute în sistem?
-



Bune practici

Erori de evitat - 1

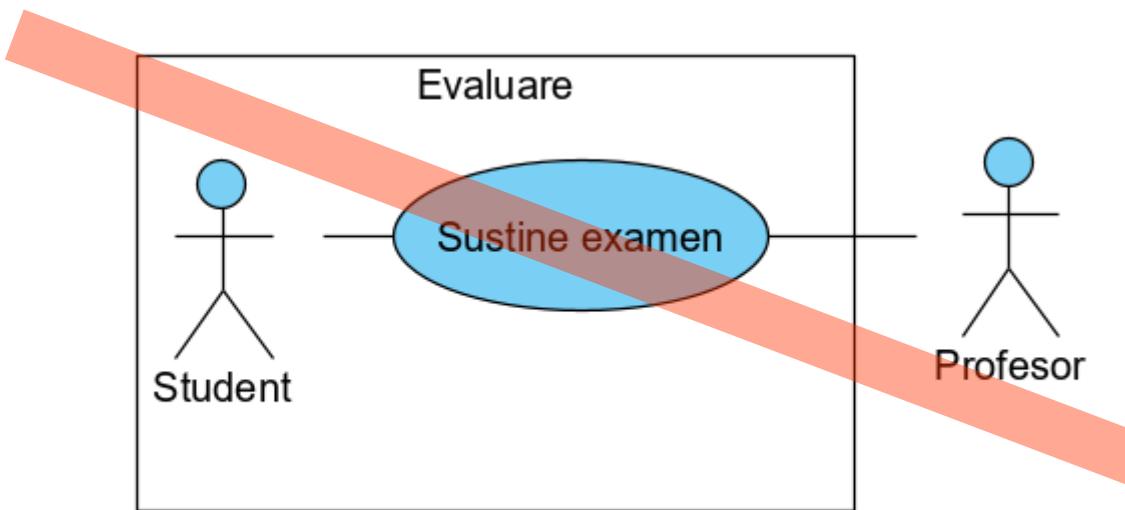
- Cazurile de utilizare nu modelează procese sau fluxuri de lucru!



Bune practici

Erori de evitat - 2

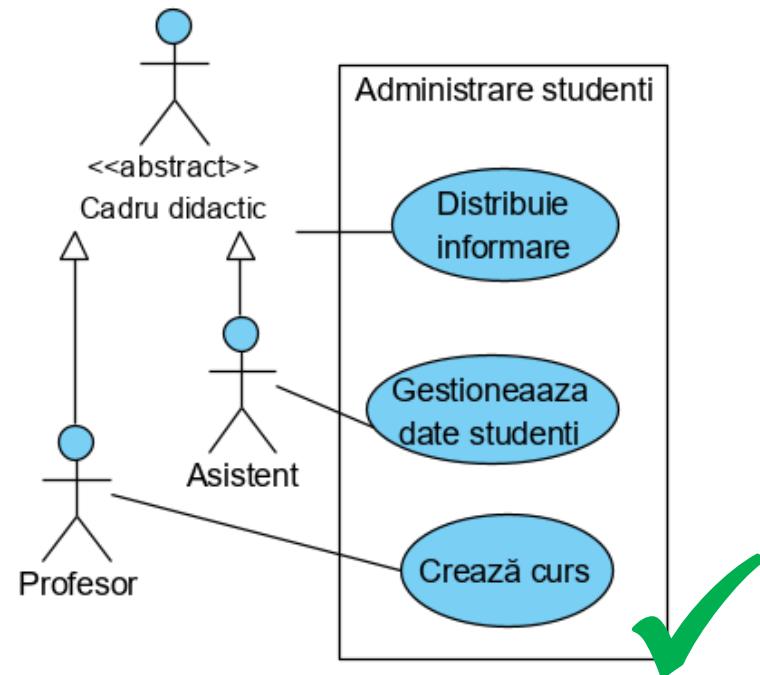
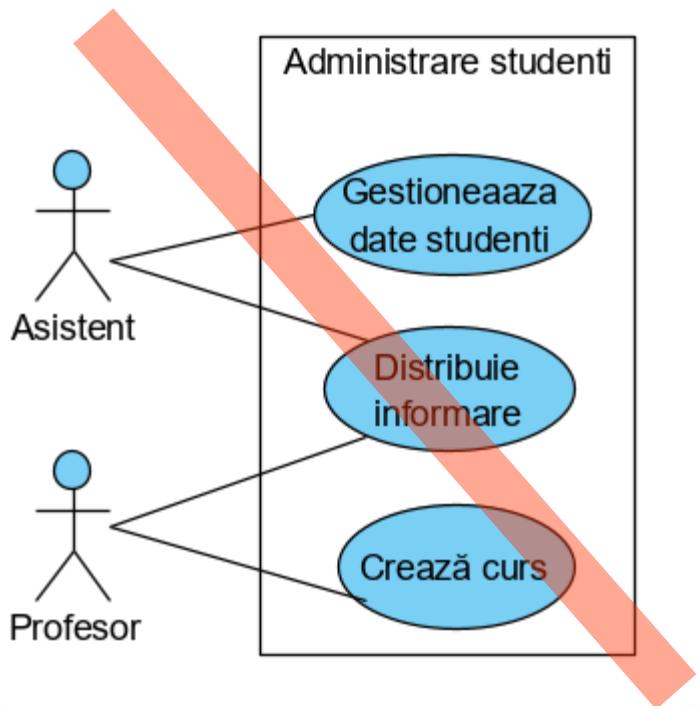
- Actorii nu sunt parte a sistemului, ce aceea trebuie poziționați în afara granițelor!



Bune practici

Erori de evitat - 3

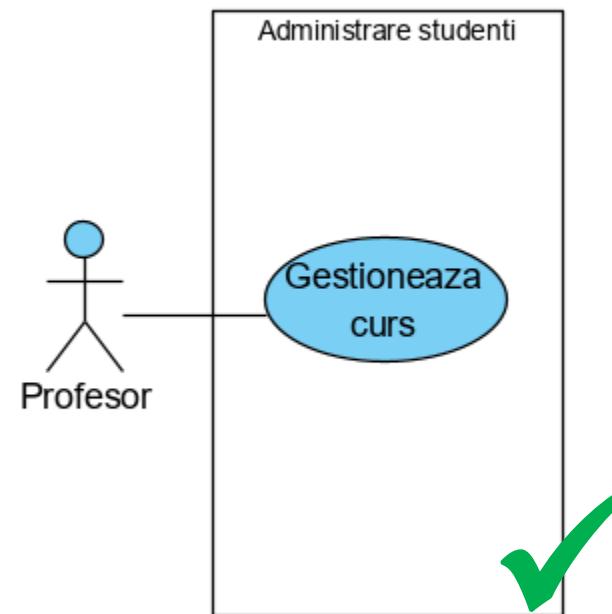
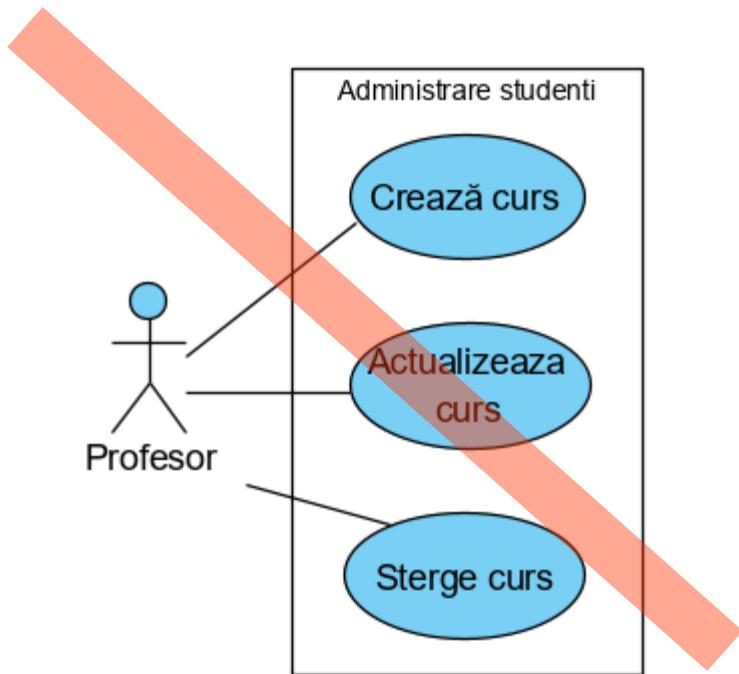
- Cazul de utilizare **Distribuie informare** necesită DOAR un actor **Asistent SAU** un actor **Profesor** pentru a se executa



Bune practici

Erori de evitat - 4

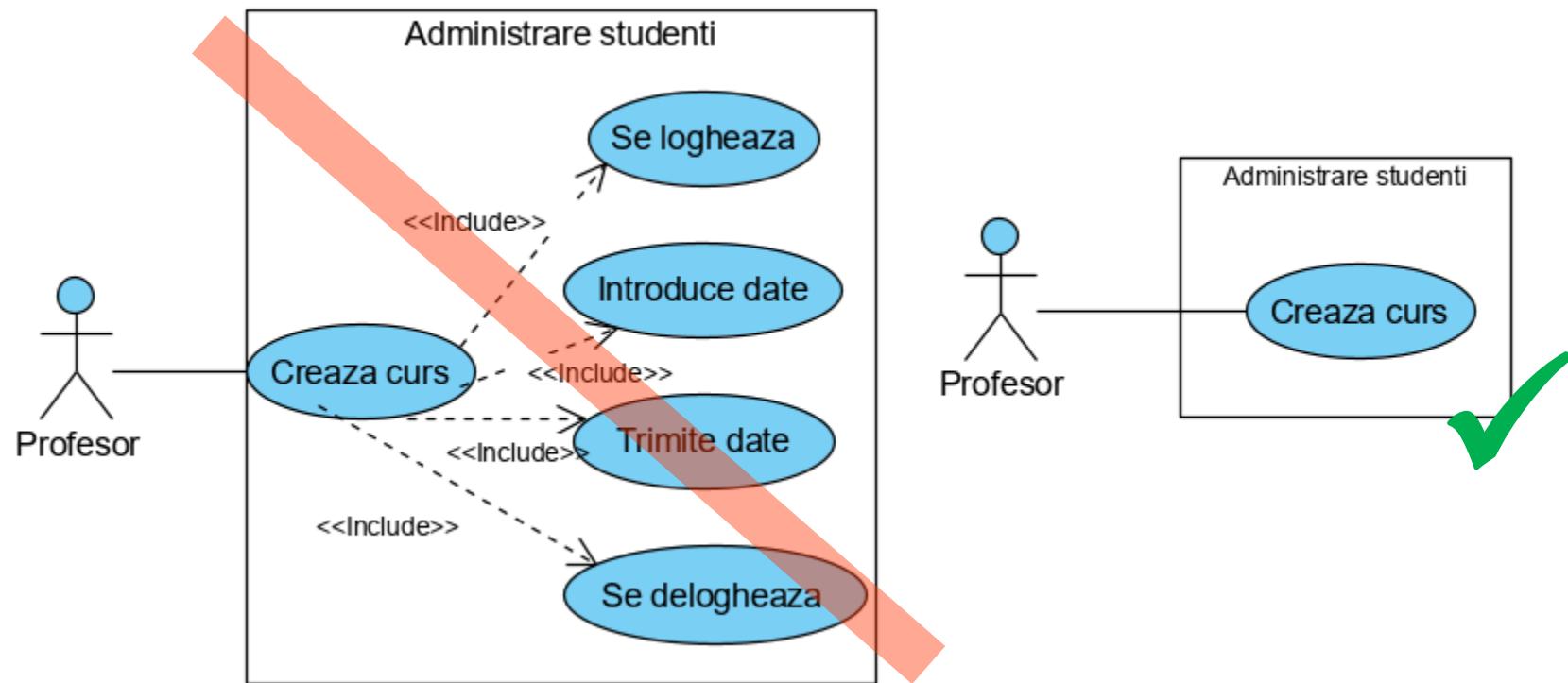
- Mai multe cazuri de utilizare mici care au același obiectiv pot fi grupate într-un singur caz de utilizare.



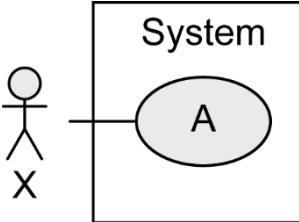
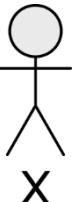
Bune practici

Erori de evitat - 5

- Pașii de executat sunt părți ale cazului de utilizare, cazurile de utilizare NU trebuie descompuse funcțional

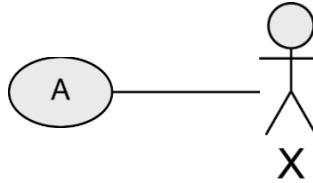
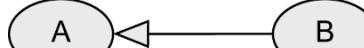
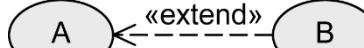
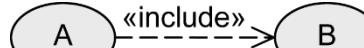


Notății - 1

Nume	Notație	Descriere
Sistem		Granițele dintre sistem și utilizatorii acestuia
Caz de utilizare		Unitate funcțională a sistemului
Actor		Rol al utilizatorilor în sistem



Notății - 2

Nume	Notăție	Descriere
Asociere		Relație între cazuri de utilizare și actori
Generalizare		Relație de moștenire între actori sau cazuri de utilizare
Relația de extindere		B extinde A: utilizare optională a CU B de către CU A
Relația de includere		A include B: necesită folosirea CU B de către CU A





Recapitulare

1. Ce descrie diagrama cazurilor de utilizare?
 2. Ce relații există între actori și cazuri de utilizare?
 3. Poate un actor să nu comunica cu niciun caz de utilizare? Dar cu mai multe cazuri de utilizare?
 4. Ce semnifică într-o relație de asociere o multiplicitate mai mare decât unu la capătul actorului?
 5. Ce reprezintă un actor secundar? Dar un actor activ?
 6. Pentru relația de includere, cazul de utilizare care îl include pe altul este complet? Argumentați.
 7. Explicați relația de extindere dintre cazurile de utilizare.
 8. Ce este un actor abstract și cum poate fi folosit? Exemplificați.
 9. Cum reprezentăm situația în care mai mulți actori sunt necesari pentru realizarea unui caz de utilizare?
-



Instrumente CASE

Cuprins

- Instrumente CASE: concepte, obiective și facilități
- Clasificarea instrumentelor CASE
- Arhitectura mediului CASE
- Visual Paradigm





Instrumente CASE

- ❑ CASE : Computer Aided Software Engineering
- ❑ Instrumentele CASE - instrumentele integrate care susțin activitățile front-end și back-end legate de dezvoltarea sistemelor informatiche
- ❑ Utilitate de ordin:
 - ❑ cantitativ
 - ❑ calitativ
- ❑ Instrumentele CASE reduc substanțial sau elimină multe din problemele de proiectare și dezvoltare a aplicațiilor informatiche.





Obiective

□ Obiectivul principal al tehnologiei CASE:

de a îmbunătăți productivitatea și calitatea sistemelor rezultate, asistând echipa de dezvoltare pe parcursul diferitelor etape ale procesului de realizare, de la identificarea cerințelor funcționale și nefuncționale ale sistemului până la proiectarea și implementarea acestuia, luând în considerare toate caracteristicile tehnice și operaționale relevante.





Obiective

- specificarea corectă și completă a cerințelor sistemului;
- reducerea timpului și costului de proiectare și dezvoltare;
- integrarea activităților de proiectare și dezvoltare prin utilizarea unor metodologii/metode comune;
- standardizarea procesului de proiectare și dezvoltare a sistemelor informaticе;
- simplificarea și îmbunătățirea procesului de testare;





Obiective

- simplificarea etapei de întreținere a sistemelor informaticice;
- realizarea unor documentații de calitate;
- îmbunătățirea managementului proiectelor;
- reutilizarea modulelor aplicațiilor și a documentației;
- îmbunătățirea portabilității aplicațiilor;
- asigurarea acurateței componentelor construite





Facilități

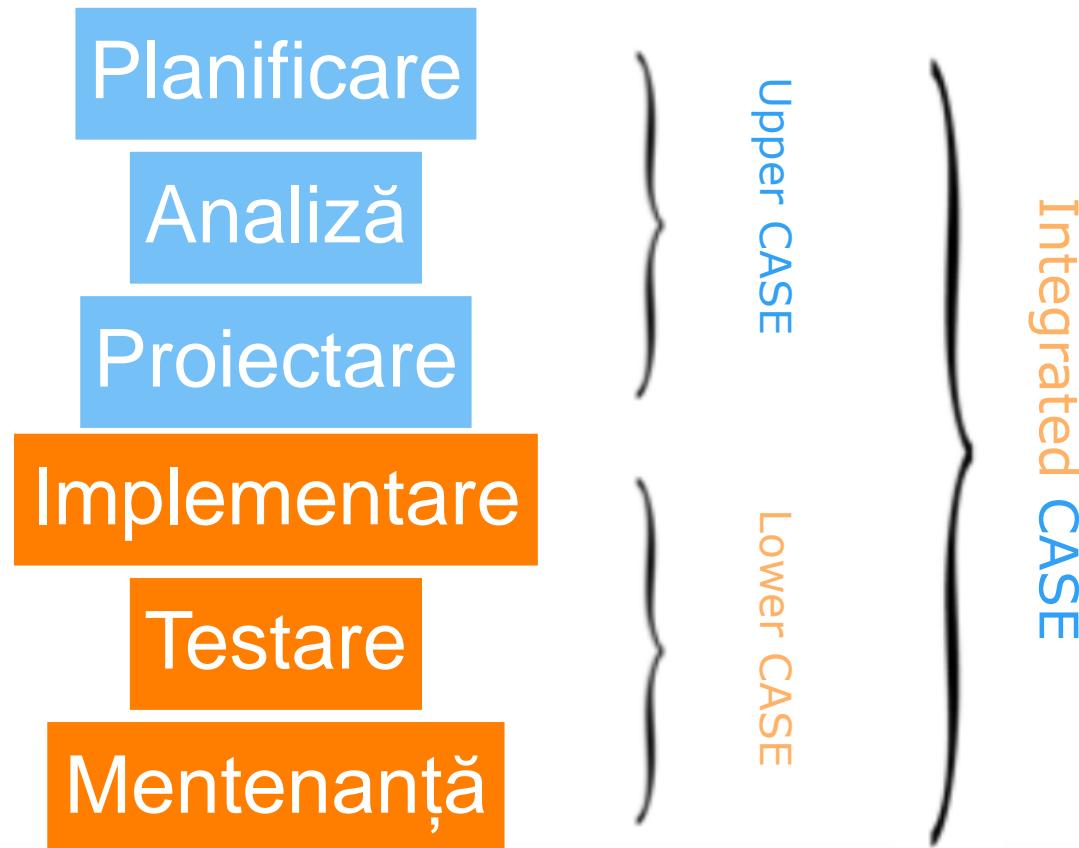
- ❑ suport pentru conducerea proiectului. Deosebit de utile sunt instrumentele de planificare și estimare a timpului și resurselor necesare realizării sistemului, precum și gestiunea versiunilor proiectului;
- ❑ generarea documentației de realizare a sistemului informatic;
- ❑ stocarea și regăsirea datelor din depozitul central de date (repository) prin utilitare specifice;
- ❑ verificarea automată a consistenței și completitudinii datelor printr-un analizor ce conține reguli specifice pentru fiecare metodologie/metodă;
- ❑ generarea automată a codului de program, pornind de la specificațiile de proiectare;
- ❑ tehnica de inginerie inversă (reverse engineering) prin care se permite revenirea dintr-o etapă de realizare a aplicației la o etapă precedentă pentru eventuale modificări
- ❑ suport pentru realizarea de prototipuri, prin limbaje de programare de nivel înalt și generatoare.



Tipologia instrumentelor CASE



1. După aria de cuprindere a ciclului de realizare a aplicației:



Tipologia instrumentelor CASE



- *Instrumente CASE front-end* (sau upper CASE) care oferă suport pentru primele etape de realizare a aplicațiilor informatiche (analiza și specificarea cerințelor, proiectarea logică).
 - *Instrumente CASE back-end* (sau lower CASE) care oferă suport pentru ultimele etape de realizare a aplicațiilor informatiche (proiectarea fizică, elaborarea programelor, testarea, întreținerea sistemului).
 - *Instrumente CASE cross life cycle* care oferă suport pentru activitățile ce apar în mai multe etape ale procesului de proiectare și dezvoltare a sistemelor informatiche (de exemplu, instrumente utilizate pentru managementul proiectului, generatoare de documentație).
-





2. După **scopul** utilizării:

- *Instrumente CASE pentru construirea diagramelor* – componentele sistemului, fluxurile de date și fluxurile de control între diferitele componente ale acestuia pot fi reprezentate în formă grafică/
- *Instrumente CASE pentru managementul proiectului* - utilizate pentru planificarea proiectelor, estimarea costurilor și eforturilor și planificarea resurselor. Instrumentele de management al proiectelor ajută la stocarea și partajarea informațiilor despre proiect în timp real în întreaga organizație.



Tipologia instrumentelor CASE



- *Instrumentele de documentare* generează documentație pentru utilizatorii tehnici și utilizatorii finali. Utilizatorii tehnici sunt specialisti, membri ai echipei de dezvoltare care utilizează manualule de sistem, manualule de instruire, manuale de instalare etc. Documentele utilizatorului final descriu funcționarea și instrucțiunile sistemului, cum ar fi manualul utilizatorului;
- *Instrumente CASE pentru analiză* - aceste instrumente ajută la colectarea cerințelor, verifică automat dacă există inconsecvențe, inexacități în diagrame, redundanțe de date sau omisiuni eronate.
- *Instrumente CASE pentru proiectare* - aceste instrumente îi ajută pe proiectanții de software să proiecteze structura produsului, care poate fi descompusă în module mai mici folosind tehnici de rafinare. Aceste instrumente oferă detalii despre fiecare modul și interconectările dintre module



Tipologia instrumentelor CASE



- *Instrumente pentru managementul schimbărilor* - automatizează urmărirea modificărilor, gestionarea fișierelor, gestionarea codului etc
 - *Instrumente de programare* - aceste instrumente constau din medii de programare precum IDE (Integrated Development Environment), biblioteci și instrumente de simulare. Ele oferă ajutor în construirea produselor software și includ facilități pentru simulare și testare.
 - *Instrumente de asigurare a calității* - asigurarea calității într-o organizație software monitorizează procesul de inginerie și metodele adoptate pentru a dezvolta produsul software pentru a asigura îndeplinirea standardelor organizației. Instrumentele QA constau în instrumente de configurare și control al modificărilor și instrumente de testare software.
-





Tipologia instrumentelor CASE

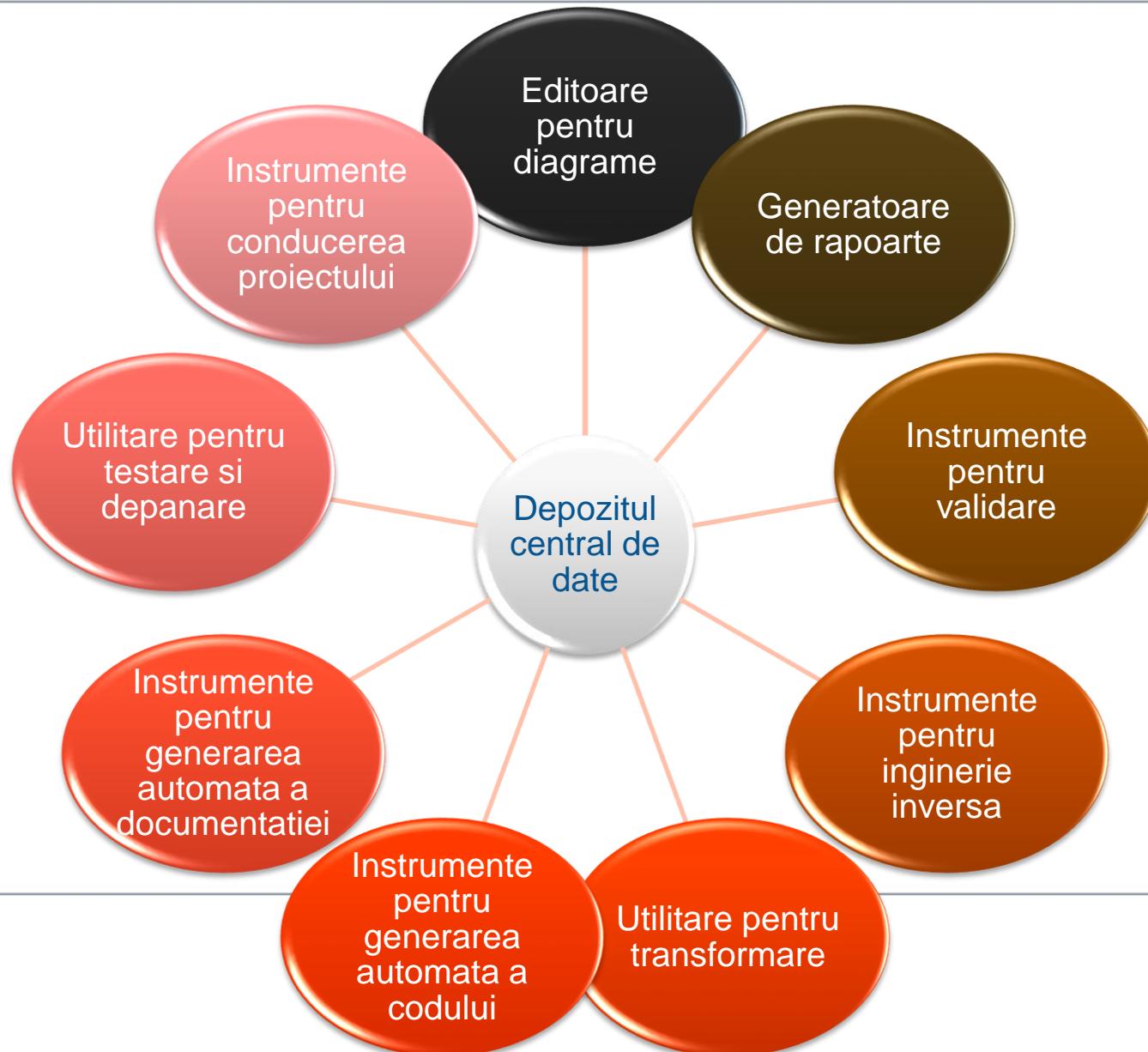
3. După dimensiunea suportului oferit:

- Instrumente CASE propriu-zise* ce oferă suport pentru o singură activitate din cadrul unei etape de realizare a aplicațiilor (de exemplu, editoare de diagrame și text, instrumente pentru analiza consistenței și completitudinii specificațiilor de sistem, depanatoare etc.);
- Bancurile de lucru CASE* (workbenches) care oferă suport pentru o etapă din ciclul de realizare a aplicației;
- Mediile CASE* care oferă suport pentru cea mai mare parte (sau toate) dintre etapele de realizare a sistemului informatic. Din această categorie fac parte următoarele instrumente: IBM Rational Architect Cradle/3SL, Corporate Modeler/CASEWise Inc etc.





Arhitectura mediului CASE





Arhitectura mediului CASE

- **Depozitul de date central (nucleul unui mediu I-CASE)** stochează toate obiectele și informațiile necesare pentru proiectarea, modelarea și generarea aplicațiilor. Depozitul de date central conține depozitul de informații (information repository) și dicționarul de date. Depozitul de informații conține informațiile despre afacerile organizației și portofoliul ei de aplicații. Dicționarul de date gestionează și controlează accesul la depozitul de informații. El stochează descrieri ale datelor și ale resurselor de prelucrare a datelor.
- **Editoarele pentru diagrame** permit reprezentarea vizuală a unui sistem și a componentelor lui. Diagramele sunt foarte eficace pentru reprezentarea fluxurilor de procese, a structurilor de date și a structurilor de program.



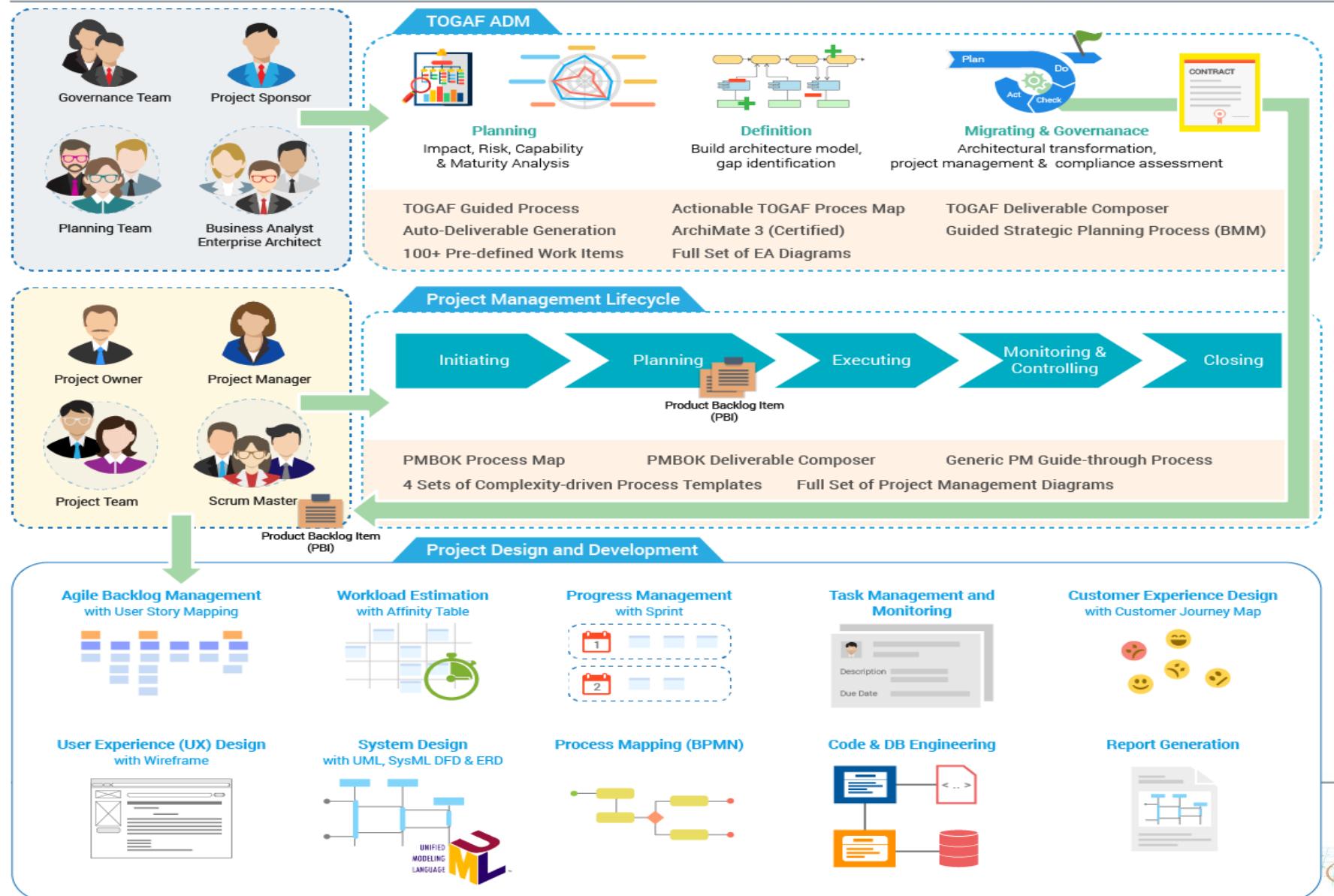


Arhitectura mediului CASE

- **Utilitarele pentru transformare** convertesc elementele obținute cu instrumentele de analiză în elemente ale proiectării.
- **Generatoarele de forme și rapoarte** sunt utilizate pentru a crea, modifica și testa prototipurile de forme și rapoarte și pentru a identifica datele care vor fi afișate sau colectate pentru fiecare formă și raport
- **Instrumentele pentru validare/verificare** generează rapoarte prin care se identifică inconsistențele, redundanța și lipsurile din diagrame, forme și rapoarte.
- **Instrumente pentru generarea automată a codului** pornind de la specificațiile de proiectare conținute în depozitul de date central (instrumente pentru generarea obiectelor bazei de date și a modulelor aplicației).

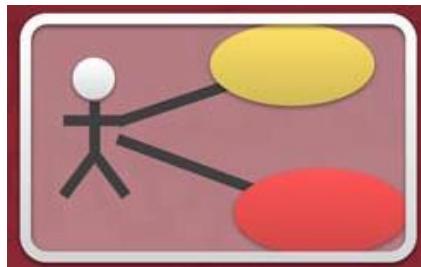


Visual Paradigm

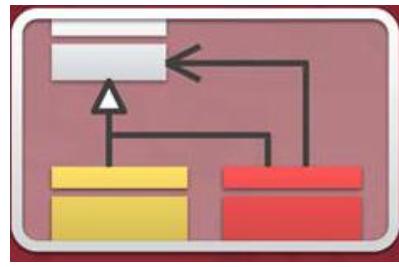


Visual Paradigm

1. Se axează pe trei direcții principale:



Identificarea
cerințelor



Construirea de
modele



Generarea de cod și
baze de date

2. Asigură interoperabilitatea cu alte instrumente CASE (Visio, Visual UML, Rational) și integrarea cu instrumentele IDE de marcă (Net Beans)
 3. Acoperă mare parte a ciclului de viață al unui sistem informatic
-



Visual Paradigm

	Enterprise	Professional	Standard	Modeler	Community
Diagrame UML	✓	✓	✓	✓	✓
Diagrame SysML	✓	✓	✓	✓	✓
Diagrama EA	✓	✓	✓	✓	✓
Sabloane predefinite	✓	✓	✓	✓	✓
Analiză textuală	✓	✓	✓	✓	✓
Diagrame BPMN	✓	✓	✓	✓	
Instrument de machetare	✓	✓	✓		
Harta Scrum User Story	✓	✓			
Ghid de management al proiectului	✓				



Visual Paradigm

Include modele ale unor limbaje standard

- *Modelarea UML* - Pot fi create toate tipurile de diagrame UML 2.x prin construirea de modele de cazuri de utilizare, comportamentale, de interacție, structurale, de amplasare, de cazuri de test.
- *Modelarea BPMN* – Pot fi create: diagrame de procese de afaceri, diagrame de flux de date, diagrame de hărți de procese, diagrama lanțului de procese conduse prin evenimente, organigrame. Diagramele de procese de afaceri pot fi exportate în BPEL.
- *Modelarea SysML* - SysML este un limbaj general pentru ingineria aplicațiilor și sistemelor informatici. VP-UML permite crearea diagramei de cerințe specifice limbajului SysML.



Visual Paradigm

Modelarea cerințelor

Identifică cerințele prin intermediul a mai multe mecanisme:

- *Analiza textuală* oferă editor de text prin intermediul căruia sunt înregisterate cerințele în format textual și care permite identificarea termenilor sau obiectelor importante (clase, cazuri de utilitate) pentru descrierea problemei.
- *Cardurile CRC* conțin informații precum descrierea clasei, atributele acesteia și responsabilitățile. Au formate proprii de afișare a informațiilor.
- *Diagrame de cerințe SysML* pentru a identifica cerințele funcționale sau non-funcționale ale sistemului.
- *Editorul de interfețe utilizator* prin intermediul căruia se crează machete ale ecranelor.
- *Managementul glosarului de termeni* prin care se identifică și se descrie vocabularul proiectului.



Visual Paradigm

Modelarea bazelor de date

- Se pot crea două tipuri de diagrame pentru modelarea bazelor de date: diagrame Entitate-Relație și diagrame ORM (pentru a vizualiza maparea dintre modelul de obiecte și modelul de date).
- Se pot modela nu numai caracteristici ale tabelelor, ci și proceduri stocate, declanșatori, secvențe și viziuni ale bazei de date într-o diagramă Entitate-Relație.
- Diagramele se pot construi de la zero sau prin inginerie inversă plecând de la o bază de date existentă.
- Sincronizare între diagrama de clase și diagrama Entitate-Relație pentru a asigura consistența între cele două modele.
- Generarea de cod SQL pornind de la modele.



Visual Paradigm

Generare de cod

- Generatoarele de inginerie inversă și directă oferă suport pentru ingineria modelelor. Facilitatea **Java Round-Trip engineering** permite sincronizarea continuă a codului și a modelului pentru limbajul Java.

Model	Inginerie directă	Inginerie inversă
Java	x	x
C++	x	x
XML Schema	x	x
PHP	x	x
Python Source	x	x
Objective-C	x	x
CORBA IDL Source	x	x
.NET dll sau fișiere .exe		x
CORBA IDL Source		x
XML (structure)		x
JDBC		x
Hibernate		x
C#	x	
VB.NET	x	
ODL	x	
ActionScript	x	
Delphi	x	
Perl	x	
Ada95	x	
Ruby	x	



Visual Paradigm

Interoperabilitate

Interoperabilitatea facilitează schimbul de modele cu alte instrumente.

Model	Import	Export
Telelogic Modeler	x	
Rational Rose	x	
ERwin Data Modeler project	x	
Rational Software Architect	x	
Rational DNX	x	
NetBeans 6.x UML diagramme	x	
Visio	x	
BPEL for Oracle workflow engine		x
BPEL for JBoss workflow engine		x
Diagram (JPG, PNG, SVG, EMF, PDF)		x
Microsoft Excel	x	x
EMF UML2 model	x	x
XMI (1.0, 1.2, 2.1)	x	x
XML (nativ)	x	x
Microsoft Word pentru modelul CU	x	x



Visual Paradigm

Integrarea cu medii IDE

- Oferă suport pentru întreg ciclul de dezvoltare a unui sistem informatic folosind pentru etapa de programare următoarele produse de tip IDE :
 - Eclipse
 - NetBeans/Sun ONE
 - IntelliJ IDEA

Generarea documentației

- Documentația poate fi partajată și proiectată împreună cu beneficiarii sistemului folosind unul dintre formatele: HTML (report generation) , HTML (project publisher) , PDF , Word.



ACADEMIA DE STUDII ECONOMICE BUCUREŞTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

PROIECTAREA SISTEMELOR INFORMATICE

- CURS 4 -

BUCUREŞTI
2020-2021

Diagrama de clase

*Modelarea
structurii
statice*

Sinteză: Analiza orientată obiect a sistemelor informaticе

În etapa de analiză a sistemului sunt analizate specificațiile și cazurile de utilizare, identificându-se cele mai importante concepte cu care lucrează sistemul, împreună cu relațiile dintre acestea. Se reprezintă grafic, printr-o diagramă, structura domeniului claselor pentru sistemul analizat.

1. Se inițiază reprezentarea **diagramei de clase**, care va fi finisată și în etapele următoare. Diagrama claselor reprezintă grafic **structura statică** a sistemului, prin includerea claselor identificate, a pachetelor și a relațiilor dintre acestea. Un pachet poate conține clase, interfețe, componente, noduri, colaborări, cazuri de utilizare, diagrame sau alte pachete.
2. Se inițiază construirea **diagramei de obiecte** care modelează **instanțele** elementelor conținute în diagramele de clase. Aceste diagrame cuprind un set de obiecte și relațiile dintre acestea la un anumit moment.



Sinteză: Analiza orientată obiect a sistemelor informative

3. Pentru a evidenția stările prin care poate trece un obiect sau un eveniment (mesaje primite, erori, condiții de realizare) sunt reprezentate **diagramele de stare**, care reprezintă ciclul de viață al obiectelor, subsistemelor și sistemelor. Stările obiectelor se schimbă la recepționarea evenimentelor sau semnalelor.
4. **Diagrama de activitate** este realizată cu scopul de a evidenția acțiunile și rezultatul acestor acțiuni și pentru a scoate în evidență fluxurile de lucru.
5. Pentru a evidenția interacțiunile dintre obiecte se construiesc **diagramele de interacțiune**: diagramele de secvență și, respectiv, diagramele de comunicare.
 - **Diagramele de secvență** descriu modul în care interacționează și comunică obiectele, prin focalizare pe mesajele care sunt transmise și recepționate.
 - **Diagramele de comunicare** permit reprezentarea atât a interacțiunilor, cât și a legăturilor dintre un set de obiecte care colaborează. Se utilizează când este utilă vizualizarea coordonatei spațiale.



Modelarea structurii statice. Diagrama de clase

Diagrama de clase este **instrumentul principal** de analiză și proiectare a structurii **statische** a sistemului.

În ea se precizează **structura** claselor sistemului, **relațiile** între clase și structurile de **moștenire**.

La proiectare se utilizează aceeași diagramă și se modifică pentru a fi conformă cu **detaliile de implementare**.



Diagrama de clase

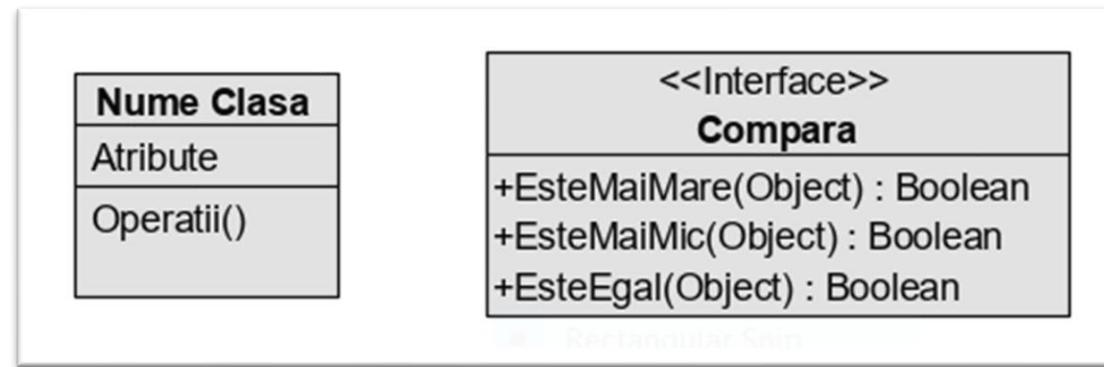
1. Scopul diagramei claselor este de a prezenta **natura statică** a claselor punând în evidență atributele, operațiile și asocierile.
2. Majoritatea instrumentelor de modelare orientate obiect generează **cod sursă** din diagrama claselor.
3. Celelalte diagrame UML furnizează diferite puncte de vedere din care să fie identificate atributele, operațiile și asocierile dintre clase. Ele ajută la **validarea** diagramei claselor, putând servi la clarificarea unei probleme specifice.



Definirea unei clase

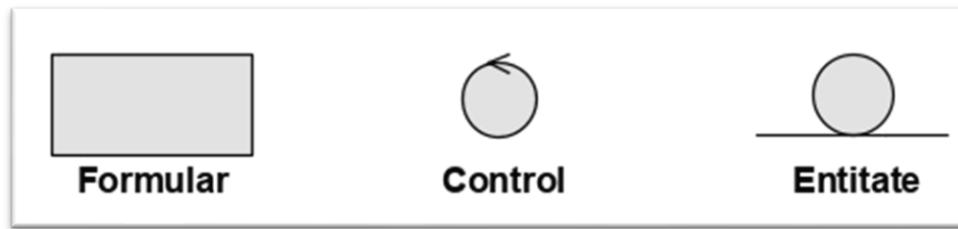
A

1. Ansamblu de obiecte care au aceleasi caracteristici și contrângeri.
2. Caracteristicile unei clase sunt attributele și operațiile.
3. Clasele *abstracte* nu pot fi instanțiate. Rolul lor este de a permite altor clase să le moștenească, în vederea reutilizării caracteristicilor.
4. O interfață descrie un set de caracteristici și obligații publice. Specifică, de fapt, un contract. Orice instanță care implementează interfața trebuie să ofere serviciile furnizate prin contract.



Exemple de clase stereotype uzuale

- entitate (<<entity>>) – o clasă pasivă, care nu inițiază interacțiuni;
- control (<<control>>) – inițiază interacțiuni, conține o componentă tranzacțională și este separator între entități și limite;
- limită (<<boundary>>) – este aflată la periferia sistemului, dar în interiorul său. Reprezintă limita de legătură cu actorul sau cu alte sisteme informatiche;
- enumerare (<<enumeration>>) - este folosită pentru definirea tipurilor de date ale căror valori sunt enumerate.
- primitivă (<<primitive>>) - o formă de clasă care reprezintă tipuri de date predefinite, cum ar fi tipul Boolean.



Atribute

1. Fiecare atribut este descris cel puțin prin numele său.
2. Este de preferat ca numele atributului să înceapă cu literă mică.
3. Se pot adăuga și informații adiționale, iar forma generală a unui atribut este:

**[vizibilitate][/]nume[:tip][multiplicitate][=valoare implicită]
[{proprietate}]**

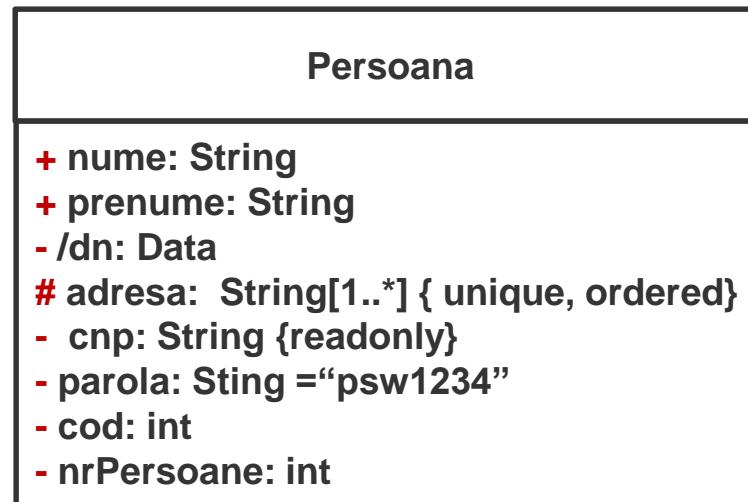


[vizibilitate][/]nume[:tip][multiplicitate][=valoare implicită] [{proprietate}]

Atribute - Vizibilitate

➤ Vizibilitatea poate fi:

- + public: poate fi văzută și folosită de oricine
- - private: numai clasa însăși poate avea acces
- # protected: au acces clasa și subclasele acesteia
- ~ package: numai clasele din același pachet pot avea acces



[vizibilitate][/]nume[:tip][multiplicitate][=valoare implicită] [{proprietate}]

Atribute – Atribut derivate și nume

- / simbolizează un atribut derivat
- acesta se calculează pe baza altor valori și este, de obicei, readonly
- numele sunt desemnate prin substantive

Persoana

```
+ nume: String  
+ prenume: String  
-/dn: Data  
# adresa: String[1..*] { unique, ordered}  
- cnp: String {readonly}  
- parola: String =“psw1234”  
- cod: int  
- nrPersoane: int
```

Persoana

```
+ nume: String  
+ prenume: String  
-/dn: Data  
# adresa: String[1..*] { unique, ordered}  
- cnp: String {readonly}  
- parola: String =“psw1234”  
- cod: int  
- nrPersoane: int
```

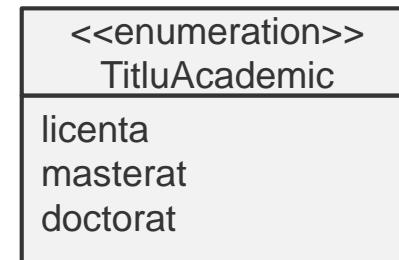
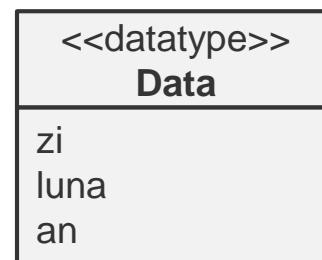
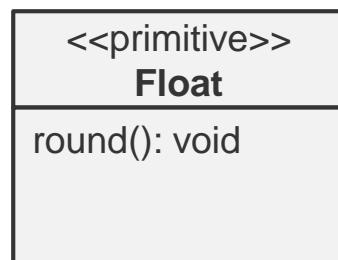


[vizibilitate][/]nume[:tip][multiplicitate][=valoare implicită] [{proprietate}]

Atribute – Tip de date

Tipuri de date

- Tipuri de date primitive
 - Predefinite: Boolean, Integer, String
 - Definite de utilizator: «**primitive**»
 - Tipuri de date compozite: «**datatype**»
- Enumerații: «**enumeration**»

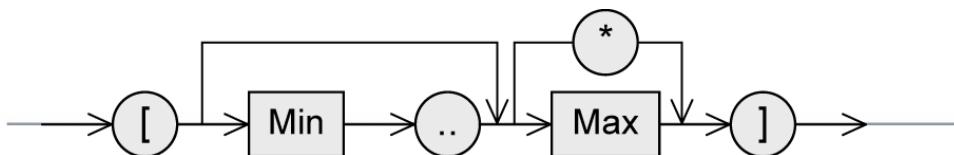


[vizibilitate][/]nume[:tip][multiplicitate][=valoare implicită] [{proprietate}]

Atribute -Multiplicitate

1. UML permite specificare multiplicităților pentru attribute, atunci când dorim să definim mai mult de o valoare pentru un atribut. Au următoarea semnificație:

Multiplicitate	Sens
1	Exact 1 (implicit)
2	Exact 2
1..4	De la 1 la 4 (inclusiv)
3, 5	3 sau 5
1..*	Cel puțin unul sau mai mulți
*	Nelimitat (inclusiv 0)
0..1	0 sau 1



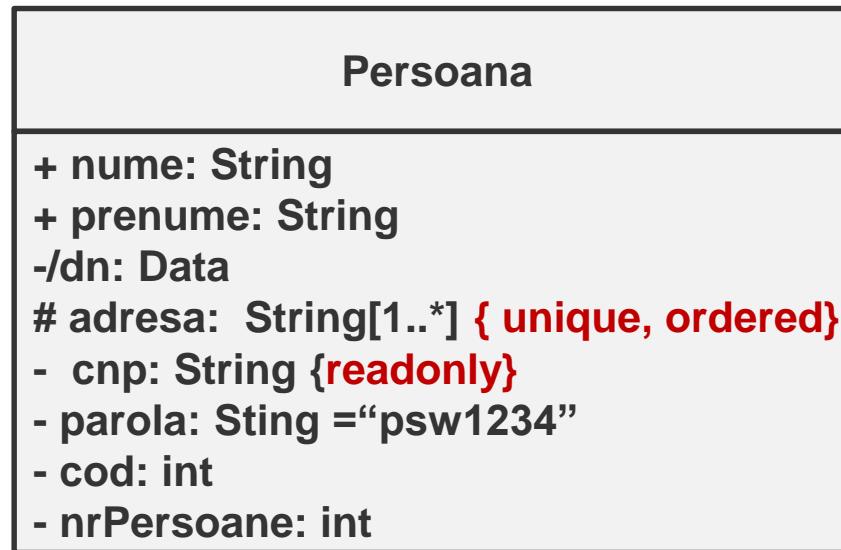
Persoana

```
+ nume: String  
+ prenume: String  
-/dn: Data  
# adresa: String[1..*] { unique, ordered}  
- cnp: String {readonly}  
- parola: String =“psw1234”  
- cod: int  
- nrPersoane: int
```

[vizibilitate][/]nume[:tip][multiplicitate][=valoare implicită] **{proprietate}**

Atribute - Proprietate

- Proprietate indică o proprietate suplimentară care se aplică atributului:
 - {readonly}: atributul poate fi citit, dar nu modificat
 - {ordered}, {unordered}: o mulțime ordonată sau neordonată
 - {unique}, {non-unique}: mulțimea poate conține sau nu elemente identice



Operații

1. Forma generală a unei operații este:

[vizibilitate] nume ([direcție] lista parametri) [:tip returnat] [{proprietate}]

1. Vizibilitatea – aceeași ca și la clase
2. Direcție - 'in' | 'out' | 'inout' | 'return'
3. Tip returnat – dacă operația returnează ceva, adică este o funcție
4. Un exemplu de proprietate a unei operații: {query} - nu are efecte secundare, nu schimbă starea unui obiect sau a altor obiecte, exemplu operațiile de tip “get”.

Exemple de atribute:

- - varsta: Integer {varsta>18}
- # nume:String[1..2] = "Ioana"
- ~ Id:String {unique}
- / sumaTotala:Real=0

Exemple de operații:

- + setVarsta (out varsta: Integer)
- + getVarsta(in Id:String): Integer {query}
- - schimbaNume(inout nume:String)



Constrângeri

1. O constrângere este o expresie care restricționează un anumit element al diagramei de clase.
2. Aceasta poate fi o expresie formală (scrisă în Object Constraint Language - OCL) sau o formulare semi-formală sau informală.
3. Acestea sunt reprezentate între acolade.
4. Pot fi scrise imediat după definirea elementului sau ca un comentariu.
5. O constrângere poate avea și un nume, astfel:
{nume : expresie booleană }

Exemple de constrângeri OCL:

context Organizatie

inv: self.departamente → isUnique (nume)

inv: departamente.angajati → isUnique (cod)

Produs
-nume : String {nume->NotEmpty()}
-pret : Real {pret>0}



Relații între clase

Relația de asociere implică stabilirea unei relații între clase.

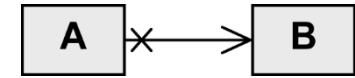
1. Este caracterizată prin:

- denumire (optională)
- multiplicități – se trec la cele 2 capete ale asocierii;
- roluri ale asocierii: se trec la fiecare capăt al asocierii și conțin o descriere scurtă și reprezentativă (1 – 2 substantive)
- direcție de navigare

1. Tipuri de asocieri:

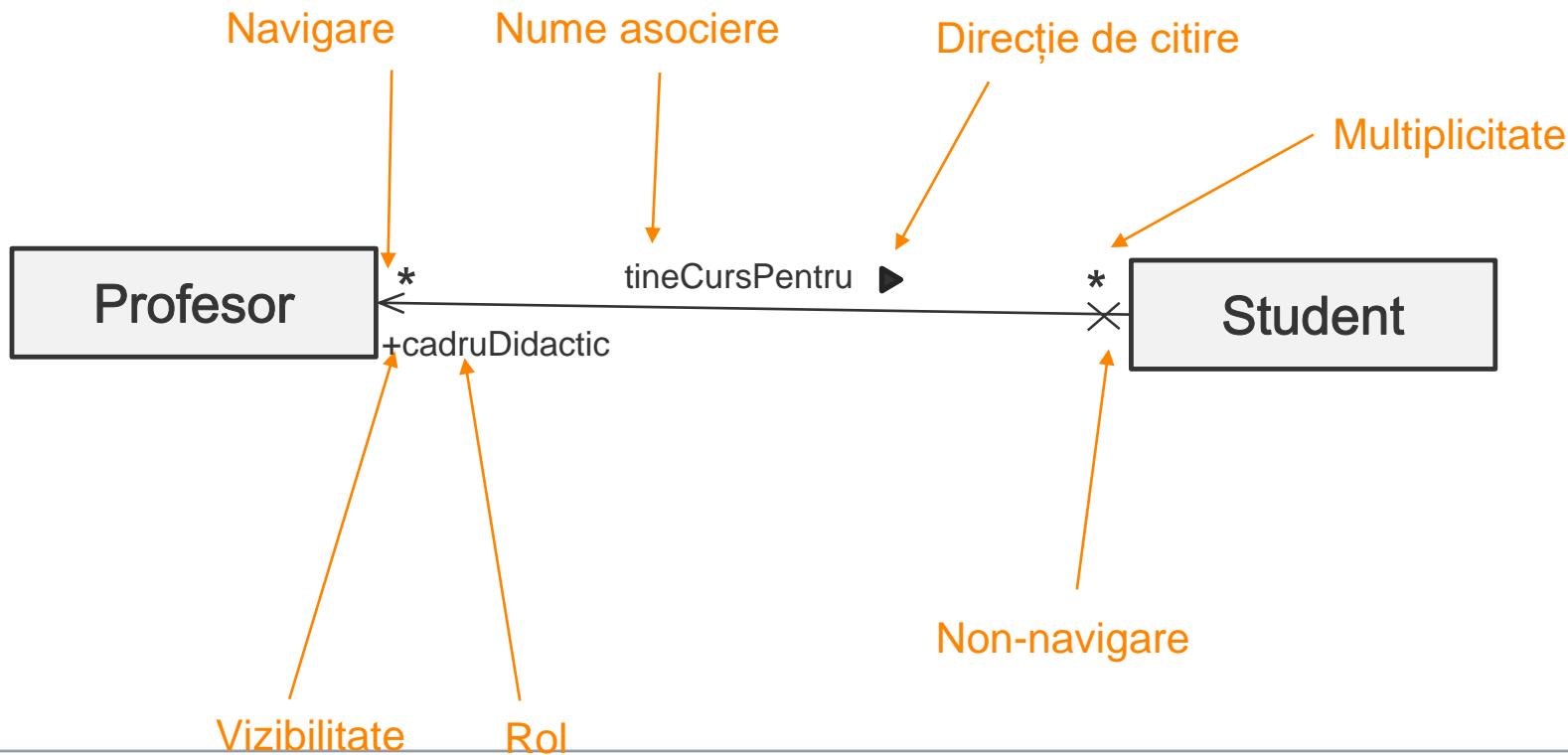
- unare
- binare
- n-are





Asocierea binară

1. Conectează între ele instanțele a două clase.



Asocierea binară - Navigare

1. **Navigare**: un obiect cunoaște obiectul său partener și poate accesa attributele și operațiile vizibile ale acestuia

- Indicată prin intermediul unui arc

2. **Non-navigare**

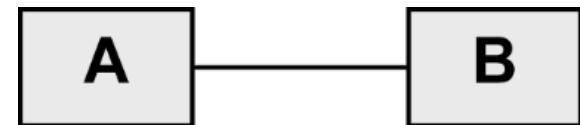
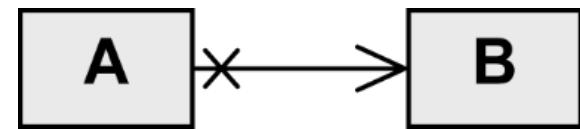
- Indicată prin cruce

3. **Exemple:**

- A poate accesa attributele și operațiile vizibile ale lui B
- B nu poate accesa niciun atribut și operație a lui A

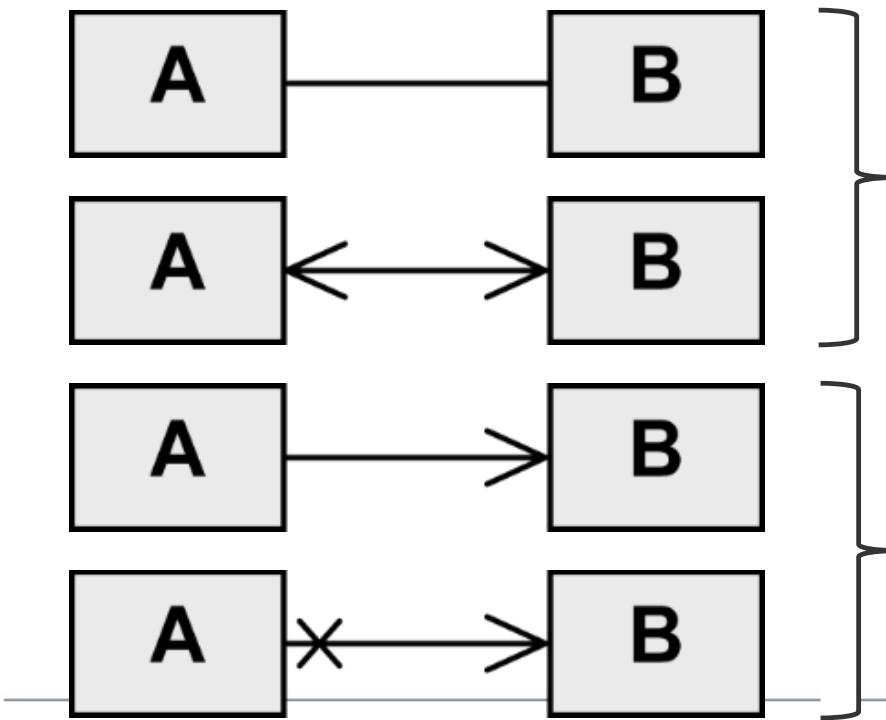
4. **Navigare nedefinită**

- Navigarea implicită este bidirectională

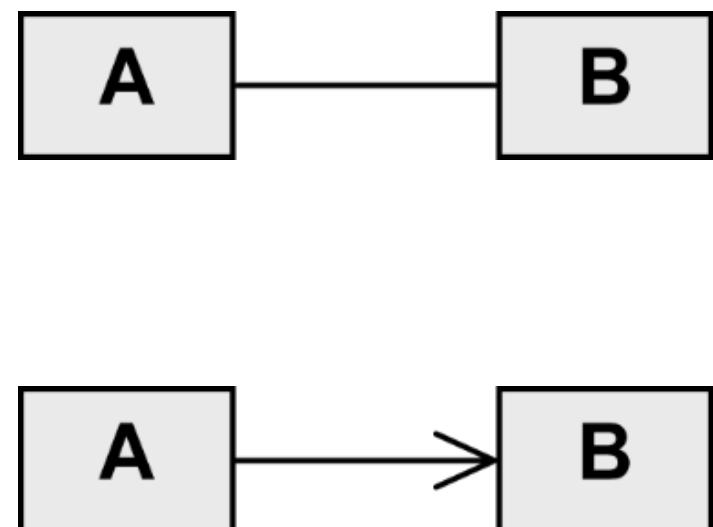


Navigare- Standard UML vs. bune practici

Standard UML

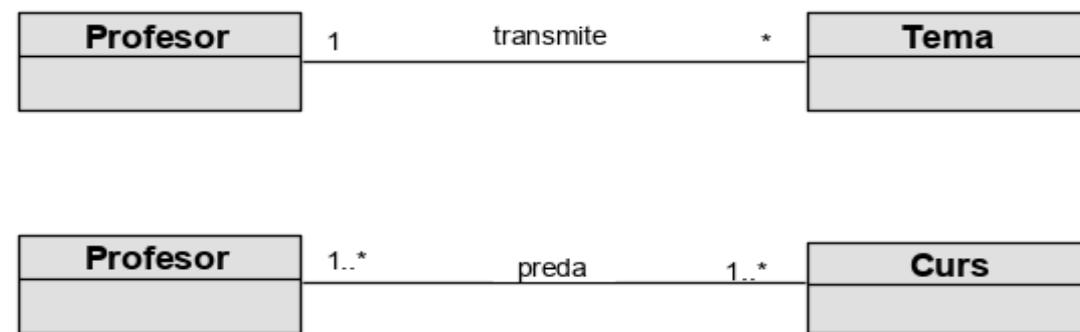


Bune practici

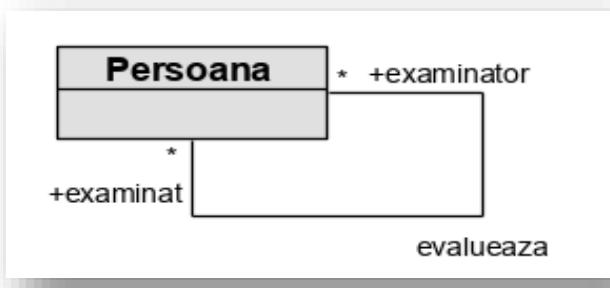


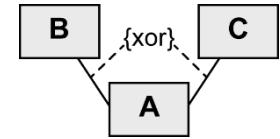
Asocierea – Multiplicitate și rol

1. Multiplicitate: numărul de obiecte alei unei clase care pot fi asociate cu exact un obiect al clasei de la capătul opus al asocierii



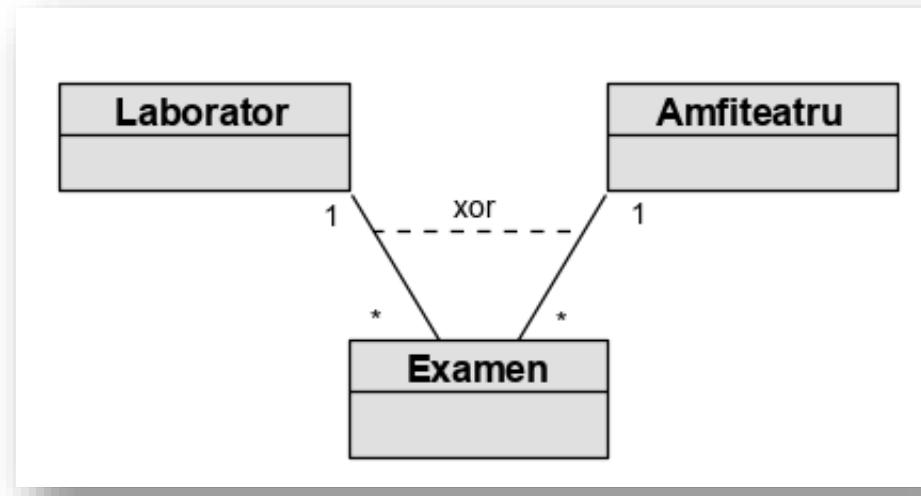
2. Rol: descrie modul în care un obiect este implicat într-o relație de asociere

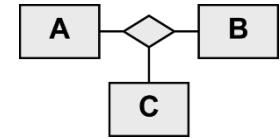




Asocierea binară – constrângere xor

1. Constrângere “sau exclusiv”
2. Un obiect al clasei **A** se poate asocia cu un obiect al clasei **B** sau cu un obiect al clasei **C**, dar nu cu amândouă în același timp.



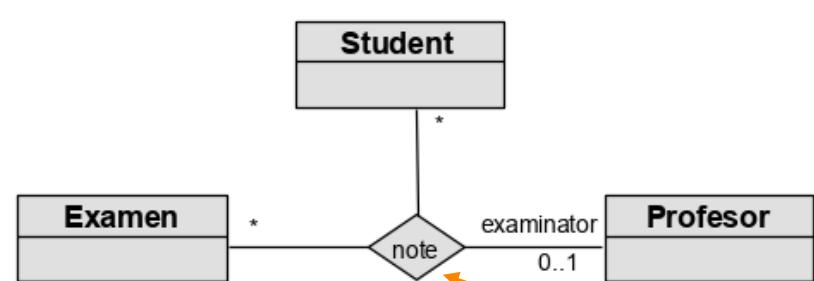


Asocierea n-ară

Mai mult de două obiecte partenere sunt implicate într-o relație.

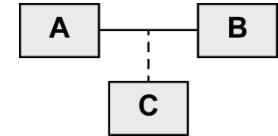
Nu există direcții de navigare

- **(Student, Examen) → (Profesor)**
 - Un student susține un examen cu unul sau niciun profesor
- **(Examen, Profesor) → (Student)**
 - Un examen cu un profesor poate fi susținut de oricărți studenți
- **(Student, Profesor) → (Examen)**
 - Un student poate fi evaluat de un profesor pentru oricâte examene



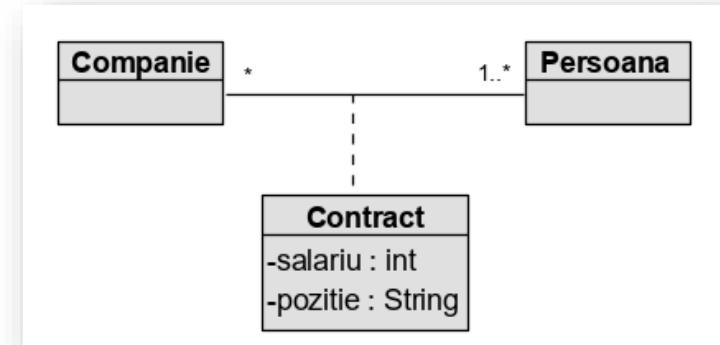
Asociere ternară



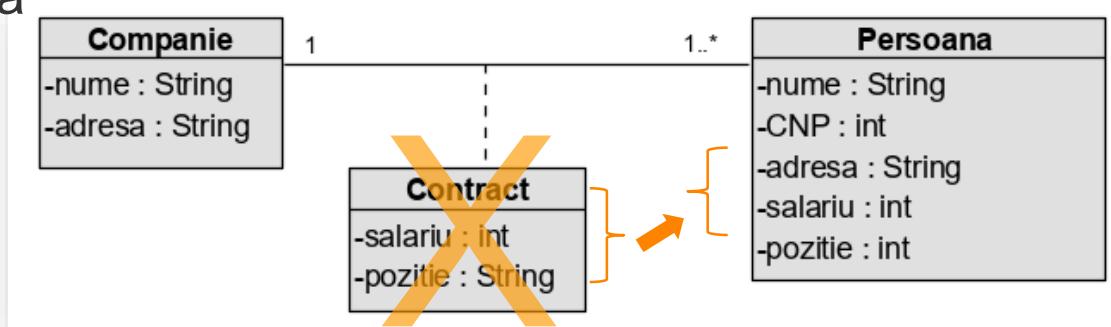


Asociere modelată ca o clasă

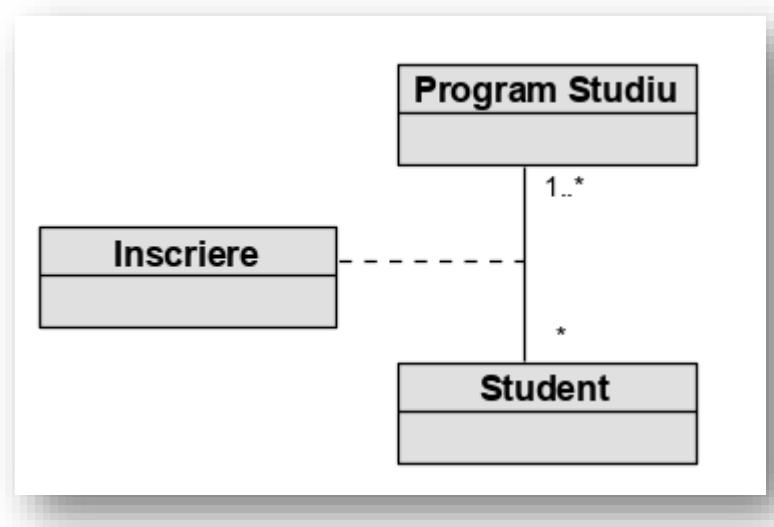
Asignează attribute unei relații între clase și nu clasei în sine.
Necesară atunci când modelăm asocieri multi-la-multi



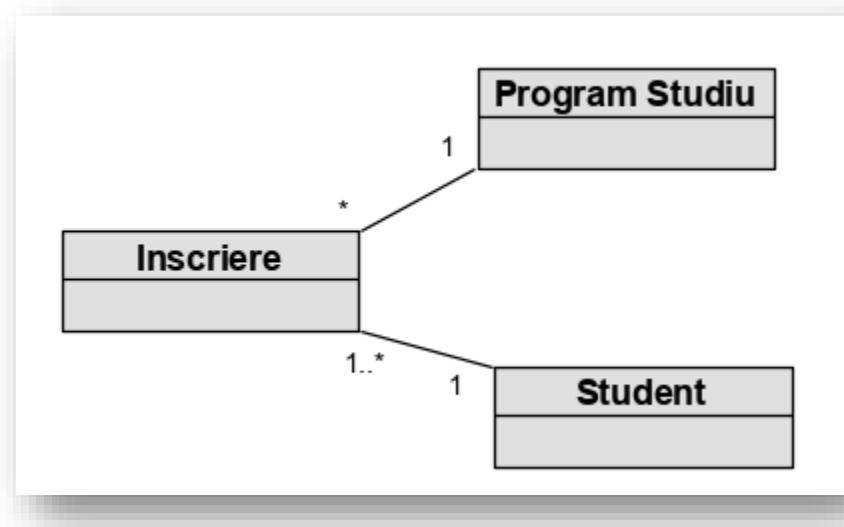
Pentru asocierile unu-la-unu sau unu-la-mulți este posibilă, dar nu necesară



Asociere modelată ca o clasă vs. clasă



≠



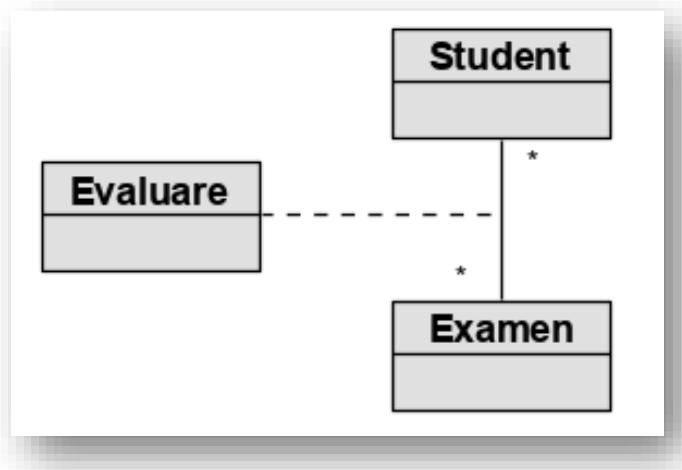
Un Student se poate înscrie la un anumit program de studiu **o singură dată**

Un Student se poate înscrie la un anumit program de studiu de **mai multe ori**



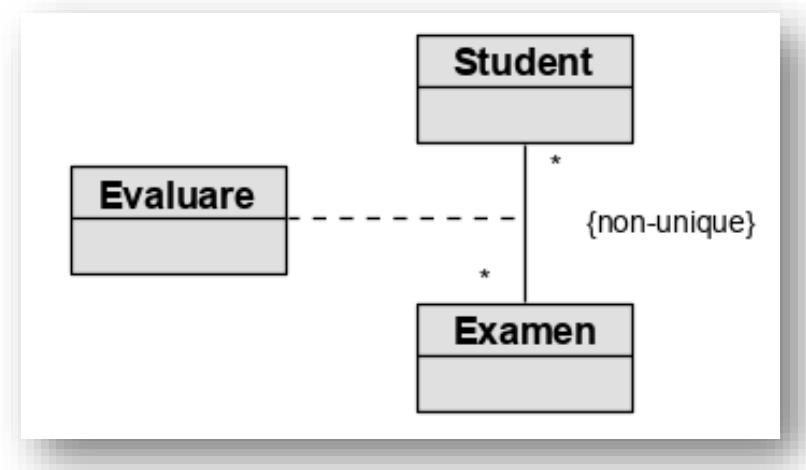
Asociere modelată ca o clasă – unique/ non-unique

Implicit: fără duplicate



Unui student i se permite să fie evaluat pentru un examen **o singură dată**.

Non-unique: cu duplicate



Unui student i se permit **mai multe** evaluări pentru un examen.

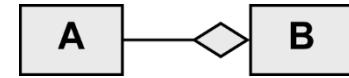


Agregarea

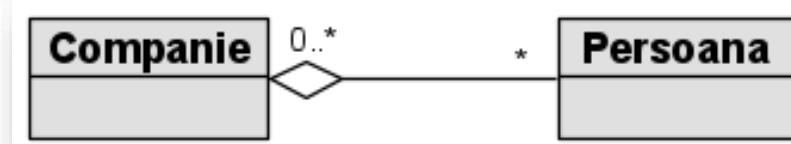
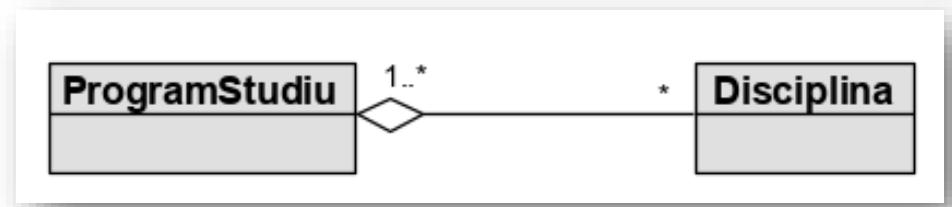
- Este o formă specială de asociere binară
- Reprezintă o relație de tip parte/întreg
- Este folosită pentru a arăta că o clasă este parte a altrei clase
- Proprietăți ale agregării:
 - *Tranzitivă*: dacă **B** este parte din **A** și **C** este parte din **B**, atunci **C** este și parte din **A**.
 - *Asimetrică*: nu este posibil ca **A** să fie parte din **B** și **B** să fie parte din **A**, în același timp
- Poate fi de două tipuri:
 - Agregare partajată (agregare)
 - Agregare compusă (componere)



Agregarea partajată



- Exprimă o formă slabă de apartenență a părții la întreg
- Părțile pot exista independent de întreg
- Multiplicitatea la capătul agregat poate fi > 1
- Aceleași părți partajate pot fi incluse simultan în mai multe clase întreg
- Dacă se sterge clasa întreg, clasele parte vor exista în continuare
- *Sintaxa:* se reprezintă sub forma unui romb gol plasat la capătul clasei întreg
- *Exemplu:* Disciplina este parte a unui program de studiu



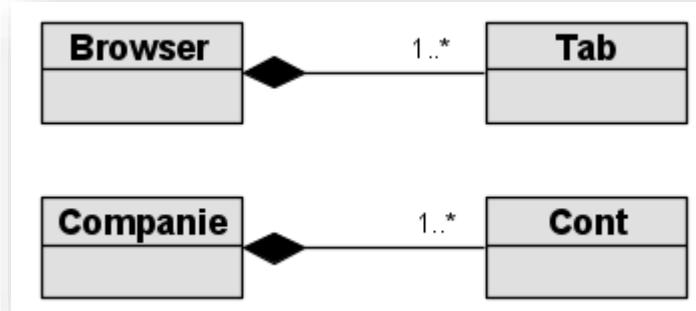


Agregarea compusă

- Exprimă o formă puternică de apartenență a părții la întreg
- Multiplicitatea la capătul agregat este maxim 1
 - ✓ Aceleași părți partajate nu pot fi incluse simultan în mai multe clase întreg
- Dacă se sterge clasa întreg, se vor sterge și clasele parte
- Sintaxa: se reprezintă sub forma unui romb plin plasat la capătul clasei întreg
- Example: Proiectoarul este parte a unei săli, iar sala este parte a unei clădiri



Dacă este ștearsă Clădirea,
se va sterge și Sala



Relații de asociere

Asociere

Obiectele știu unele de existența celorlalte și pot lucra împreună.

Agregare

1. Protejează integritatea configurației.
2. Funcționează ca un tot unitar.
3. Control prin intermediul unui singur obiect.

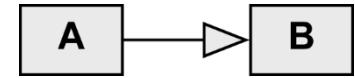
Componere

Fiecare parte poate fi membră a unui singur obiect aggregat.

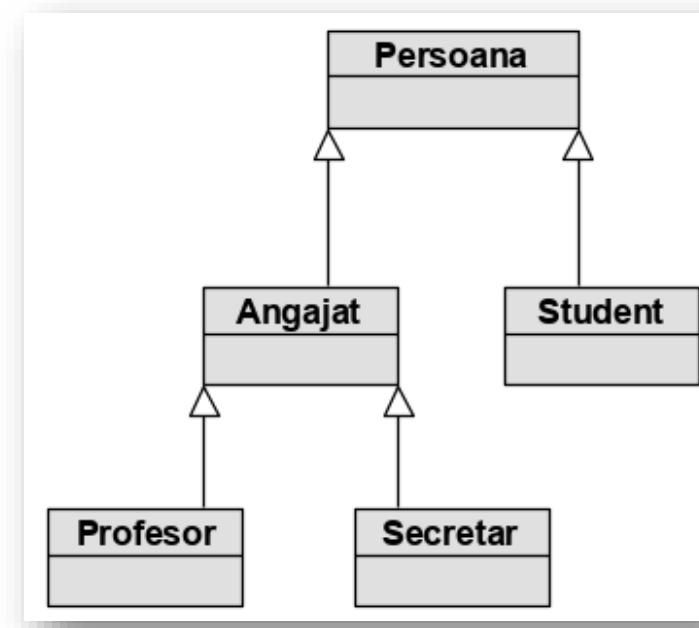
Relații între asociere, agregare și compunere



Generalizarea

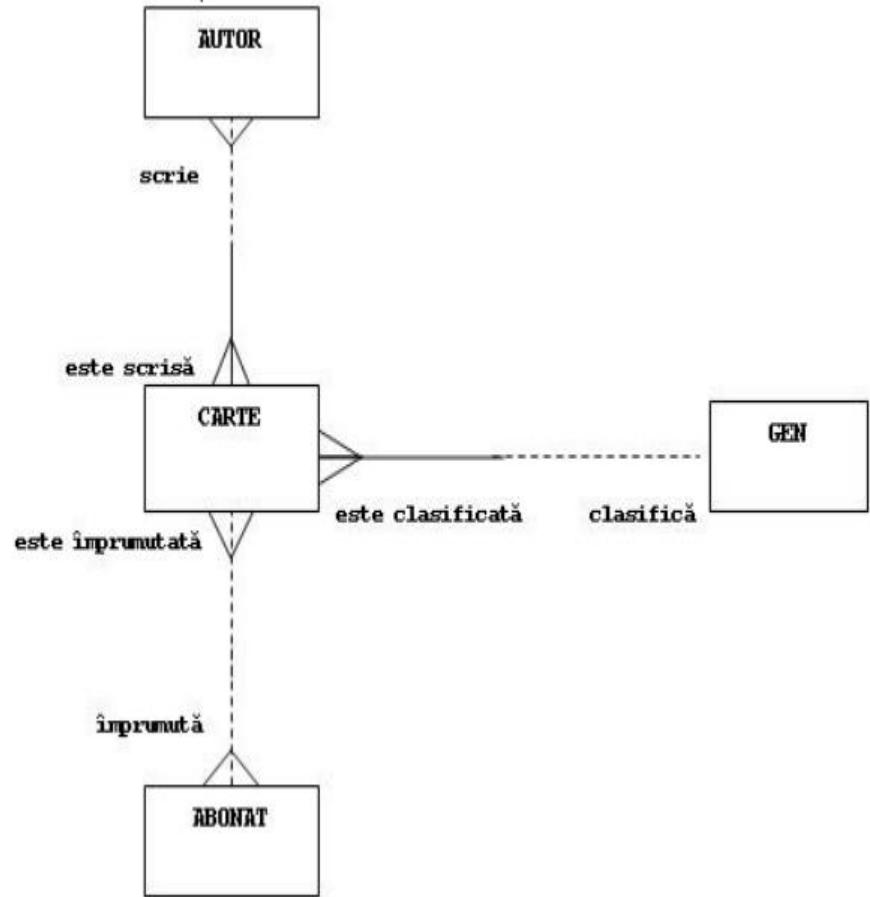


- Caracteristicile (atributele și operațiile), asocierile și agregările care sunt specifice pentru o clasă generală (superclasă) sunt transferate subclaszelor sale.
- Se mai numește informal și relație de genul “este un tip de”.
- Se reprezintă sub forma unui triunghi gol plasat la capătul superclasei.
- Clasele abstrakte sunt intens utilizate în contextul generalizării.
- Este permisă moștenirea multiplă.
- Subclasele își pot defini propriile caracteristici, atrbute și agregări.



În modelarea structurată

1. Pentru identificarea entităților din domeniul analizat, în analiza structurată se folosea **diagrama entitate asociere**
2. Acesta nu este inclusă în UML, dar este folosită pe larg de analiști în proiectarea bazelor de date și de administratorii de baze de date în implementarea bazei de date
3. Nu poate fi folosită pentru a reprezenta *relații de generalizare* sau *relații parte-intreg*



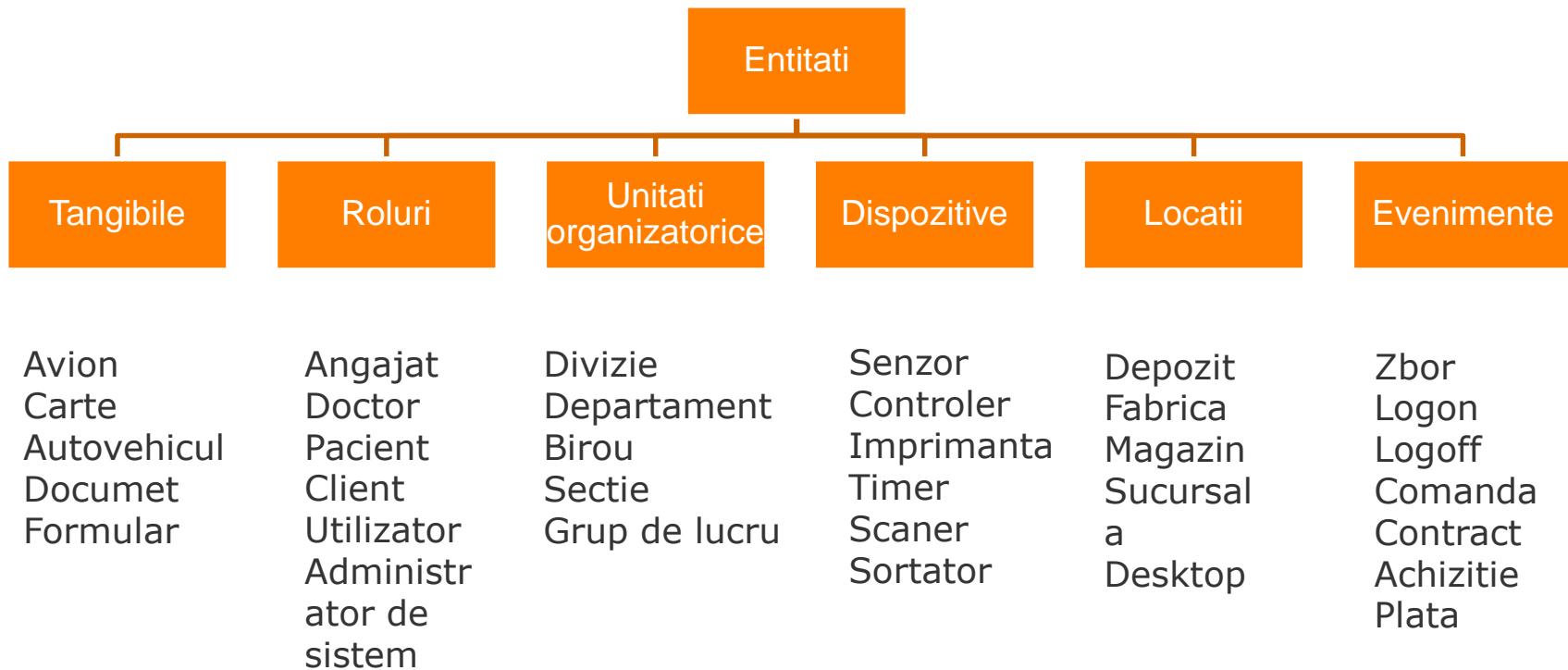
Tehnici de identificare a claselor

- a) **Tehnica brainstorming** – se utilizează o lista de control cu toate tipurile de entități care apar în mod frecvent și se realizează ședințe de brainstorming pentru a identifica clasele domeniului pentru fiecare dintre tipurile respective
- b) **Tehnica substantivelor** – se identifică toate substantivele care apar în descrierea sistemului și se hotărăște dacă substantivul este o clasă a domeniului, un atribut sau un element neimportant



a. Tehnica de brainstorming

1. Avem elemente de genul...

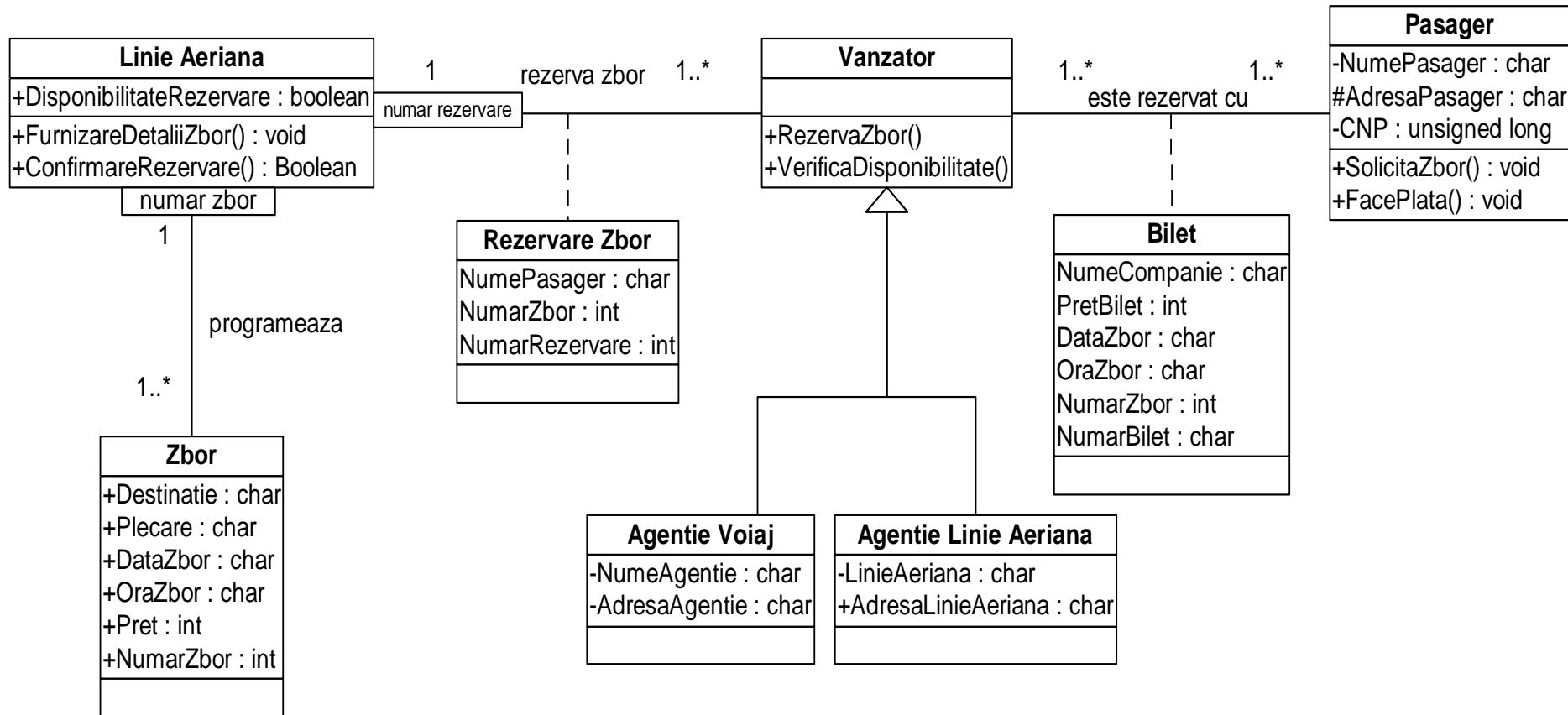


b. Tehnica substantivelor

1. Se identifică **toate substantivele** pe baza cazurilor de utilizare, actorilor și altor informații despre sistem
 2. Se **preiau informațiile** necesare din sistemele, procedurile, formularele și rapoartele existente
 3. Se **rafinează** lista de substantive
 - Apartine ariei de cuprindere a sistemului analizat?
 - Este nevoie să pastrăm mai mult de un astfel de element?
 - Este un sinonim pentru un element deja identificat?
 - Este un atribut al altui element înregistrat?
 - Este doar un output al sistemului obținut pe baza unui alt element deja identificat?
 - Este doar un input care rezultă din înregistrarea altui element identificat?
 4. Se crează o **listă principală de substantive** și se notează care trebuie inclus/exclus/ discutat
 5. Se prezintă lista utilizatorilor, beneficiarilor, membrilor echipei spre **validare**
-



Diagama claselor



Tehnica fișelor CRC

- Este utilizată uneori ca extensie a UML-ului pentru o analiză orientată pe responsabilități.
- Definițiile claselor sunt **rafinate** pe baza responsabilităților lor și a altor clase cu care acestea colaborează pentru a îndeplini aceste responsabilități.
- Pentru fiecare clasă se întocmește o **fișă** pe care se evidențiază responsabilitățile clasei și care sunt clasele cu care ea **colaborează** pentru a îndeplini aceste responsabilități.
- Aceste informații se transpun direct într-o **diagramă** a claselor, responsabilitățile corespund **metodelor**, iar colaborările corespund **asocierilor** dintre clase.



Tehnica fișelor CRC

Pasager	
Superclasă: <none>	
Subclase: <none>	
Responsabilități	Colaboratori
Plata biletului	Vanzator_bilet
Solicitare zbor	Vanzator_bilet



Pentru a afla prețul trebuie să colaborez cu pasagerul și compania aeriană

Bilet	
Superclasă: <none>	
Subclase: <none>	
Responsabilități	Colaboratori
Află pretul	Companie, Pasager

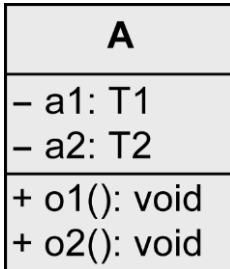
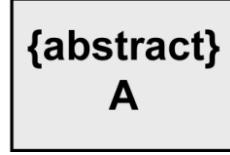
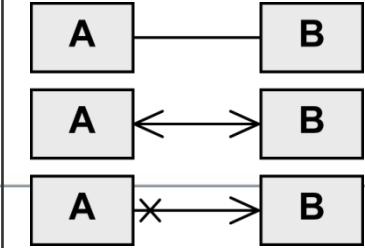
Companie	
Superclasă: <none>	
Subclase: <none>	
Responsabilități	Colaboratori
Furnizează detalii zbor	Vanzator_bilet, Zbor
Confirmă rezervarea	Vanzator_bilet, Zbor

Zbor	
Superclasă: <none>	
Subclase: <none>	
Responsabilități	Colaboratori
Locuri disponibile	<none>

Vanzator_bilet	
Superclasă: <none>	
Subclase: <none>	
Responsabilități	Colaboratori
Verifică disponibilitatea	Companie, Zbor
Rezervă zbor	Companie, Pasager

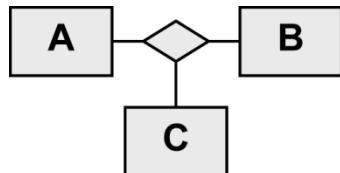
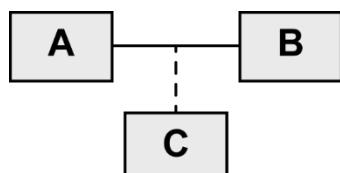
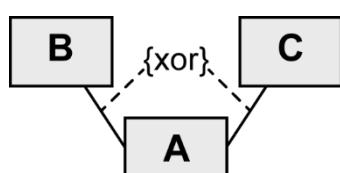


Notății - 1

Nume	Notație	Descriere
Clasă		Descrie structura și comportamentul unui set de obiecte
Clasă abstractă		Clasă care nu poate fi instantiată
Asociere		Relații între clase: navigare nespecificată, navigare în ambele direcții, non-navigare într-o direcție



Notății - 2

Nume	Notăție	Descriere
Asociere n-ară		Relație între n clase (aici trei clase)
Asociere modelată ca o clasă		Descriere mai detaliată a unei asocieri
Relație xor		Un obiect A își este într-o relație cu un obiect B sau cu un obiect C , dar nu cu amândouă simultan



Notății - 3

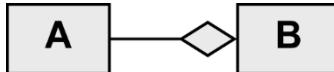
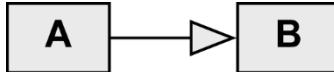
Nume	Notăție	Descriere
Agregare partajată		Relație de tip parte/întreg (A este parte din B)
Agregare compusă		Relație puternică de tip parte/întreg (A este parte din B)
Generalizare		Relație de moștenire (A moștenește de la B)



Diagrama de obiecte

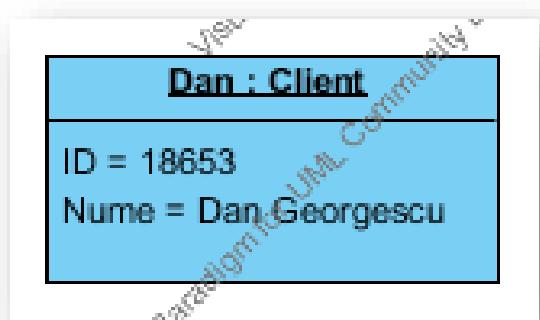
*Modelarea
statică*

Diagrama de obiecte

1. Conță din obiecte și legăturile dintre acestea.
2. Are rolul de a valida diagrama de clase.
3. O legătură reprezintă o relație între două obiecte.

Diagrama de obiecte	Diagrama de clase
Modelează fapte despre anumite entități.	Modelează reguli pentru tipuri de entități.
Reprezintă obiecte reale.	Reprezintă abstractizări ale conceptelor.
Leagă între ele obiecte.	Asociază entități.

- Un obiect este denumit folosind numele acestuia, semnul ":" urmat de numele clasei căreia îi aparține: *nume obiect : nume clasa* .
- Pot exista și obiecte anonime, denumite doar prin numele clasei.

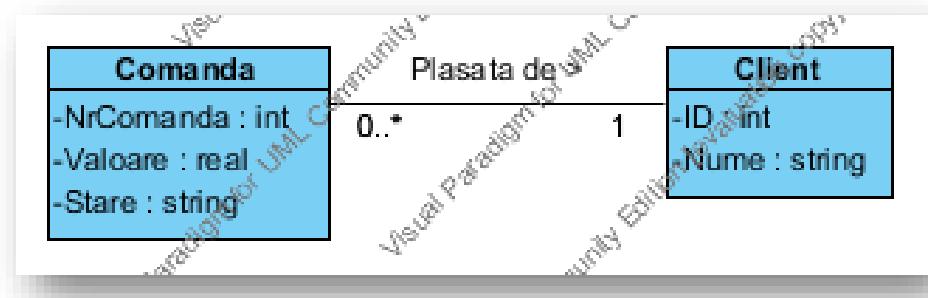


Notăriile diagramei de clase și obiecte - comparație

Diagrama de clase	Diagrama de obiecte
Clasa are trei compartimente: nume, attribute și operații.	Obiectul are numai două compatriamente: nume și attribute.
Numele clasei este specificat singur în primul compartiment.	Formatul numelui unui obiect include și numele clasei, toată expresia fiind subliniată. Aceste notații vor fi întâlnite și în alte diagrame care reprezintă obiecte.
Al doilea compartiment descrie proprietăți sub forma atributelor.	Al doilea compartiment definește valori pentru fiecare atribut, pentru testarea modelului.
Operațiile apar în descrierea clasei.	Operațiile nu sunt incluse în obiecte, deoarece ele sunt identice pentru fiecare obiect al clasei.
Clasele sunt conectate prin asocieri, având un nume, multiplicitate, constrângerile și roluri. Clasele sunt o abstractizare a obiectelor, deci este necesar să specificăm câte clase participă într-o asociere.	Obiectele sunt conectate printr-o legătură, care poate avea un nume, roluri, dar nu și multiplicități. Obiectele reprezintă entități singulare, toate legăturile sunt unu-la-unu, iar multiplicitățile sunt irelevante.

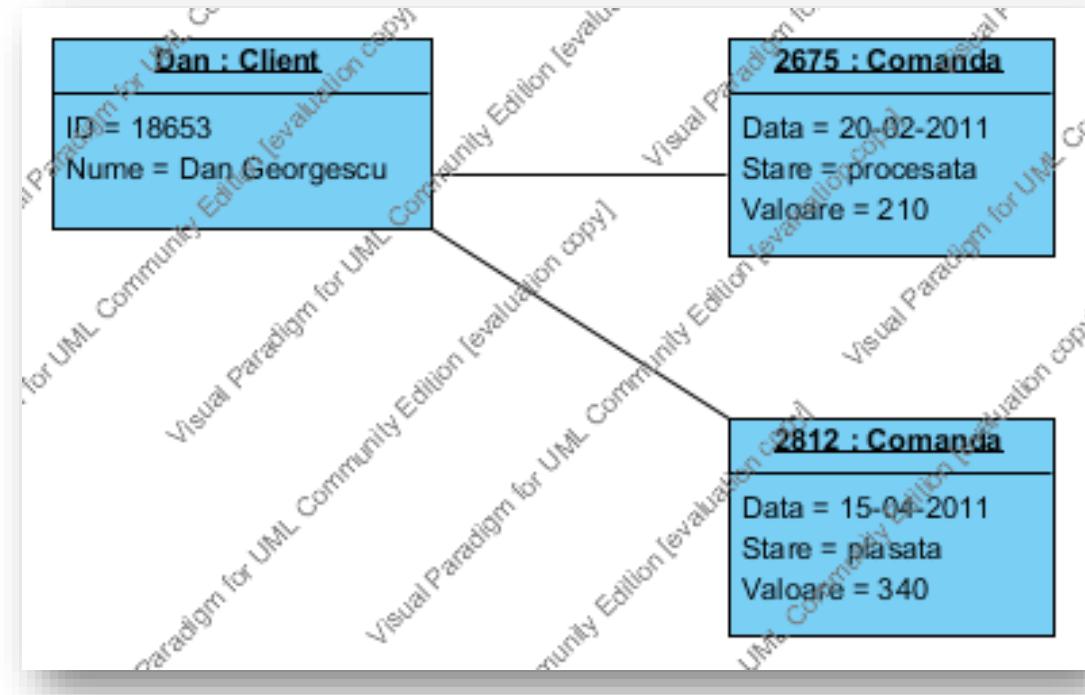
Diagrama de obiecte în Visual Paradigm

1. Se definește diagrama de clase în care clasele au specificate atrbute.



2. Se definește un obiect în diagrama de obiecte (Instance Specification).
3. Se selectează clasa căreia îi aparține obiectul: Click dreapta pe obiect -> Select Classifier-> se bifează și selectează clasa corespunzătoare
4. Opțional, se dă un nume obiectului.
5. Se definesc valorile pentru atrbute: Click dreapta pe obiect -> Slots, Define Slots (pentru atrbutele cărora vrem să le dăm valori) ->Edit Values-> Add -> Text (se introduce valoarea dorită).
6. Se creează legături (Link) între obiecte.

Diagrama de obiecte în Visual Paradigm - exemplu





Recapitulare

1. Ce este și cum se reprezintă multiplicitatea la nivelul asocierilor dintre clase? Dar la nivelul atributelor?
 2. Denumiți domeniile de vizibilitate în UML.
 3. Care este rolul moștenirii și cum se identifică?
 4. Clasele abstracte pot fi instantiată?
 5. De ce sunt importante numele de rol?
 6. Care este diferența dintre agregare și compunere?
 7. Ce tip are navigarea implicită la asocierea dintre clase?
 8. Cum se reprezintă grafic relația de generalizare?
 9. Care este rolul diagramei de obiecte?
 10. Câte comparte mente are reprezentarea grafică a unui obiect?
 11. Cum modelați relațiile dintre clasele următoare: Casa, Bucatarie, Baie, Dormitor, CutiePostala și Camera?
-



Cursul 5...

- Diagrama de activitate
- Mecanism de extensie

ACADEMIA DE STUDII ECONOMICE BUCUREŞTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

PROIECTAREA SISTEMELOR INFORMATICE

- CURS 5 -

BUCUREŞTI
2020-2021

Diagrama de activitate

Modelarea funcțională

Cuprins

- Introducere
 - Activități
 - Acțiuni
 - Fluxuri
 - Flux de control
 - Flux de obiecte
 - Nod inițial, nod final al activității, nod final al fluxului
 - Căi alternative
 - Căi concurente
 - Noduri obiect
 - Acțiuni bazate pe evenimente și acțiuni care apelează comportament
 - Partiții
 - Tratarea excepțiilor
-



Introducere

- Scopul diagramei de activitate: modelarea aspectelor care țin de procesarea procedurală.
- Ajută la reprezentarea vizuală a secvențelor de acțiuni prin care se dorește obținerea unui rezultat.
- Se poate realiza pentru unul sau mai multe cazuri de utilizare sau pentru descrierea unor operații complexe.
- Nu se construiește pentru fiecare caz de utilizare și scenariu, deoarece nu este necesar, ci numai pentru cele importante.
- Descrie fluxul de lucru dintr-un punct de plecare până într-un punct de terminare, detaliind căile de decizie care pot apărea într-o activitate.



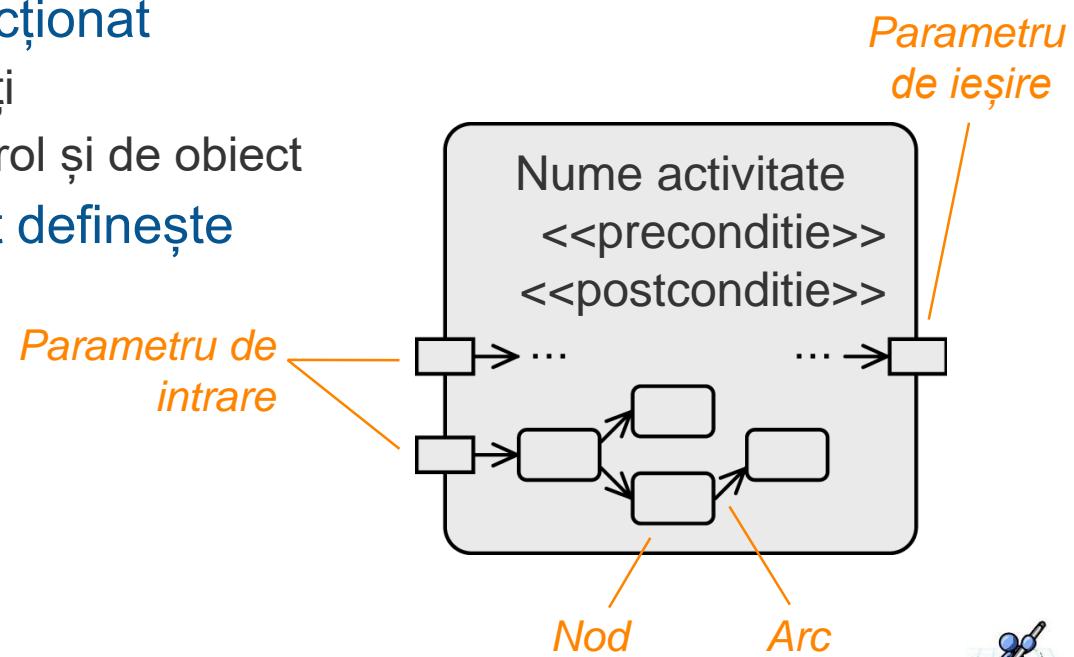
Introducere

- Este importantă în modelarea proceselor de afaceri.
- Poate fi folosită pentru a descrie procesare paralelă.
- Este bazată pe:
 - limbajele pentru definirea proceselor de afaceri
 - concepte consacrate care descriu procesarea concurrentă (rețele Petri)
- Conceptele și notațiile acoperă o gamă largă de aplicații
 - nu se reduc doar la modelarea orientată-obiect
- Permite definirea activităților independent de obiecte
 - se pot modela biblioteci de funcții, procese de afaceri...



Activitate

- Specifică comportamentul definit de utilizator la diferite niveluri de granularitate
- Exemple:
 - Definirea comportamentului unei operații sub formă de instrucțiuni atomice
 - Modelarea cursului de acțiuni dintr-un caz de utilizare
 - Modelarea proceselor de afaceri
- O activitate este un graf direcționat
 - Nodurile: acțiuni și activități
 - Arce: pentru fluxul de control și de obiect
- Fluxul de control și de obiect definește execuția
- Optional:
 - Parametri
 - Pre- și post-condiții

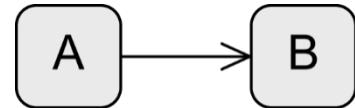


Acțiune

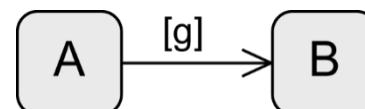
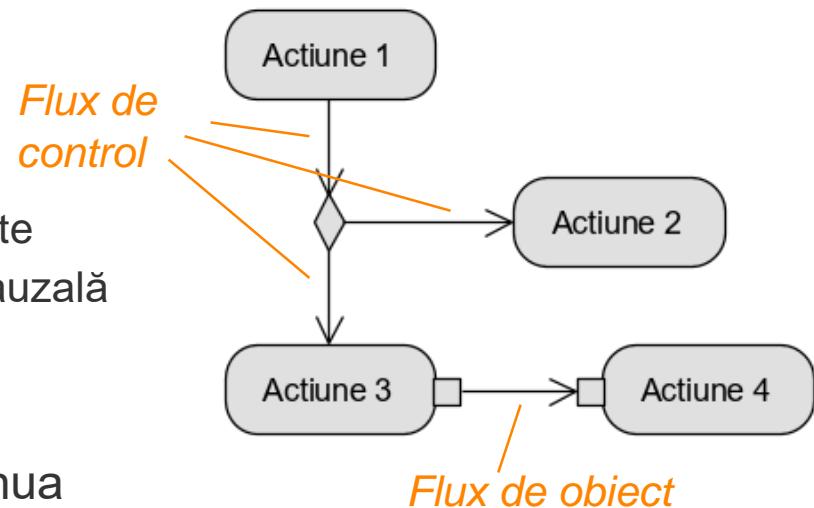
- **Elemente de bază** cu ajutorul cărora se poate specifica comportamentul
- **Sunt atomice**, deci nu mai pot fi descompuse
- Nu se specifică reguli pentru descrierea unei acțiuni
→ Se pot defini în limbaj natural sau în orice limbaj de programare
- Procesează valorile de intrare pentru a produce valori de ieșire
- Există notății speciale pentru diferite tipuri de acțiuni, cum ar fi:
 - Acțiuni bazate pe evenimente
 - Acțiuni care apelează comportament



Arce

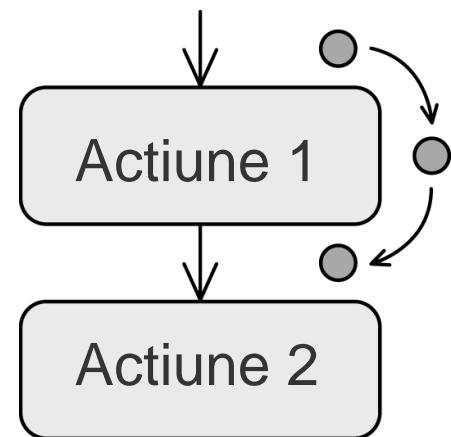


- Conectează între ele activitățile și acțiunile
- Exprimă ordinea de execuție
- Tipuri
 - Arc flux de control
 - Definește ordinea nodurilor
 - Arc flux de obiect
 - Folosit pentru a transmite date sau obiecte
 - Exprimă o dependență bazată pe date/cauzală între noduri
- Condiție (Guard)
 - Fluxul de control și de obiecte va continua doar dacă acea condiție specificată între paranteze este adevărată



Jeton (Token)

- **Mecanism virtual de coordonare** care descrie cu exactitate execuția
 - Nu este inclus în notațiile diagramei
 - Mecanism care acordă acțiunilor permisiunea de execuție
- Dacă o acțiune primește un jeton, atunci acțiunea poate fi executată
- Când o acțiune este finalizată, aceasta transmite jetonul următoarei acțiuni, declanșând execuția acesteia
- Condițiile pot preveni transmiterea jetonului
 - Jetonul este stocat în nodul anterior
- **Jeton control și jeton obiect**
 - **Jeton control:** "permisiunea de execuție" pentru un nod
 - **Jeton obiect:** transport date + "permisiunea de execuție"



Începerea și terminarea activităților

● Nod inițial

- Începe execuția unei activități
- Transmite jetoane tuturor fluxurilor de ieșire
- Păstrează jetoane până când nodurile succese le acceptă
- Pot exista noduri inițiale multiple pentru modelarea concurenței

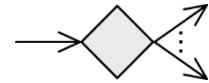
● Nod final al activității

- Încheie toate fluxurile unei activități
- Primul jeton care ajunge la nodul final al activității înceie întreaga activitate
 - Inclusiv sub-fluxuri concurente
- Sunt șterse alte jetoane de control sau obiect

⊗ Nod final al fluxului

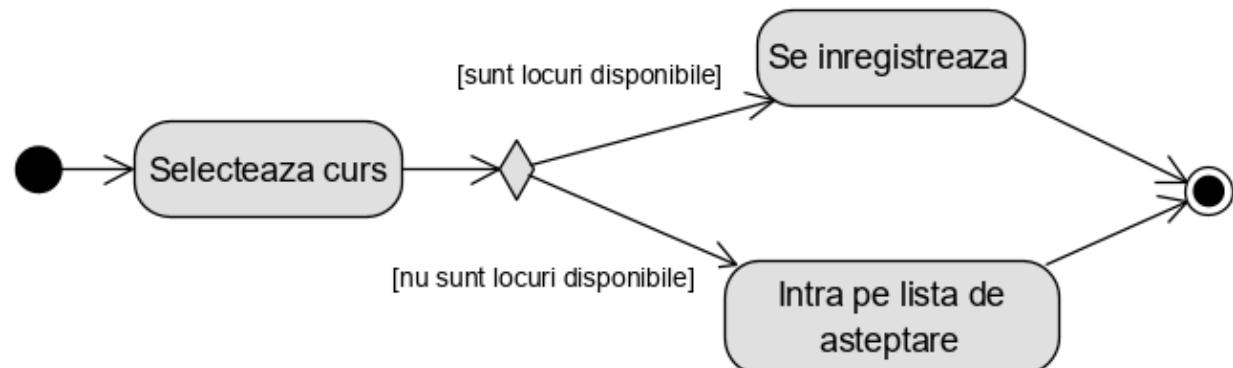
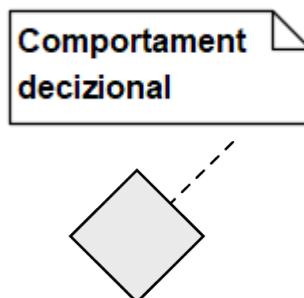
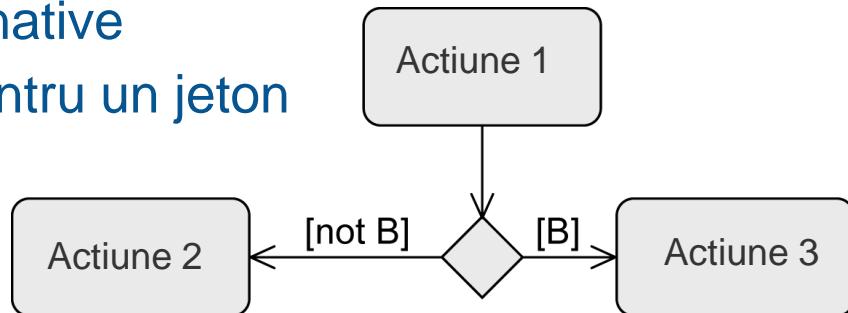
- Încheie execuția unei ramuri a activității
- Toate celelalte jetoane ale activității rămân neafectate

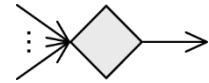




Căi alternative – Nod decizional (Decision)

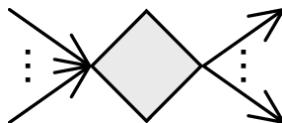
- Folosit pentru a defini ramuri alternative
- Reprezintă un punct de decizie pentru un jeton
- Arcele de ieșire au condiții
 - Sintaxa: [Expresie booleană]
 - Jetonul alege o **singură** cale
 - Condițiile trebuie să fie mutual exclusive
- Comportament decizional
 - Specifică ce comportament este necesar pentru evaluarea condițiilor



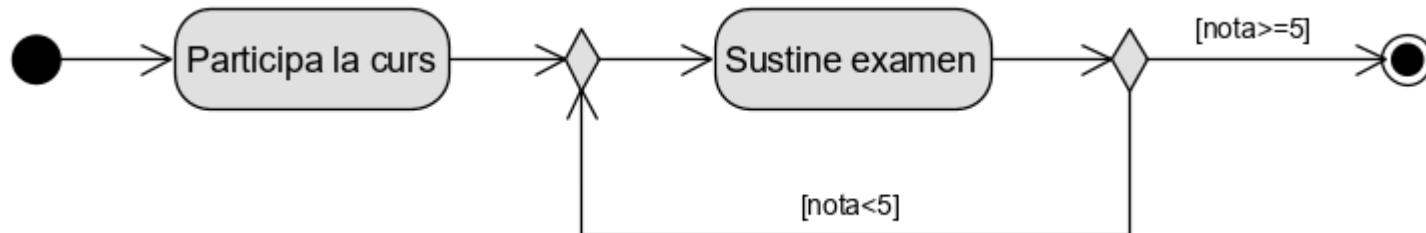


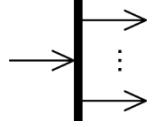
Căi alternative – Nod de îmbinare (Merge)

- Realizează convergența fluxurilor mutual exclusive
- Transmite jetonul următorului nod
- Există și versiunea combinată a nodului decizional și de îmbinare



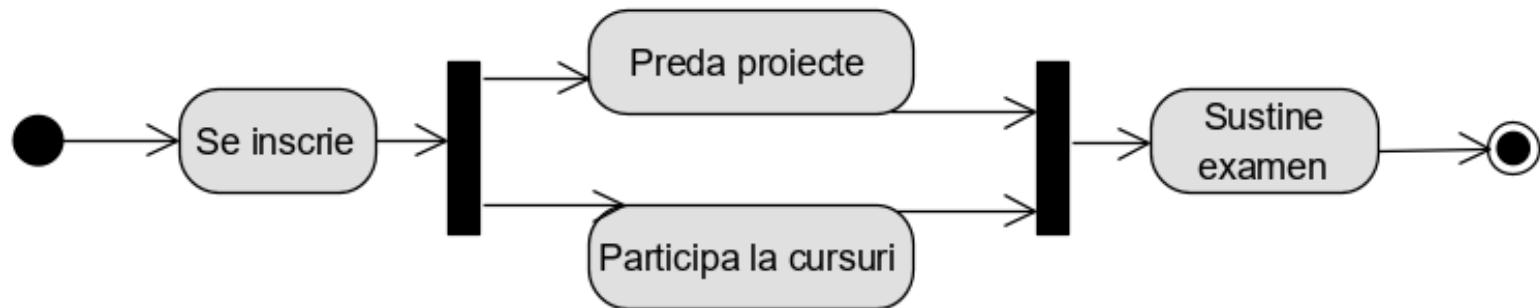
- Nodurile decizionale și de îmbinare pot fi folosite pentru a modela bucle:

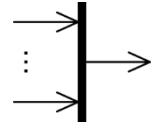




Căi concurente – nod de bifurcație (Fork)

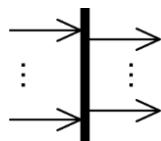
- Nod în care intră un singur flux și ieș mai multe
- Denotă începutul unor acțiuni paralele
- Se realizează copii ale jetonului pentru toate arcele de ieșire



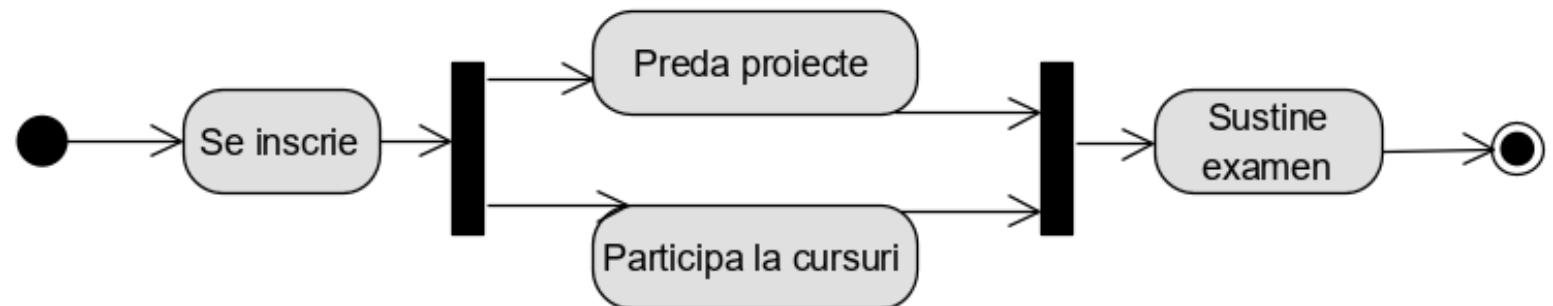


Căi concurente – Nod de sincronizare (Join)

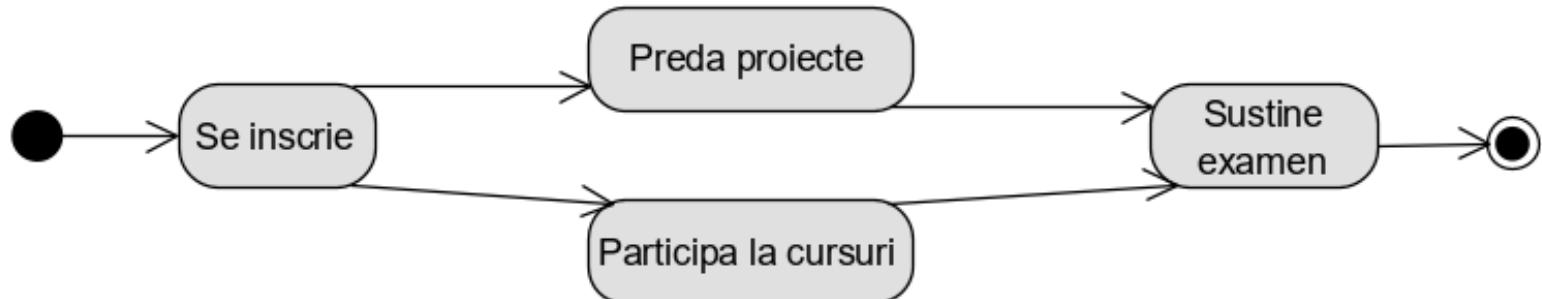
- Folosit pentru a sincroniza căi concurenre
- Procesarea jetonului
 - Așteaptă până când jetoanele sunt prezente în toate arcele de intrare
 - Sincronizează toate jetoanele de control într-unul singur și îl transmite mai departe
 - Transmite toate jetoanele obiect
- Se poate realiza combinarea nodurilor de bifurcație și sincronizare



Fluxuri de control echivalente

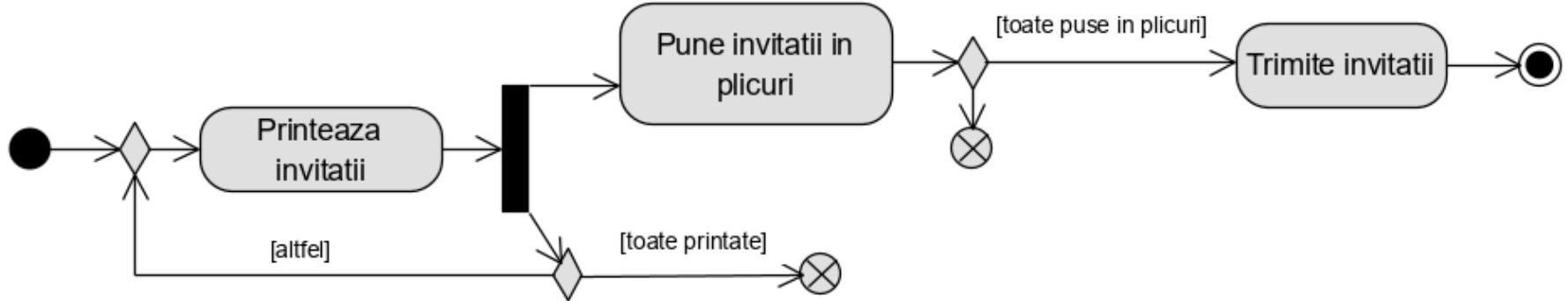


... este echivalent cu...

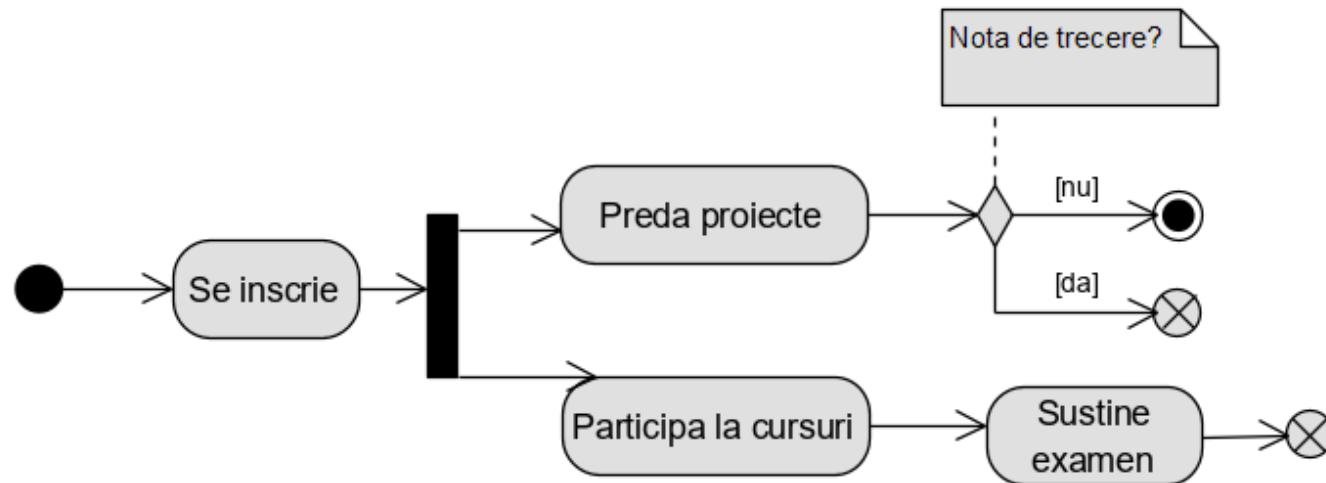


Exemplu: Crearea și trimiterea invitațiilor pentru un eveniment

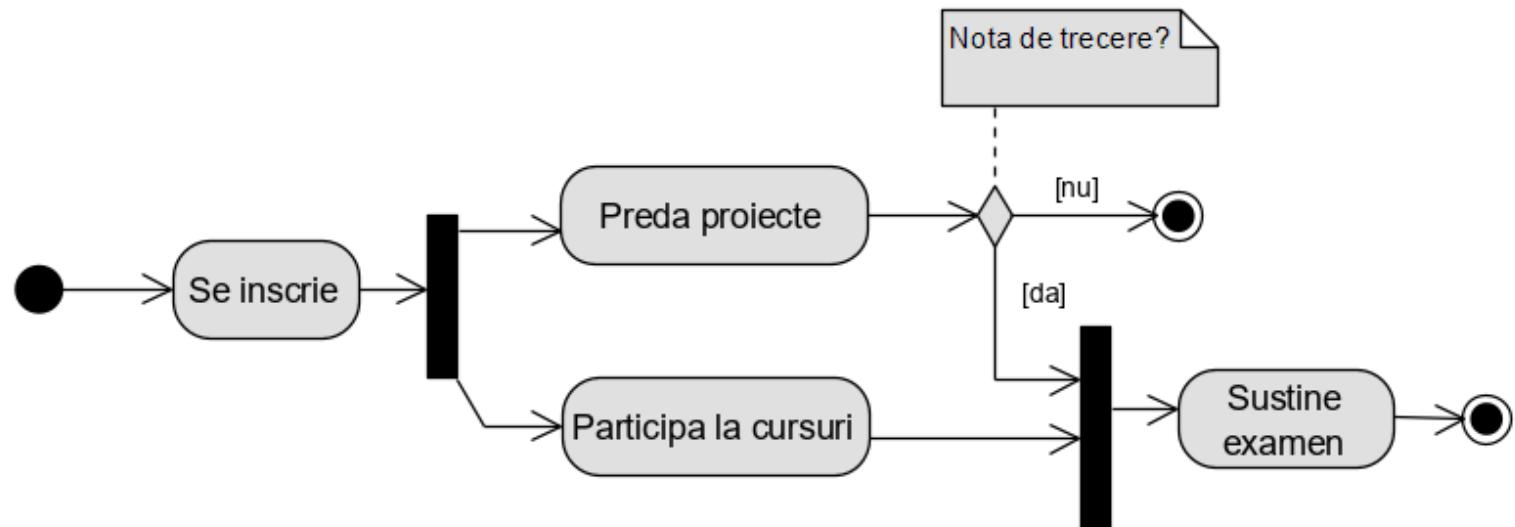
- În timp ce invitațiile sunt printate, cele deja printate sunt puse în plicuri.
- Când toate invitațiile sunt puse în plicuri, acestea sunt trimise destinatarilor.



Exemplu: Curs universitar



NU sunt echivalente... de ce?

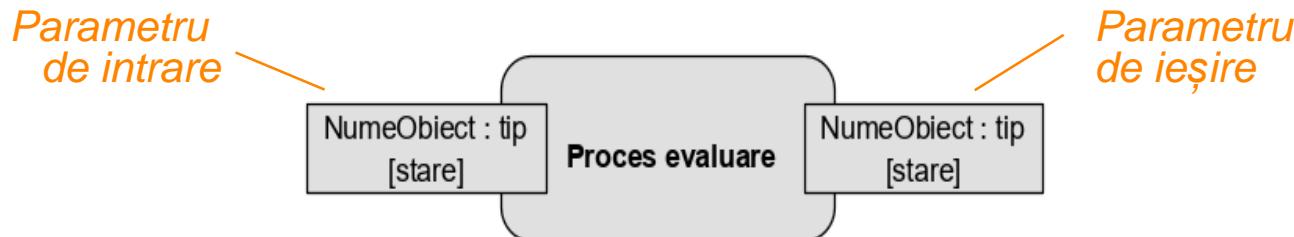


Nod de tip obiect

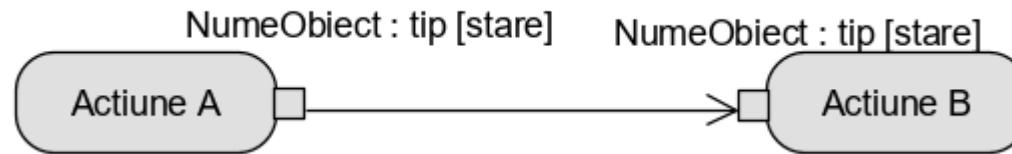
- Conține jetoane obiect
- Semnifică schimbul de date/obiecte
- Este sursa și destinația unui flux de obiect
- Poate include ca informație adițională: tipul, starea obiectului



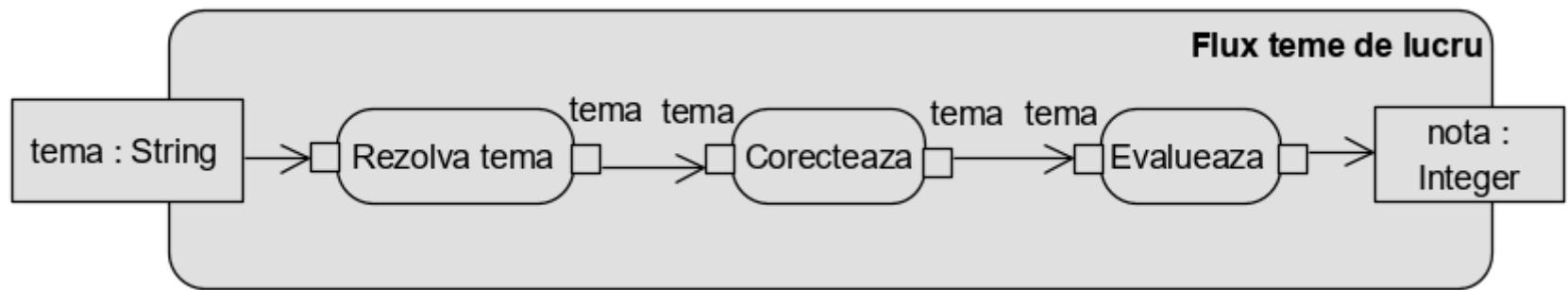
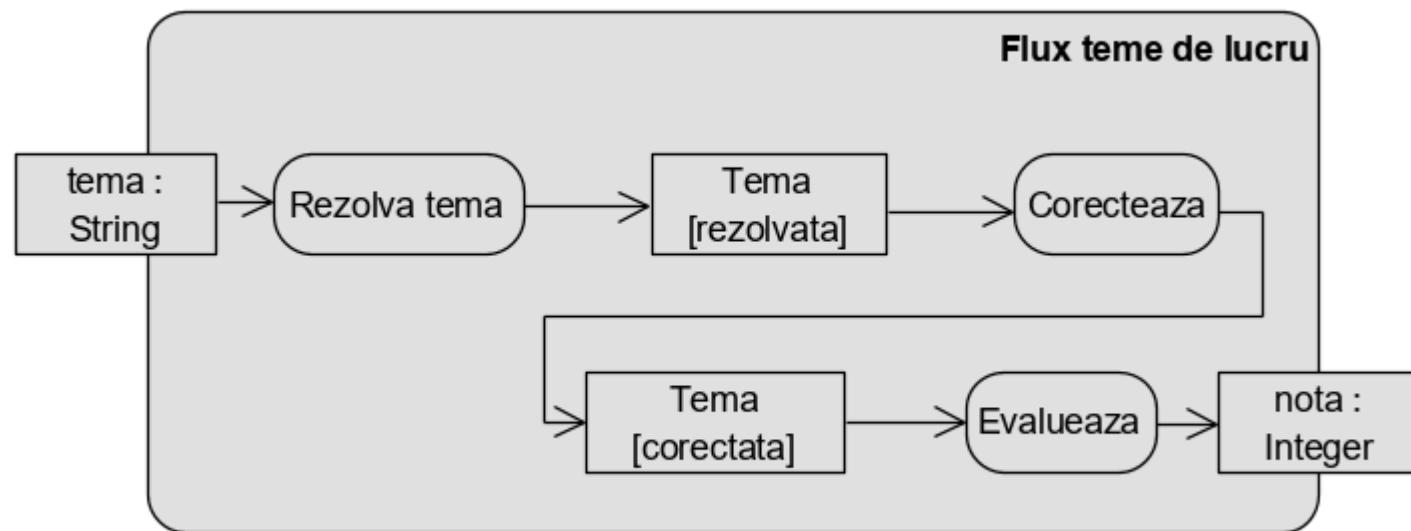
- Variante de notație: nod de tip obiect ca parametru
 - Pentru activități

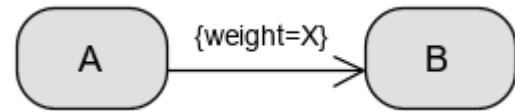


- Pentru acțiuni (“pins”)



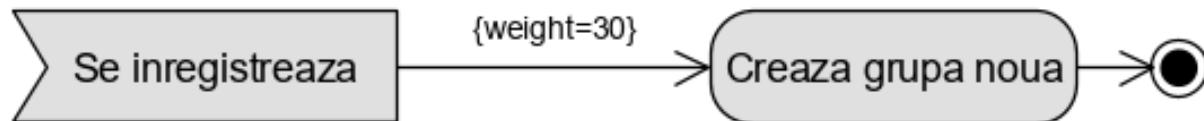
Exemple: Nod de tip obiect



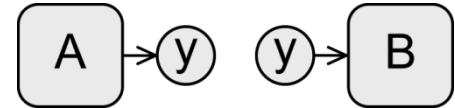


Sarcina unui arc (weight)

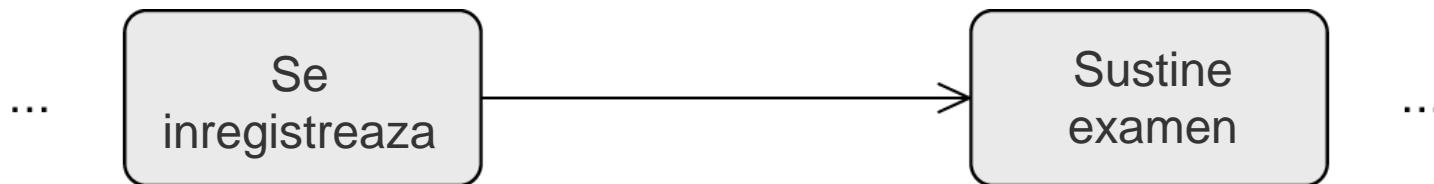
- Numărul minim de jetoane care trebuie să fie prezente pentru ca acțiunea să se execute
- Implicit: 1
- Toate jetoanele de la sursă sunt oferite simultan
- Se folosește cuvântul cheie “weight”



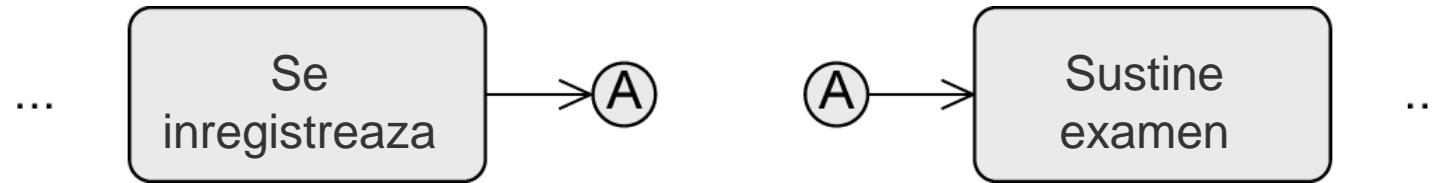
Conector



- Folosit atunci când două activități consecutive sunt poziționate în zone diferite ale diagramei
- Fără conector

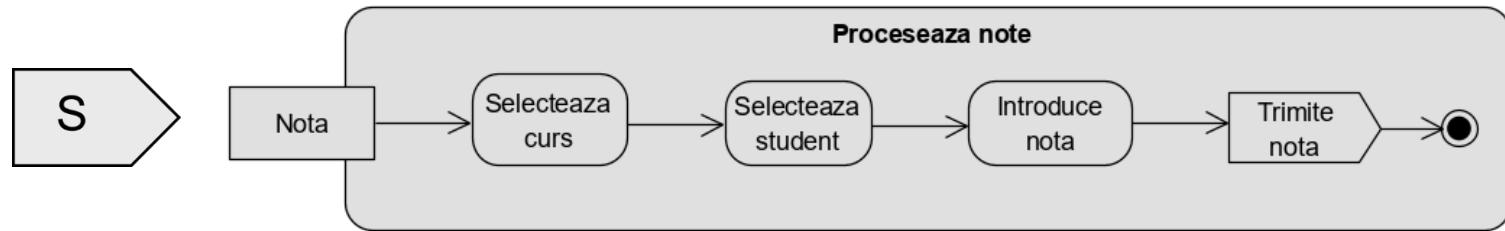


- Cu conector

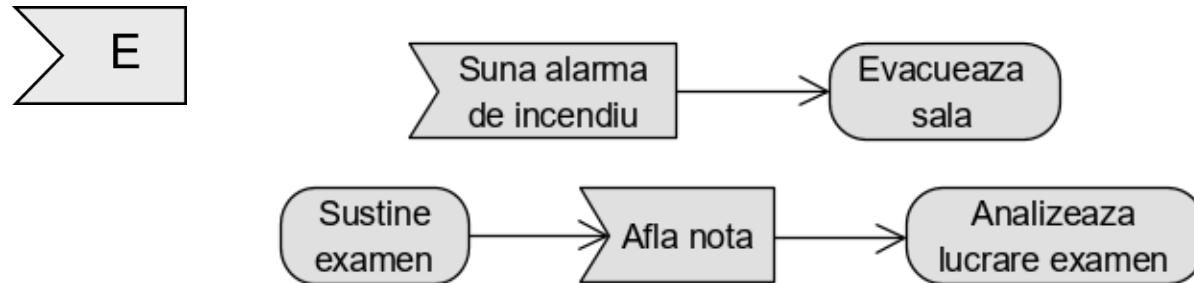


Acțiuni bazate pe evenimente

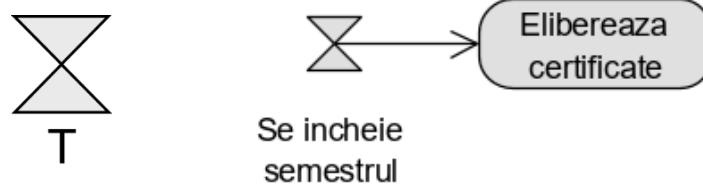
- Care trimit semnale
 - Acțiune care trimite semnal



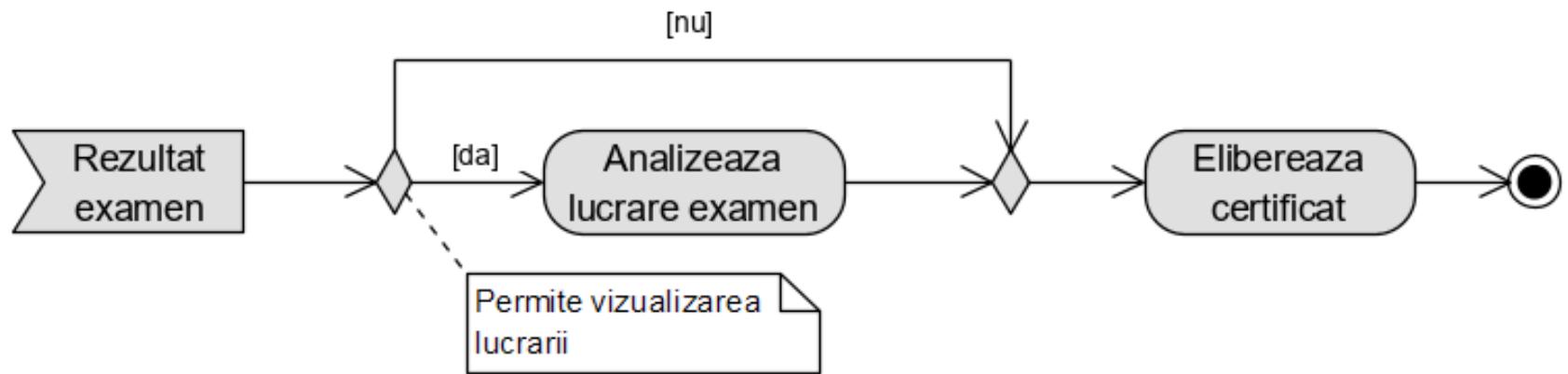
- Care acceptă evenimente
 - Acțiune care acceptă eveniment



- Acțiune care acceptă eveniment de tip timp

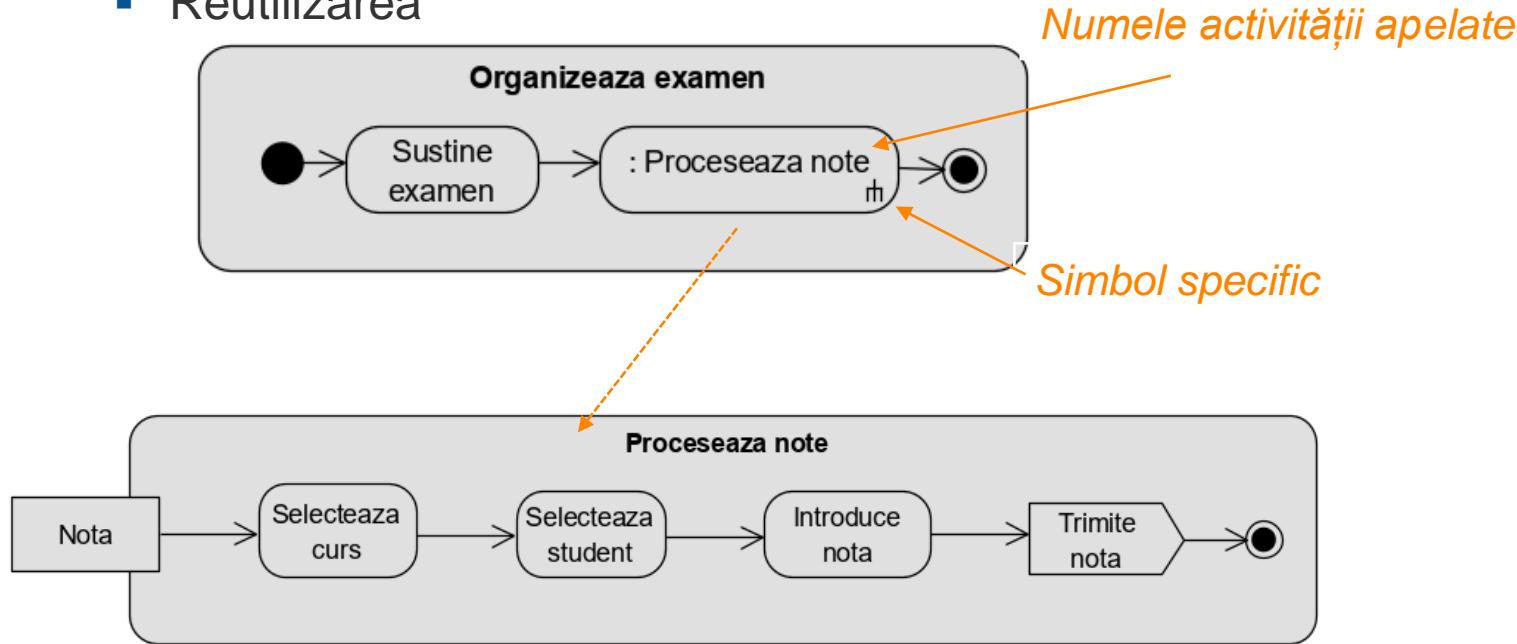


Exemplu: Acțiune care acceptă evenimente



Acțiune care apelează comportament

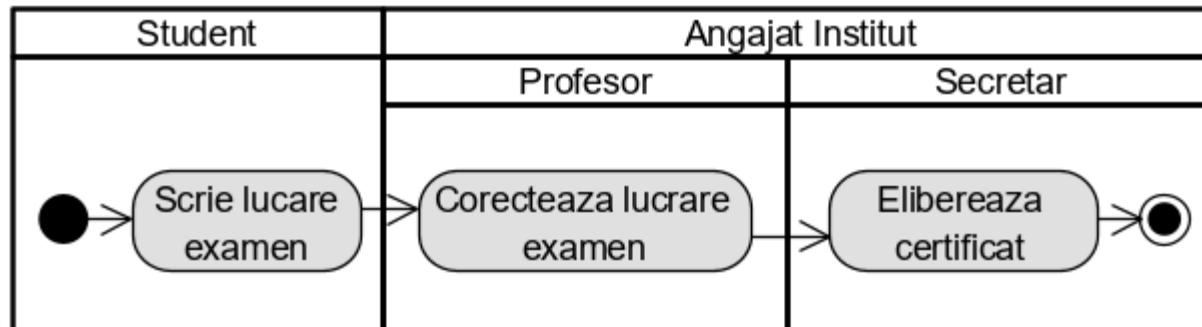
- Execuția unei acțiuni poate apela o activitate
- Conținutul activității apelate este modelat în altă parte
- Avantaje:
 - Modelele devin mai clare
 - Reutilizarea



A	B	A	
		B	

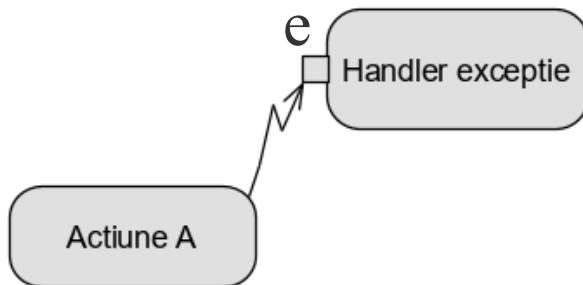
Partiții

- Sunt similare culoarelor dintr-un bazin de înot (“Swimlane”)
- Permit gruparea nodurilor și arcelor unei activități în funcție de responsabilități
- Responsabilitățile reflectă unitățile organizaționale sau rolurile
- Realizează o mai bună structurare a diagramei
- Nu schimbă semantica execuției
- Pot fi imbricate în funcție de necesități
- Exemple: partițiiile **Student** și **Angajat Institut** (cu subpartițiiile **Profesor** și **Secretar**)

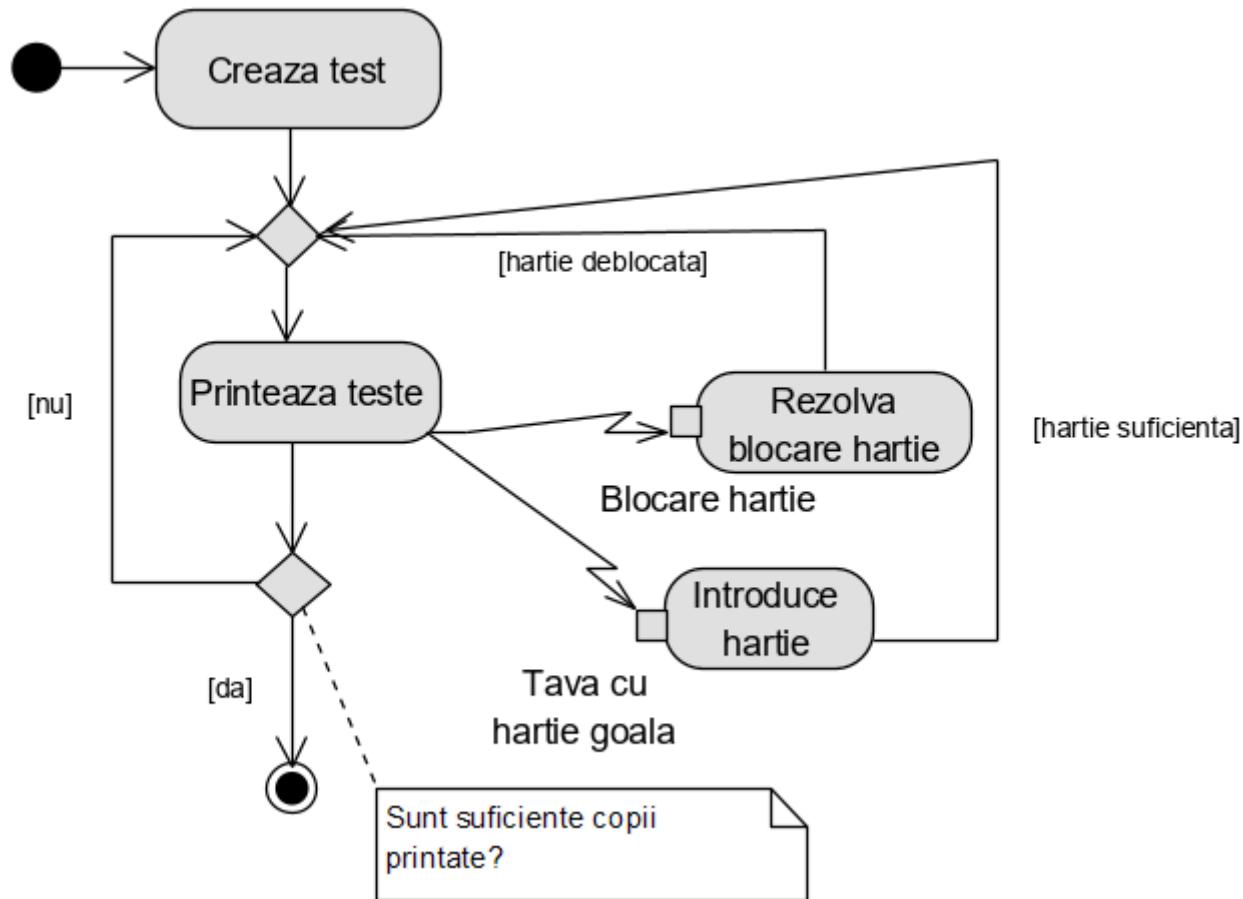


Tratarea excepțiilor (Exception Handler)

- Se referă la excepții predefinite
- Definește modul în care sistemul trebuie să reacționeze într-o anumită situație de eroare
- Handler-ul de excepții înlocuiește acțiunea în locul unde a apărut eroarea
 - Dacă are loc eroarea **e**, atunci:
 - Toate jetoanele din **Acțiunea A** sunt șterse
 - Se activează handler-ul de excepții
 - Se execută handler-ul de excepții în loc de **Acțiunea A**
 - Execuțiile următoare se realizează în mod normal
 - *Obs: În Visual Paradigm se creează un pin de intrare cu un parametru de tip Exception*

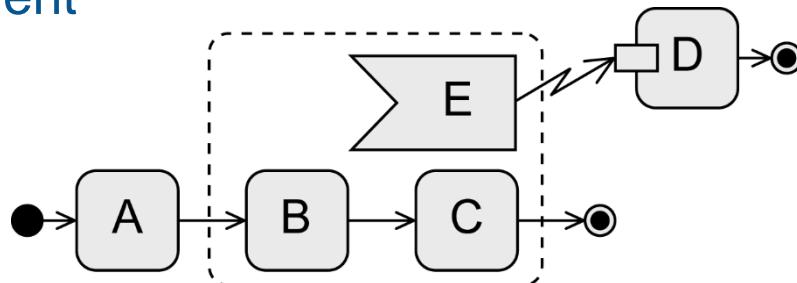


Example: Exception Handler



Tratarea excepțiilor – Regiune dintr-o activitate care poate fi întreruptă

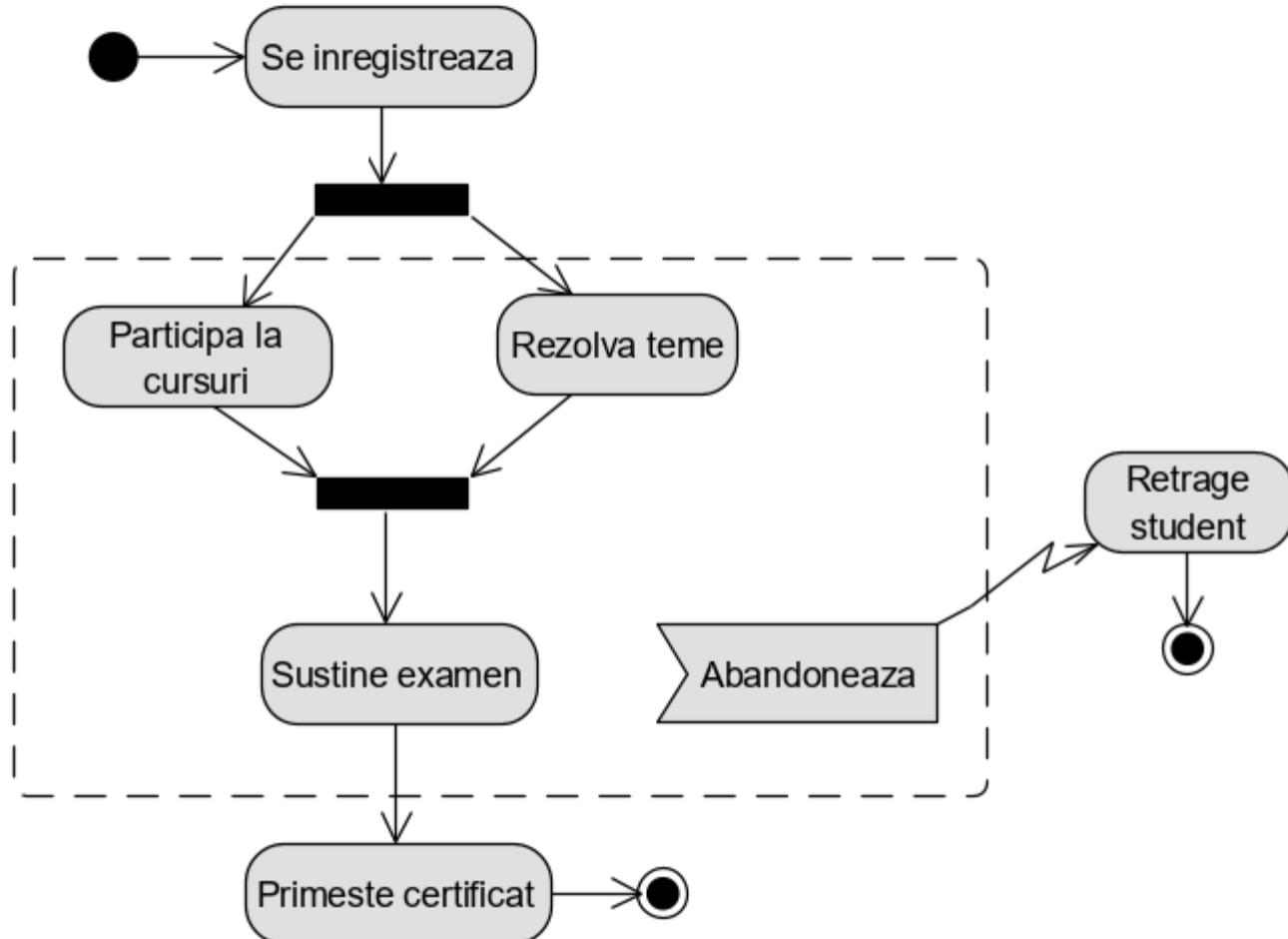
- Definește un grup de acțiuni a căror execuție trebuie terminată imediat dacă are loc un anumit eveniment. În acest caz, se execută un alt comportament



- Dacă **E** are loc în timp ce **B** sau **C** sunt executate, atunci:
 - Se activează mecanismul de tratare a excepțiilor
 - Toate jetoanele de control din interiorul dreptunghiului sunt șterse
 - **D** este activată și executată
- Se întrerupe fluxul normal de execuție



Exemplu: Regiune dintr-o activitate care poate fi întreruptă

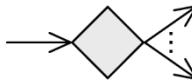
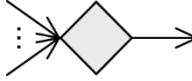
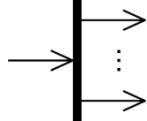
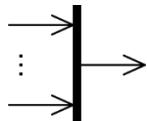


Notări -1

Nume	Notăie	Descriere
Nod acțiune	Actiune	Reprezintă o acțiune (este atomică)
Nod activitate	Activitate	Reprezintă o activitate (mai poate fi descompusă)
Nod inițial	●	Începutul execuției unei activități
Nod final al activității	○	Sfârșitul tuturor fluxurilor de execuție ale unei activități

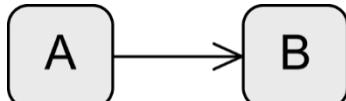
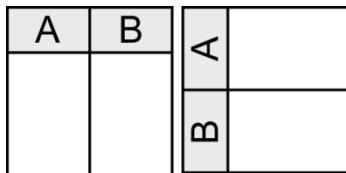


Notății -2

Nume	Notăție	Descriere
Nod decizional		Folosită pentru divergența fluxurilor mutual exclusive
Nod de îmbinare		Folosită pentru convergența fluxurilor mutual exclusive
Nod de bifurcație		Folosită pentru divergența fluxurilor concurente
Nod de sincronizare		Folosită pentru convergența fluxurilor concurente



Notății -3

Nume	Notăție	Descriere
Nod final al fluxului	\otimes	Sfârșitul execuției unui flux al activității
Arc		Conectează nodurile unei activități
Acțiune care apelează comportament		Acțiune A apelează o activitate cu același nume
Partiție		Grupează nodurile și arcele din cadrul unei activități

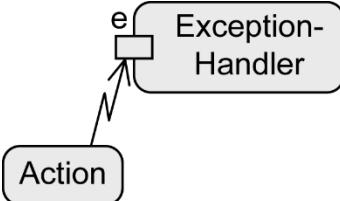
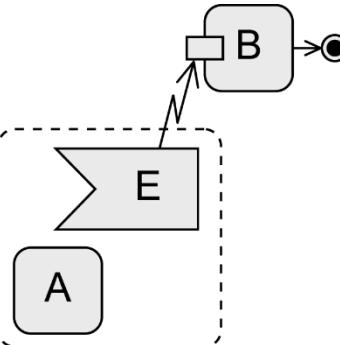


Notății -4

Nume	Notăție	Descriere
Acțiune trimite semnal		Transmite un semnal către un receptor
Acțiune care acceptă evenimente	or	Așteaptă un eveniment E sau un eveniment de tip T
Nod de tip obiect		Conțin date sau obiecte
Parametri pentru activități Parametri pentru acțiuni (pins)	 	Conțin date și obiecte ca parametri de intrare sau ieșire



Notății -5

Nume	Notăție	Descriere
Handler de excepții		Handler-ul de excepții se execută în loc de acțiune în cazul apariției evenimentului e
Regiune care poate fi întreruptă		Fluxul continuă pe o cale diferită atunci când este detectat evenimentul E



Mecanisme de extensie

Extinderea limbajului UML

Cuprins

- Introducere
- Note
- Stereotipuri
- Constrângeri
- Valori etichetate



Introducere

- UML are o largă **aplicabilitate** în proiecte de diferite tipuri și dimensiuni.
- În plus față de diagramele din standard, UML a fost creat având capacitatea de a se **extinde**.
- Această extensibilitate a limbajului îl face **versatil**.



Introducere

- Mecanismele de extensie UML includ:
 - **Note** - pot fi folosite pentru a adăuga informații suplimentare în diagramele UML, oferind explicații care nu pot fi surprinse direct de către notațiile din diagrame.
 - **Stereotipuri** - sunt un mecanism folosit pentru a clasifica orice element de modelare în UML și facilitează înțelegerea numeroaselor diagrame și modele.
 - **Constrângeri** - sunt restricții aplicate modelelor care ajută la îmbunătățirea calității modelului.
 - **Valori etichetate** - sunt identificatori aplicați modelelor, având rolul de a le eticheta cu proprietăți predefinite.

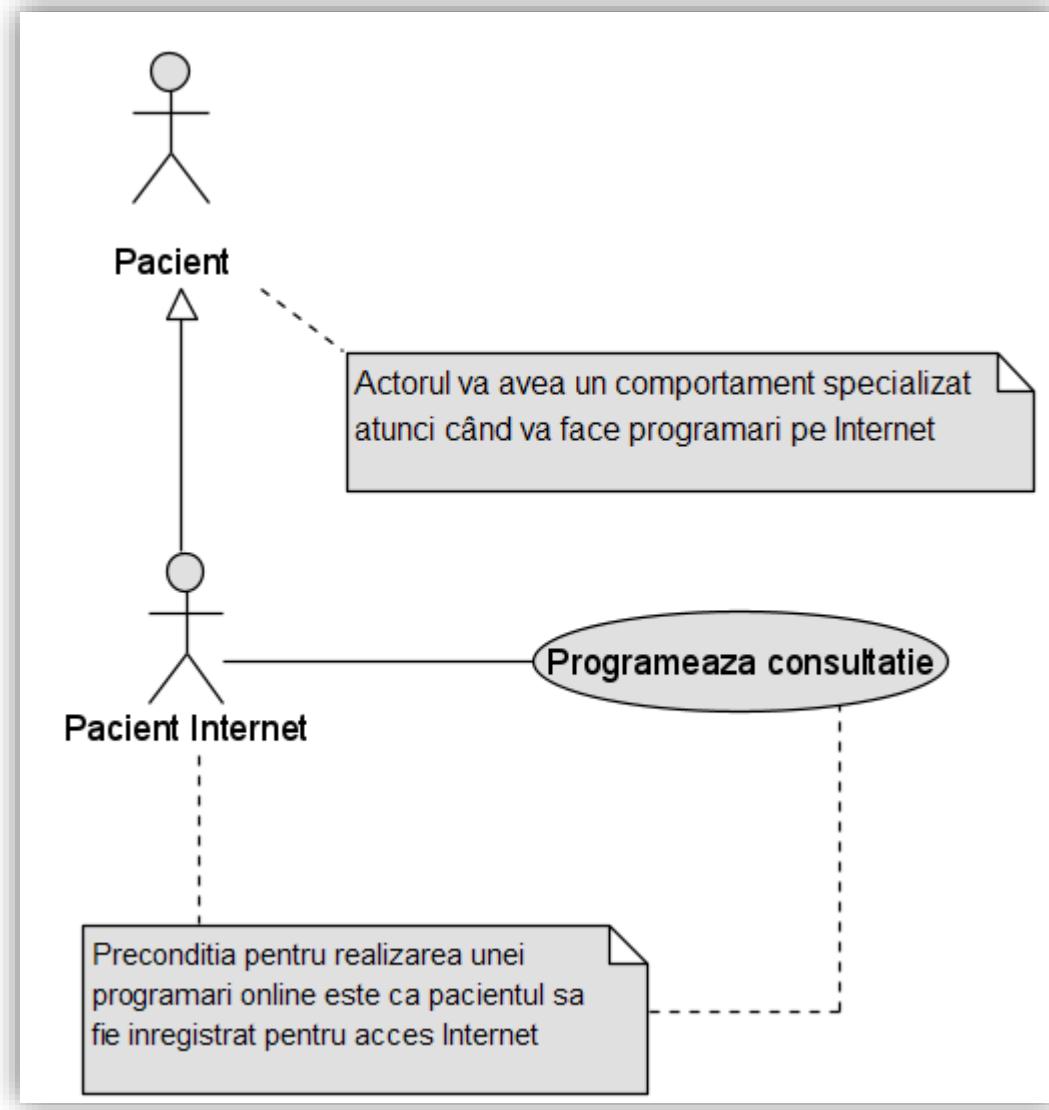


Note

- Sunt un bun mijloc descriptiv de **clarificare** a diagramelor UML.
- Oferă **explicații suplimentare** despre dependențele dintre elementele unei diagrame UML.
- Mecanismul este foarte util, în special în diagramele statice structurale, unde este **dificil** să se arate dependențe.
- Sunt reprezentate de un **dreptunghi cu un colț îndoit** și sunt legate de orice alte elemente din diagramă.
- Notele dintr-o diagramă UML sunt similare **comentariilor** dintr-un cod **sursă** bine documentat. Ajută la înțelegerea semnificațiilor mai profunde și implicate din spatele diagramei.
- Pot **contine**: comentarii textuale, simboluri grafice, descrieri detaliate, linkuri către pagini web și referințe la alte documente.
- Nu au **niciun efect** asupra **semanticii** unui model (nu modifică sensul modelului).

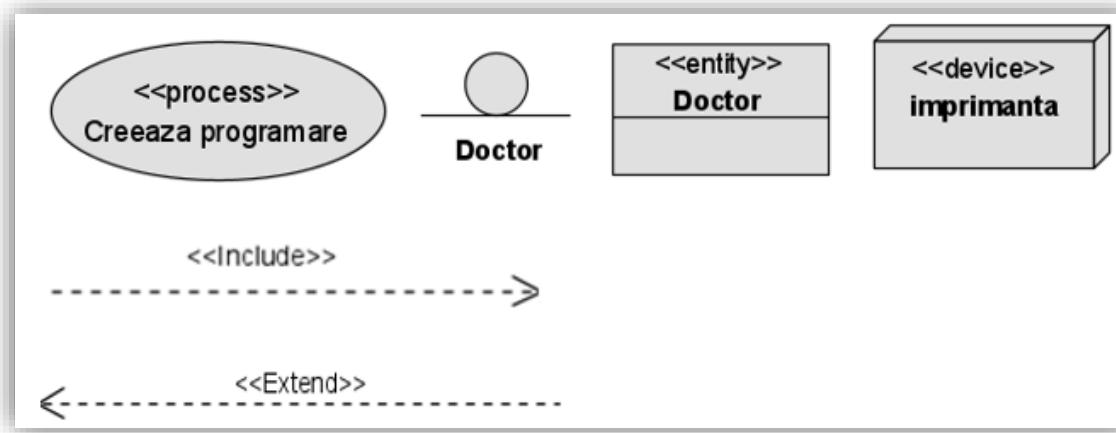


Note

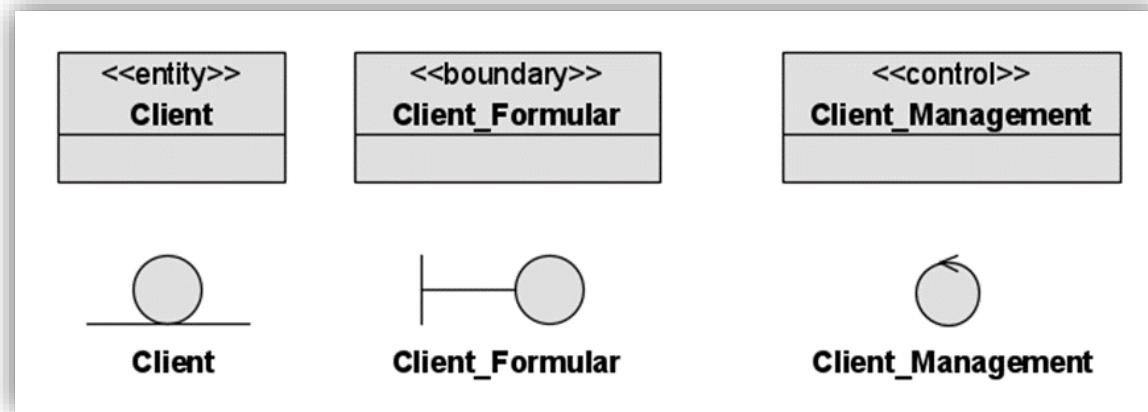


Stereotipuri - exemple

<< >>



- Stereotipuri comune care pot apărea în diagramele de clase:
 - **Clasă Entity** – modelează obiecte din domeniul afacerii
 - **Clasă Boundary** – modelează interfețe
 - **Clase Control** - modelează logica aplicației



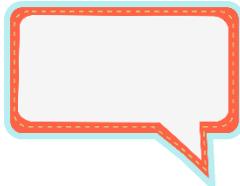
Stereotipuri

- Stereotipurile sunt folosite pentru a **clasifica** aproape orice element din UML.
- Deși sunt, în mare parte, **optionale**, ele ajută la **înțelegerea** diagramelor, oferind o modalitate de **grupare la nivel înalt** a elementelor din diagramă.
- Prin intermediul stereotipurilor, se pot folosi elementele de bază, dar cu **proprietăți speciale**, ca semantică și notație.
- Strict din punct de vedere **semantic**, este echivalent cu realizarea unor **noi elemente** de modelare.
- O astfel de grupare ajută și la o mai bună **comunicare** a scopului elementului UML.
- Utilizatorii pot crea **propriile** stereotipuri.



Stereotipuri obligatorii

- Există și stereotipuri **obligatorii**, predefinite în cadrul limbajului.
- În alte cazuri, stereotipurile pentru relații sunt **implicite** prin simbolul sau pictograma folosită pentru a reprezenta relația respectivă.
- Stereotipurile pentru relațiile la nivelul claselor sunt reprezentate **doar** prin **pictograme**.
- De exemplu, stereotipul „**moștenire**” într-o relație clasă-clasă **nu trebuie etichetat**. Vârful săgeții care reprezintă relația de moștenire între două clase face ca relația să fie unică și clară chiar și fără eticheta de stereotip.



Dați exemple de astfel de stereotipuri pentru relațiile dintre cazuri de utilizare și relațiile dintre clase!



Stereotipuri pentru clase

■ Clasa Entity (Entitate) <<entity>>



- Reprezintă stereotipul care descrie o **entitate din domeniul analizat**, de exemplu, un client sau o factură.
- Clasele entități alcătuiesc **principalul set de clase** al unui sistem și, prin urmare, sunt primele clase care trebuie descoperite în timpul etapei de analiză.
- De obicei, o diagramă de clasă la nivel de analiză este aproape întotdeauna **formată doar din clase entitate**. În etapa de proiectare se pot adăuga și **alte categorii** de clase stereotip.

■ Clasa Boundary (Interfață) <<boundary>>



- Indică faptul că acea clasă este o interfață, adică o **graniță** între sistem și utilizator.
- Clasele interfață pot descrie diferite elemente precum **formulare** sau **pagini web** sau **interfețe** la sisteme și dispozitive externe (de exemplu, o interfață electronică pentru schimb de date între două bănci).



Stereotipuri pentru clase



■ Clasa Control <<control>>

- Este un stereotip al unei clase care este folosit pentru a lega clasele entități la clasele interfață, în scopul reducerii cuplării clasei.
- Clasele control sunt clase temporare și gestionează un set de interacțiuni, secvențele și ordinea temporală într-un sistem și apoi dispar.
- Ajută la decuplarea claselor entitate de clasele interfață.

■ Clasa Utility (Utilitară) <<utility>>

- Sunt clase oferite de mediul de dezvoltare. Sunt, de obicei clase care oferă funcții utilitare generice și aplicabile în multe situații.
- De exemplu, clasele care oferă funcții pentru lucrul cu date de tip dată/timp sau conversii ale monedelor.

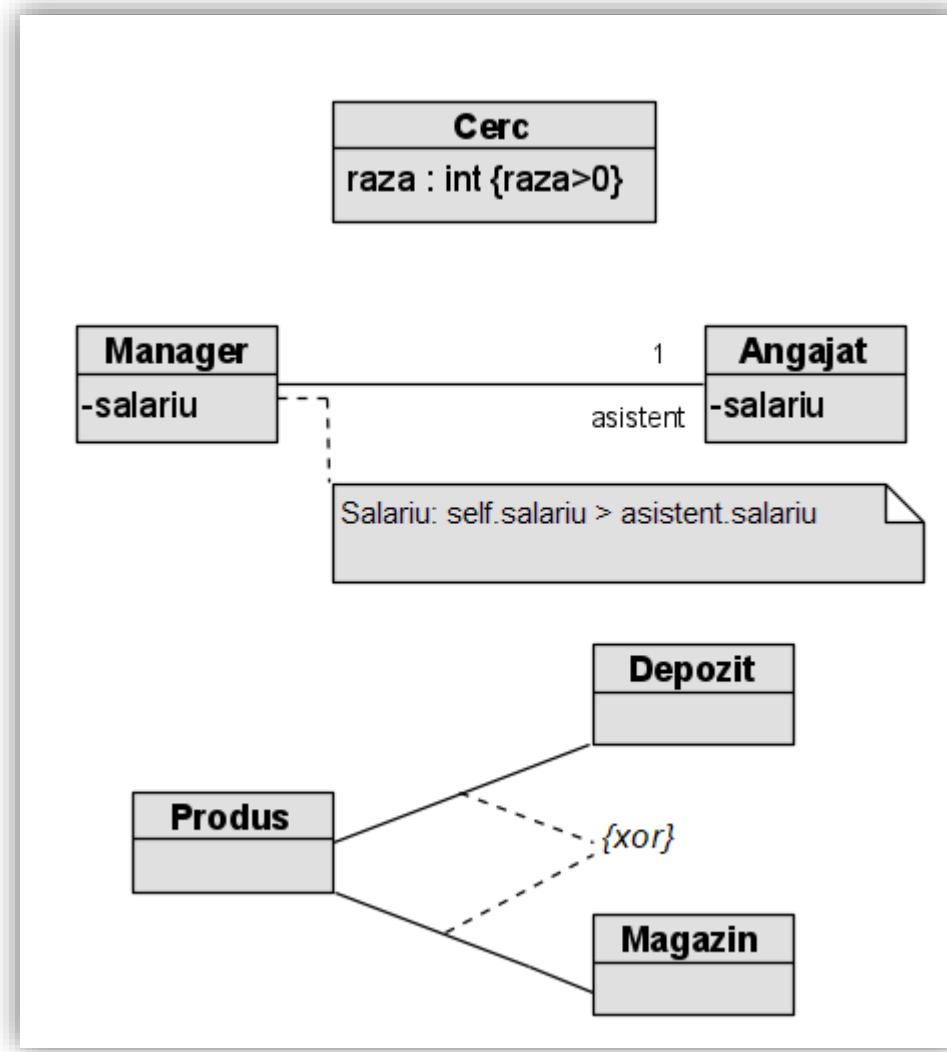
■ Clasele abstracte

- Sunt o categorie importantă de clase în orientarea obiect, chiar dacă nu au asociat un stereotip.
- Numele lor apare scris cu italic.



Constrângeri - exemple

{ }



Constrângeri

- O constrângere este o **regulă suplimentară** asociată unei diagrame UML pentru a da o semnificație specială unui **element** din diagramă, **relației** dintre mai multe elemente sau **întregii** diagrame.
- Este o extensie a semanticii unui **element UML**, permitând adăugarea de **reguli noi** sau **modificarea** celor existente.
- O constrângere specifică **condițiile** care trebuie respectate pentru ca modelul să fie corect construit.
- Poate fi utilizată pentru a **extinde notația de bază** a unui element din model sau pentru a vizualiza părți din specificațiile unui element care nu au reprezentări grafice.



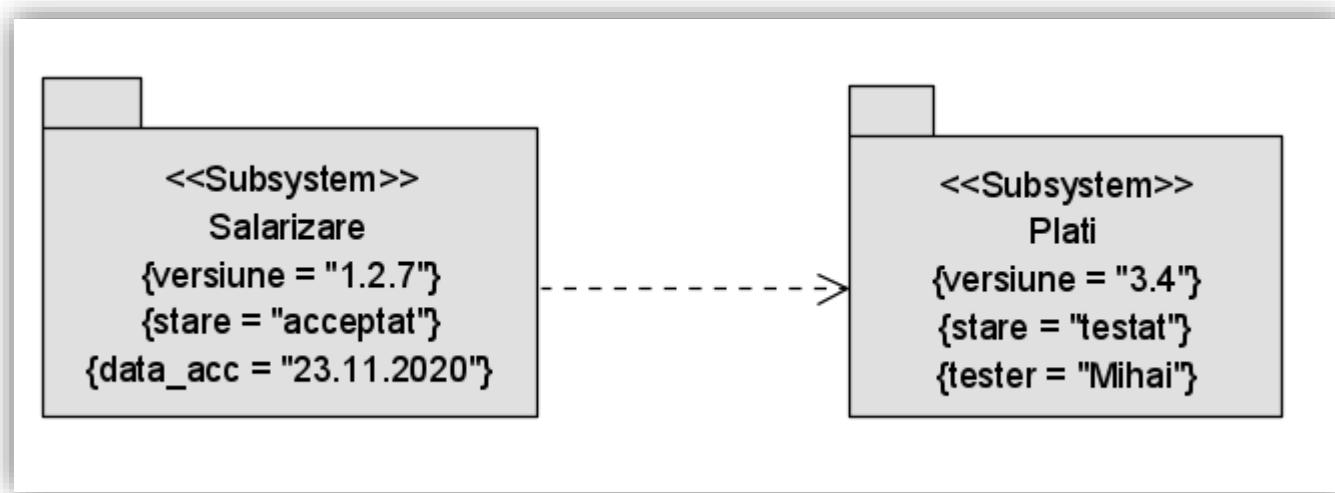
Valori etichetate

- Valorile etichetate extind proprietățile unui element de modelare, permitând crearea de informații noi în specificațiile aceluiași element.
- Sunt descrise prin intermediul perechilor de cuvinte cheie-valoare ale elementelor din model, în care cuvintele cheie sunt attribute.
- Pot fi definite pentru elementele de model existente sau pentru stereotipurile individuale, astfel încât întregul stereotip să aibă asociată aceeași valoare etichetată.
- O valoare etichetată nu este egală cu un atribut al clasei.
- Poate fi asimilată metadatelor, deoarece se aplică elementului în sine și nu instanțelor sale.
- Grafic, este redată ca un sir inclus între acolade, care este plasat sub numele unui alt element de model. Sirul constă dintr-un nume (eticheta), un separator (simbolul =) și o valoare (a etichetei).



Valori etichetate

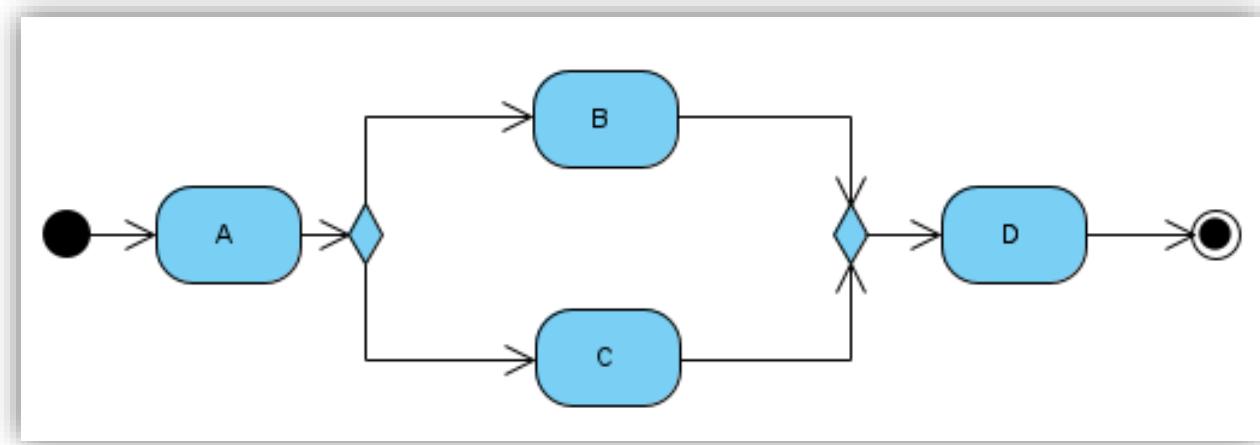
- Utilizări frecvente:
 - specificarea proprietăților relevante pentru **generarea** codului sursă
 - gestiunea **configurațiilor**
- Exemple: se poate specifica *limbajul de programare în care va fi implementată o anumită clasă sau poate fi folosit pentru a indica autorul și versiunea unei componente.*





Exerciții -1

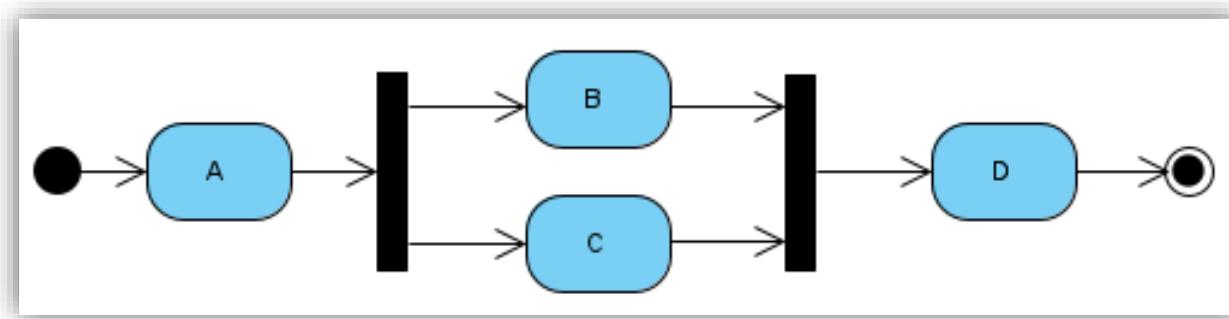
Identificați posibilele căi de execuție pentru diagrama de mai jos.





Exerciții -2

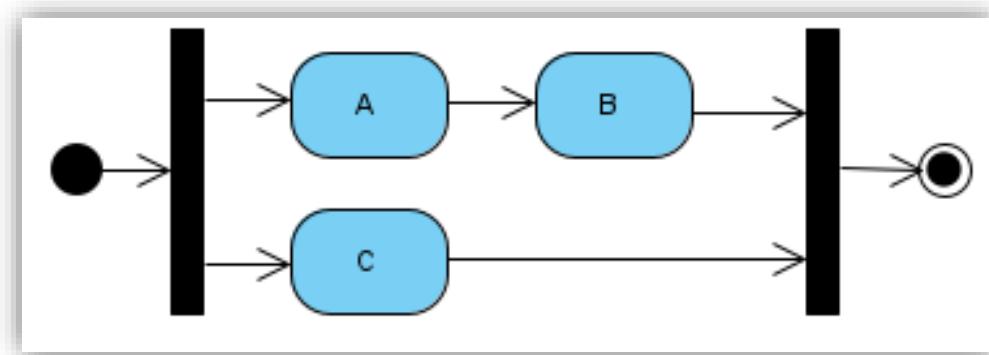
Identificați posibilele căi de execuție pentru diagrama de mai jos.





Exerciții -3

Identificați posibilele căi de execuție pentru diagrama de mai jos.

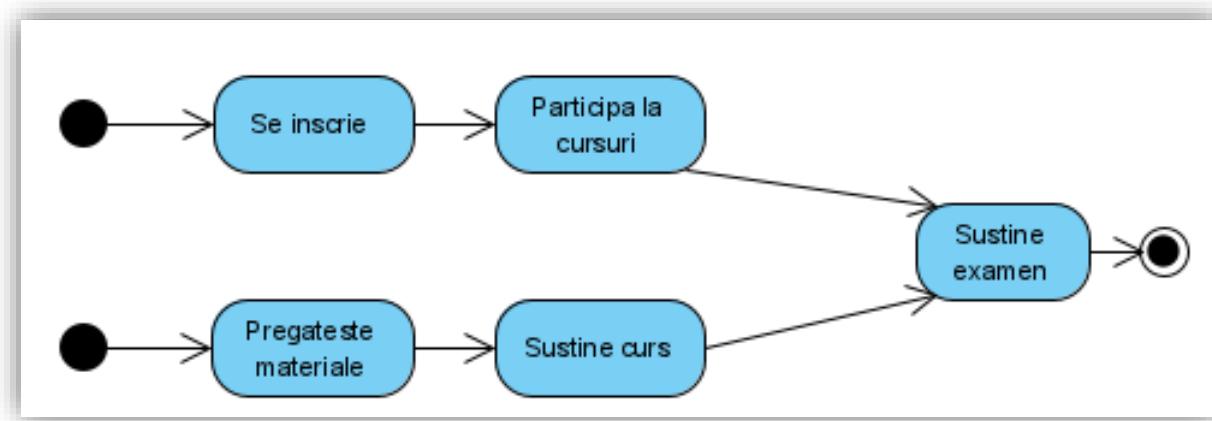




Exerciții -4

Identificați care dintre elemente de mai jos apar în această diagramă:

- Evenimente inițiale multiple
- Evenimente finale multiple
- Acțiuni concurente
- Acțiuni mutual exclusive

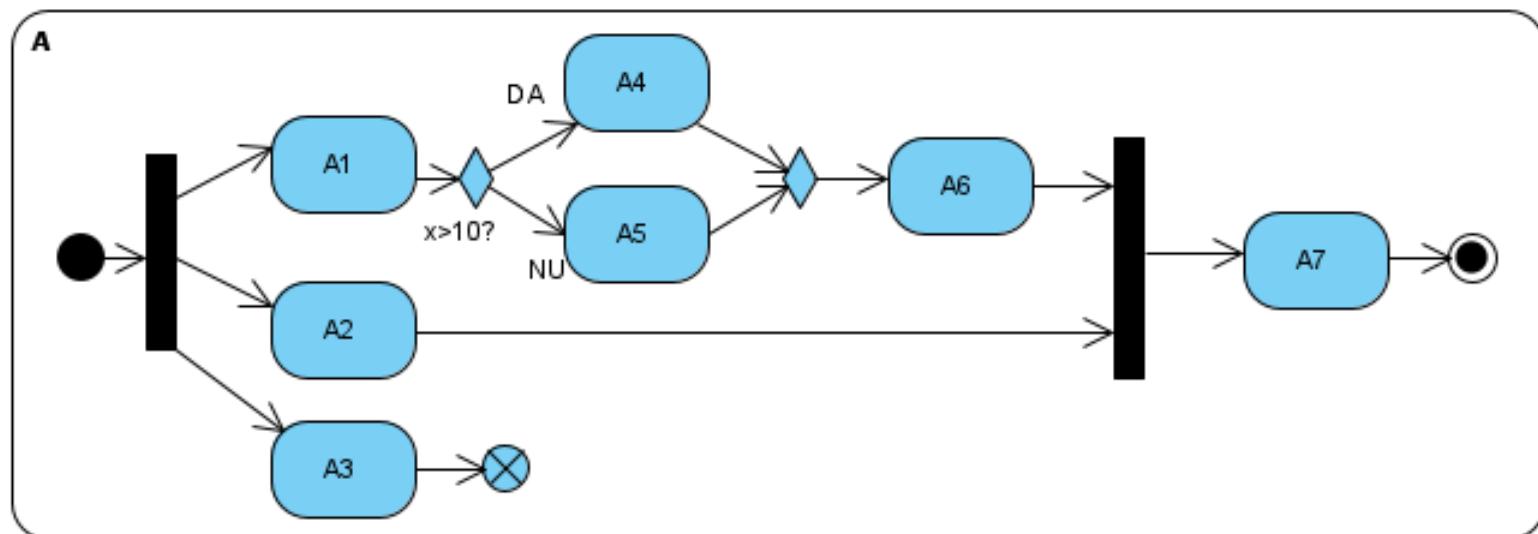




Exerciții -5

Analizați componentele activității A.

- Ce acțiuni trebuie finalizate pentru a se putea executa acțiunea A7?
- Ce se întâmplă dacă activitatea A7 este finalizată, iar A3 este încă în execuție?



ACADEMIA DE STUDII ECONOMICE BUCUREŞTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

PROIECTAREA SISTEMELOR INFORMATICE

- CURS 6 -

BUCUREŞTI
2020-2021

Diagramele de mașini cu stări

*Modelarea
dinamică*

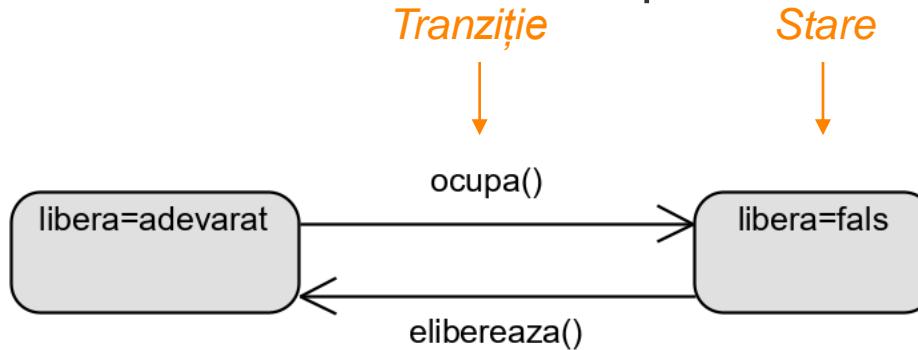
Cuprins

- Introducere
- Stări
- Tranziții
- Tipuri de evenimente
- Tipuri de stări
- Puncte de intrare și ieșire

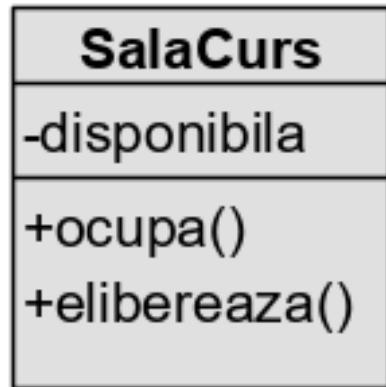
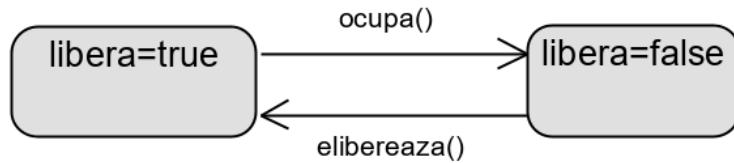


Introducere

- Fiecare obiect trece printr-un număr finit de stări diferite pe parcursul vieții sale
- Diagrama de mașini cu stări este utilizată:
 - Pentru modelarea stărilor posibile ale unui sistem sau obiect
 - Pentru a arăta cum au loc tranzitările dintre stări ca urmare a unor evenimente
 - Pentru a arăta ce comportament prezintă sistemul sau obiectul în fiecare stare
- Exemplu: descrierea la nivel înalt a comportamentului unei săli de cursuri



Exemplu: SalaCurs cu detalii



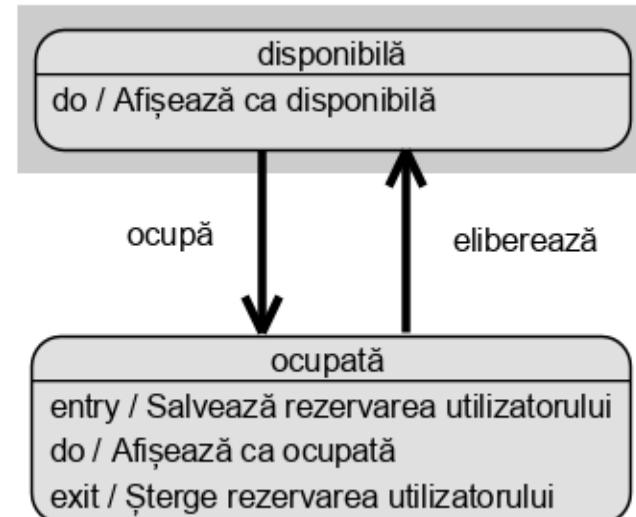
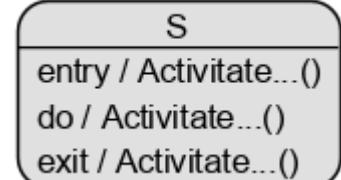
```
class SalaCurs {
    private boolean libera;

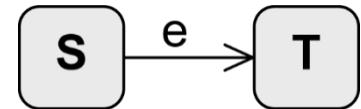
    public void ocupa() {
        libera=false;
    }
    public void elibereaza() {
        libera=true;
    }
}
```

S

Stare

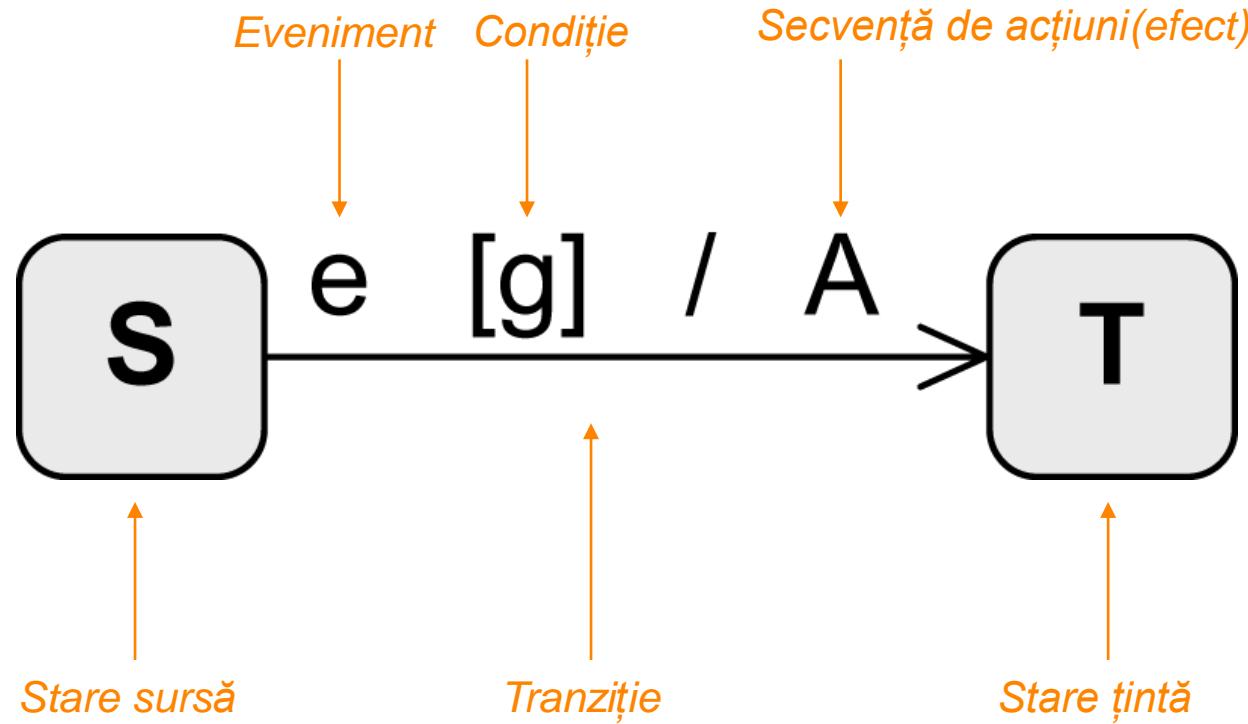
- Stări = noduri ale mașinii de stare
- Când o stare este activă
 - Obiectul se află în starea respectivă
 - Toate activitățile interne specificate în această stare pot fi executate
 - O activitate poate consta din mai multe acțiuni
- entry / Activity(...)
 - Executată când obiectul intră în stare
- exit / Activity(...)
 - Executată când obiectuliese din stare
- do / Activity(...)
 - Executată cât timp obiectul rămâne în această stare



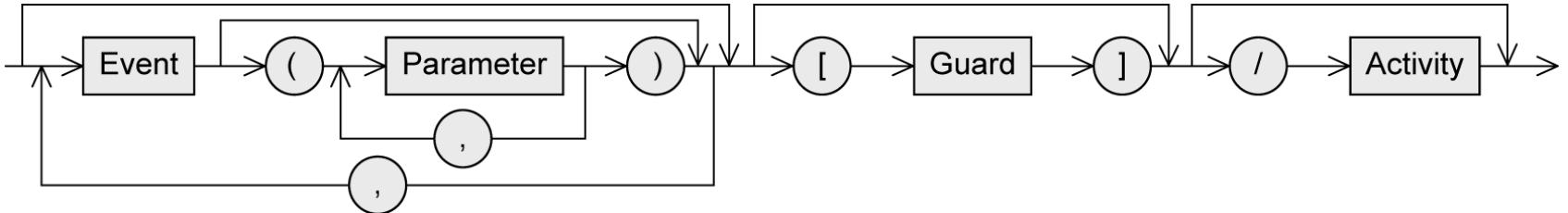


Tranzitie

- Schimbare de la o stare la alta



Tranzitie – Sintaxă



- Eveniment (trigger)
 - Stimul exogen
 - Poate declanșa o tranzitie a unei stări
- Condiție (guard)
 - Expresie booleană
 - Dacă evenimentul are loc, condiția este verificată
 - Arunci când condiția este adevărată
 - Toate activitățile din starea actuală sunt încheiate
 - Orice activitate de ieșire relevantă este executată
 - Tranzitia are loc
 - Arunci când condiția este falsă
 - Nu are loc nici o tranzitie, evenimentul este anulat
- Activitate (efect)
 - Secvența acțiunilor execute în timpul unei tranzitii de stare

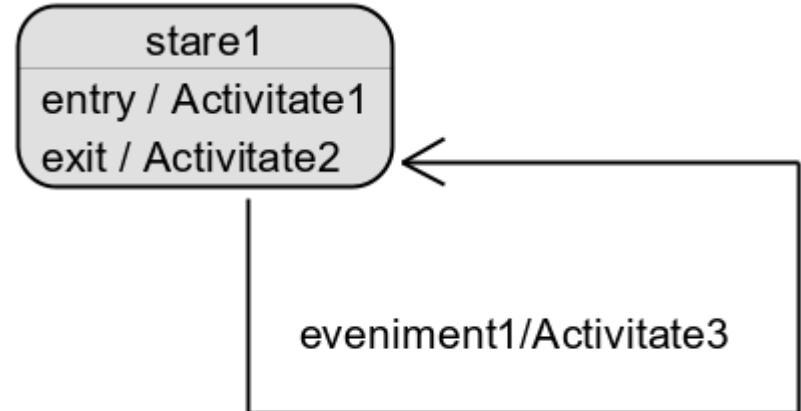


Tranzitie – Tipuri (1/2)

Tranzitie internă



Tranzitie externă



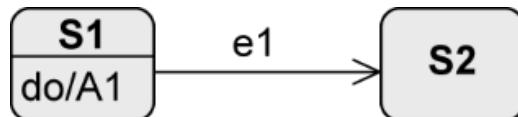
- Dacă are loc **eveniment1**
 - Obiectul rămâne în **stare1**
 - **Activitatea3** este executată

- Dacă are loc **eveniment1**
 - Obiectul ieșe din **starea1** și **Activitatea2** este executată
 - **Activitatea3** este executată
 - Obiectul intră în **starea1** și **Activitatea1** este executată

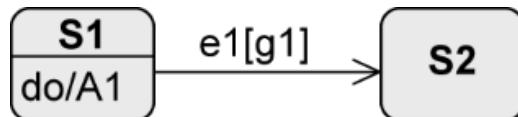


Tranzitie – Tipuri (2/2)

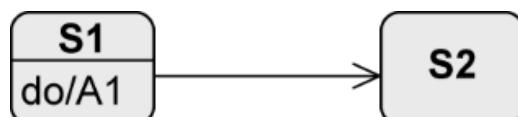
- Când au loc următoarele tranzitii?



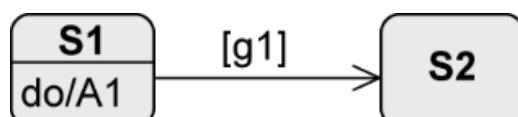
Dacă are loc **e1**, **A1** este anulată și obiectul își schimbă starea în **S2**



Dacă are loc **e1** și **g1** este evaluată ca fiind adevărată, **A1** este anulată și obiectul își schimbă starea în **S2**



Imediat ce execuția lui **A1** este terminată, se generează un eveniment de finalizare care inițiază tranzitia la **S2**

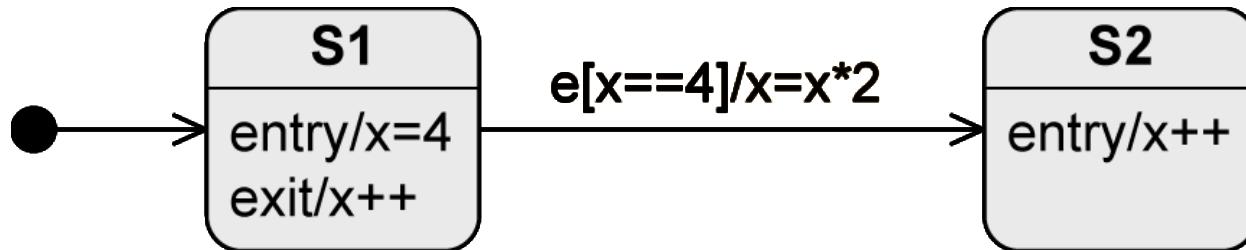


Imediat ce execuția lui **A1** este terminată, se generează un eveniment de finalizare; dacă **g1** este evaluată ca fiind adevărată, tranzitia are loc; dacă nu, această tranzitie nu se poate întâmpla niciodată



Tranziție - Secvență de executare a activității

- Presupunând că **s1** este starea activă, care este valoarea lui x după ce se realizează **e**?



S1 devine activă, **x** este setat la valoarea **4**

e are loc, condiția este verificată și evaluată ca adevărată

S1 este părăsită, **x** este setat la **5**

tranziția are loc, **x** este setat la **10**

se intră în **S2**, **x** este setat la **11**



Eveniment - Tipuri (1/2)

■ Eveniment de tip Semnal

Primirea unui semnal

- Ex: `click_dreapta`, `trimiteSMS(mesaj)`

■ Eveniment de tip Apel

Apelul unei operații

- Ex: `ocupa(utilizator,sala_de_prelegeri)`,
`înregistrează(examen)`

■ Eveniment de timp

Tranziție bazată pe timp

- Relativă: în funcție de momentul producerii evenimentului
 - Ex: `după(5 secunde)`
- Absolută
 - Ex: `când(oră==16:00)`, `când(data==2020.01.01)`

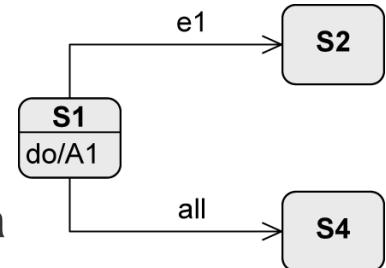


Eveniment - Tipuri (2/2)

▪ Orice eveniment de primire

Se folosește pentru a modela tranzițiile de tip **else**

- Cuvânt cheie **all**
- Din stare **S1** obiectul trece în starea **S2** la producerea evenimentului **e1** și în **S4** la producerea oricărui eveniment



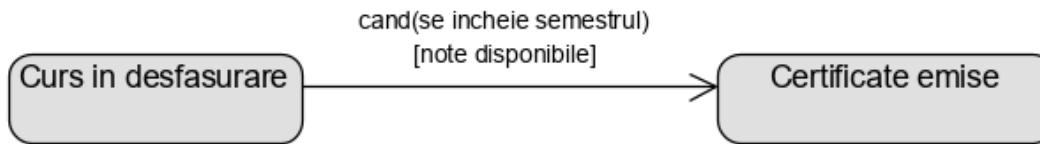
▪ Eveniment de finalizare

Generat automat atunci când tot ceea ce trebuie făcut în starea curentă este finalizat

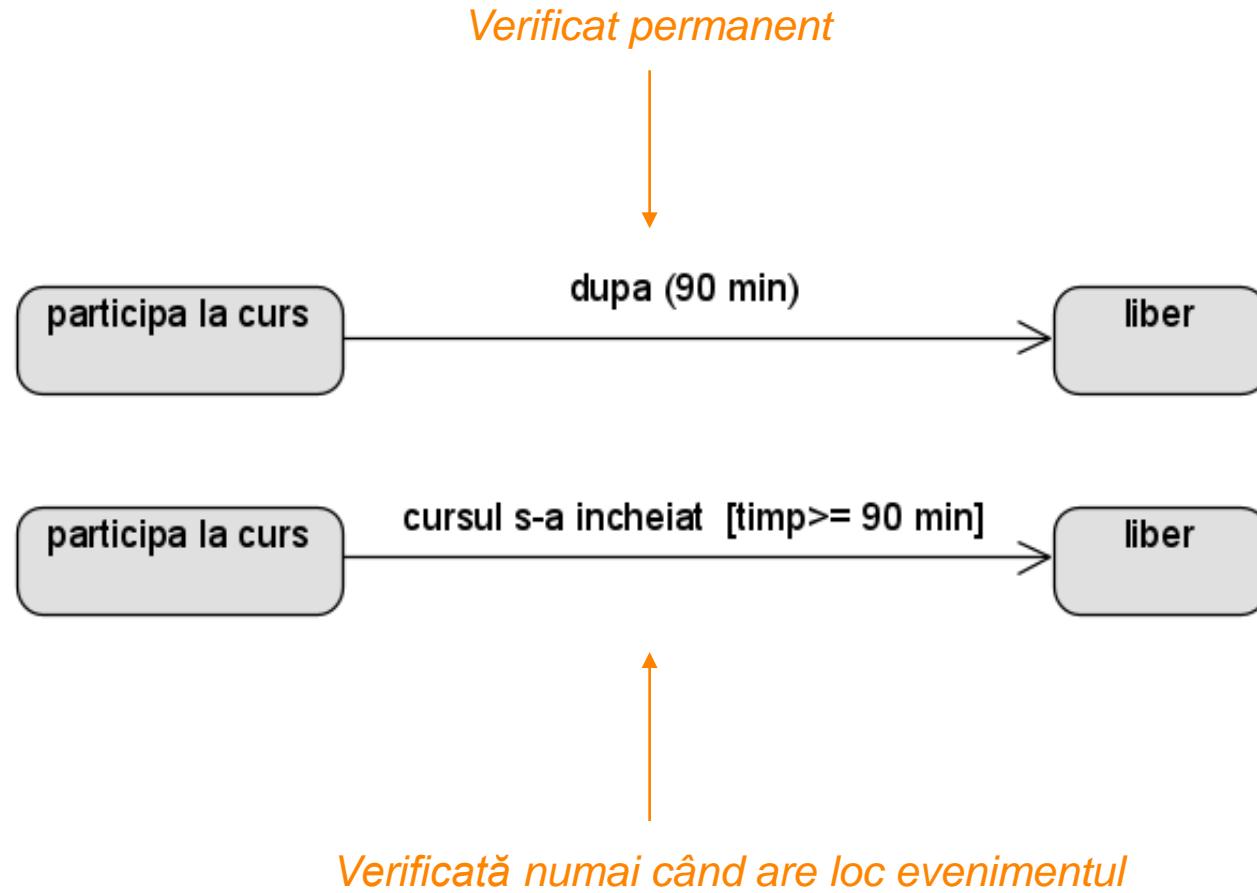
▪ Eveniment de schimbare

Se verifică permanent îndeplinirea unei condiții

- Ex: **când (x > y) , după (90min)**



Eveniment de schimbare vs. Condiție



Stare inițială

- Desemnează “începutul” unei diagrame de stare
- Este o pseudostare
 - Tranzitorie, sistemul nu poate rămâne în acea stare
 - Mai degrabă o structură de control decât o stare reală
- Nu are fluxuri de intrare
- Dacă există mai multe fluxuri de ieșire
 - Condițiile trebuie să fie mutual exclusive și să acopere toate situațiile posibile pentru a se asigura că este atinsă exact o stare țintă
- Dacă starea inițială devine activă, obiectul trece imediat în starea următoare
- Nu se pot asocia activități pseudostărilor



Stare finală și nod de încheiere

Stare finală



- Este o stare reală
- Marchează încheierea secvenței de stări
- Obiectul poate rămâne într-o stare finală pentru totdeauna

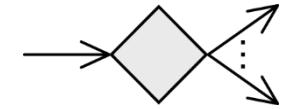
Nodul de încheiere



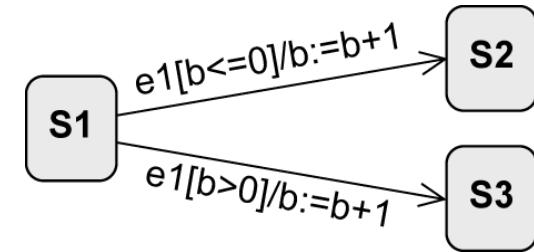
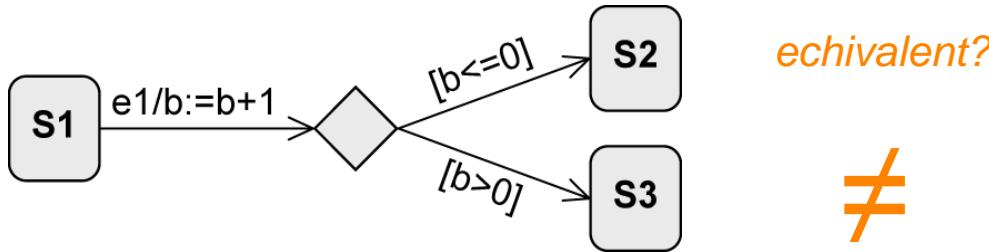
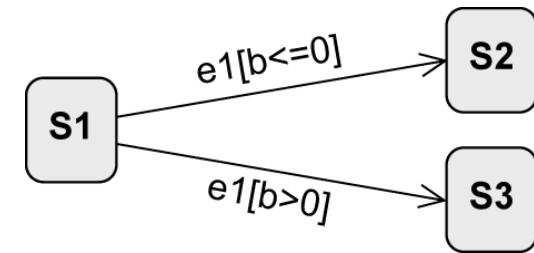
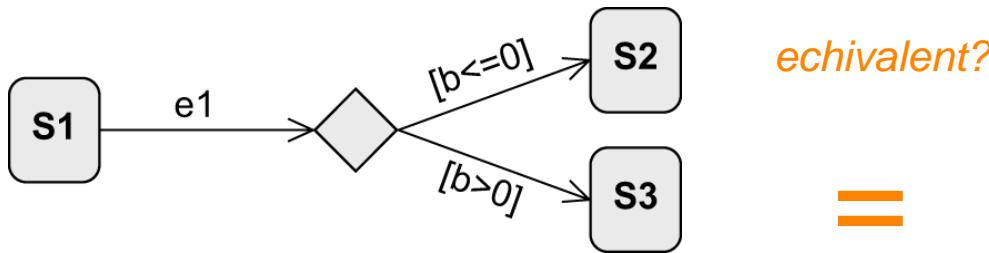
- Este o pseudostare
- Încheie mașina de stări
- Obiectul modelat își încetează existența, adică este șters



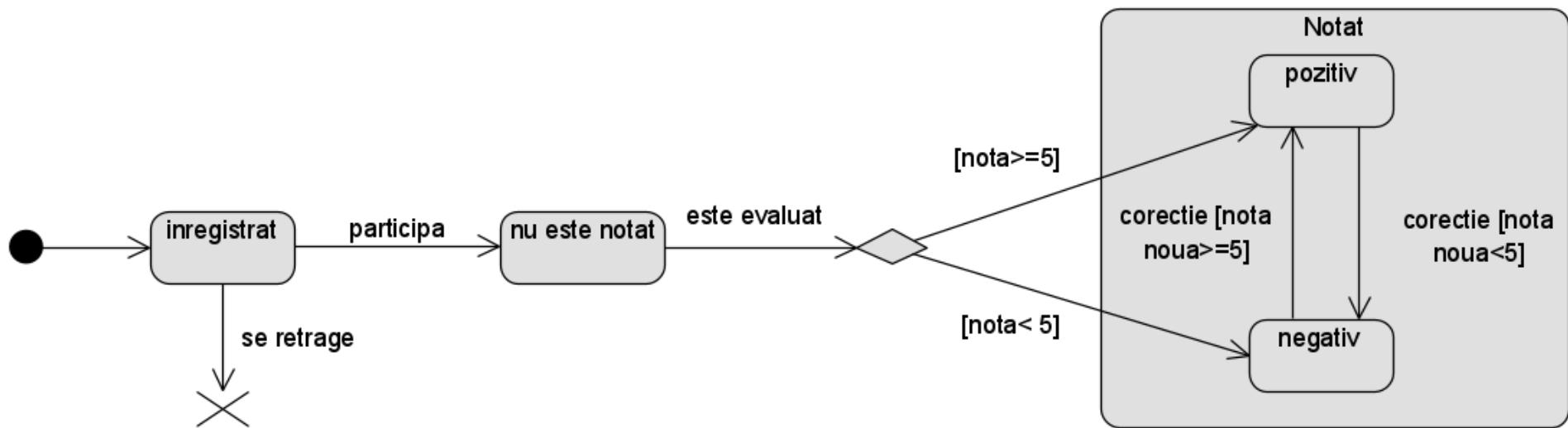
Nod decizional



- Este o pseudostare
- Folosit pentru modelarea tranzițiilor alternative



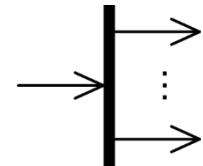
Exemplu: Nod decizional



Noduri de paralelizare și sincronizare

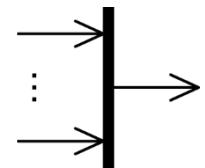
Nodul de paralelizare

- Este o pseudostare
- Împarte fluxul de control în mai multe fluxuri concurente
- Are un singur flux de intrare
- Are mai multe fluxuri de ieșire
- Pe fluxurile de ieșire nu se specifică evenimente sau condiții



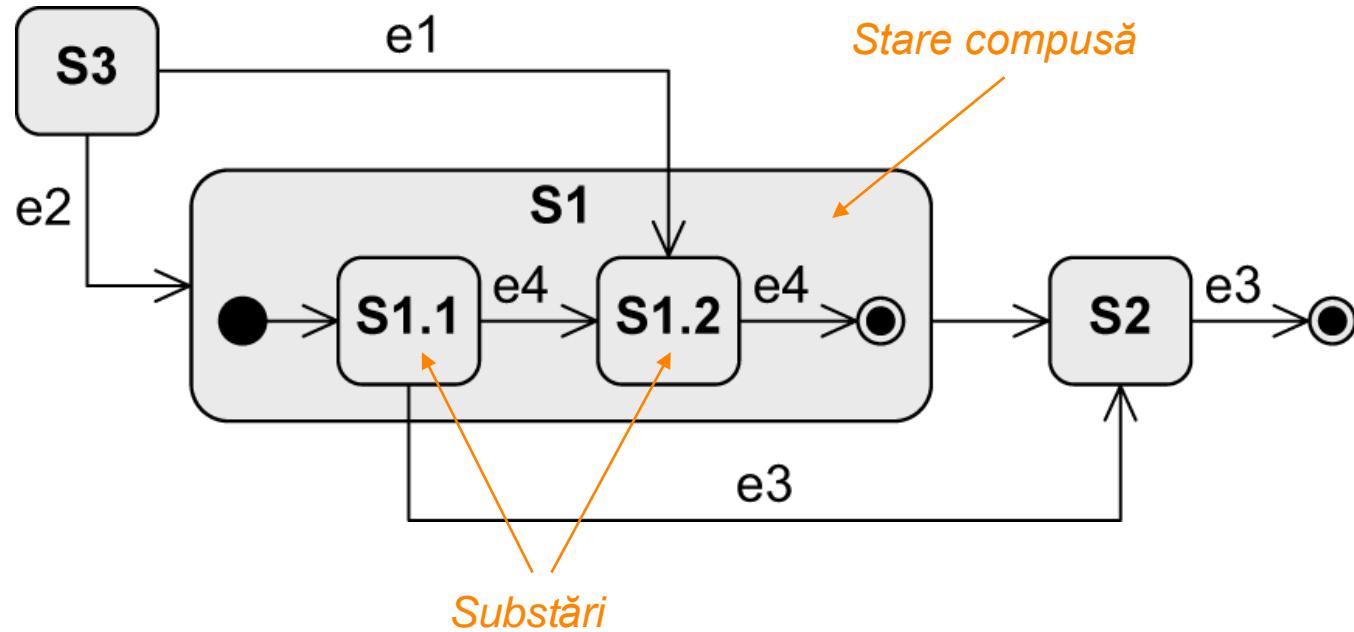
Nodul de sincronizare

- Este o pseudostare
- Fuzionează fluxuri concurente multiple
- Are mai multe fluxuri de intrare
- Are un singur flux de ieșire
- Pe fluxurile de ieșire nu se specifică evenimente sau condiții



Stare compusă

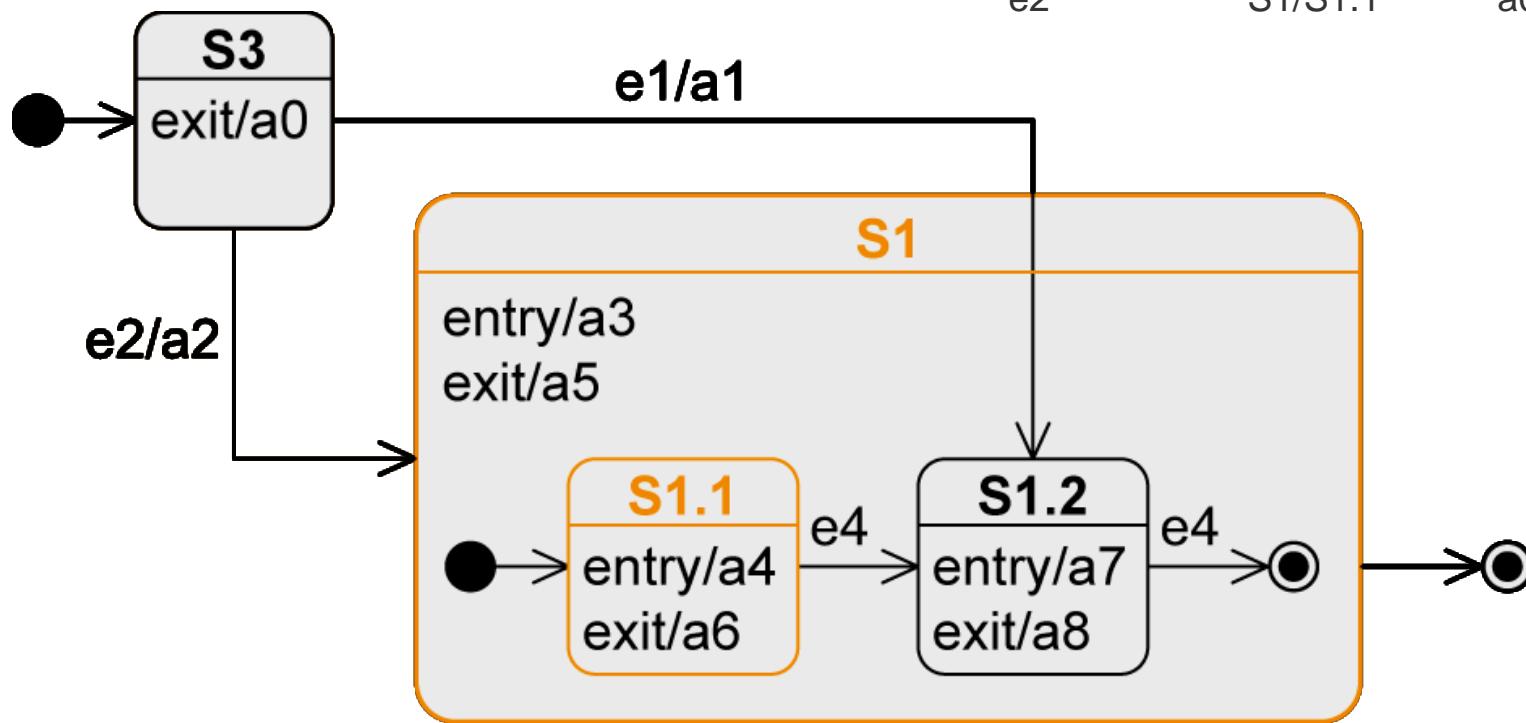
- Sinonime: stare complexă, stare imbricată
- Conține alte stări - „substări”
 - Doar una dintre substările sale este activă la un moment de timp



Intrarea într-o stare compusă (1/2)

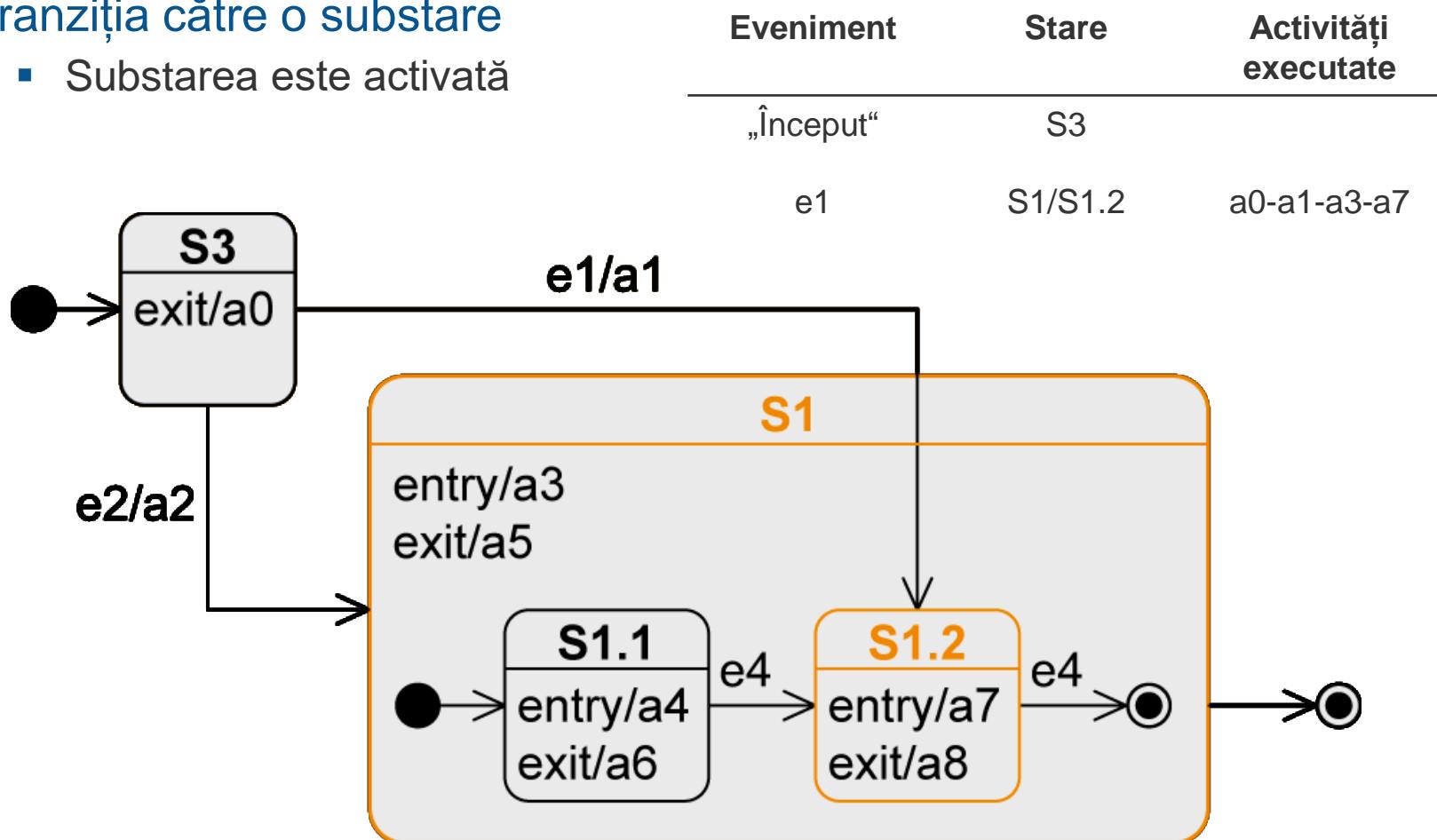
- Tranziție către graniță
 - Nodul initial al stării compuse este activat

Eveniment	Stare	Activități executate
„Început“	S3	
e2	S1/S1.1	a0-a2-a3-a4



Intrarea într-o stare compusă(2/2)

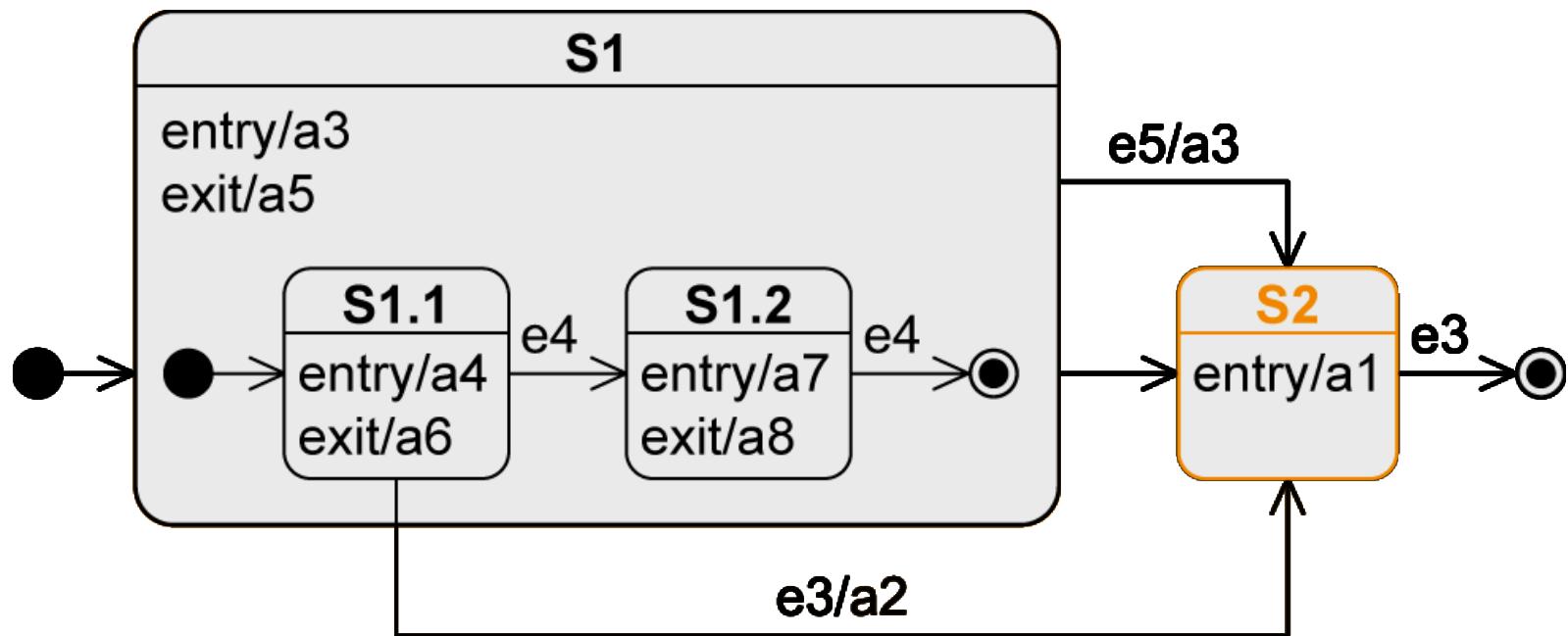
- Tranziția către o substare
 - Substarea este activată



Ieșirea dintr-o stare compusă (1/3)

- Tranzitie dintr-o substare

Eveniment	Stare	Activități executate
„Început“	S1/S1.1	a3-a4
e3	S2	a6-a5-a2-a1

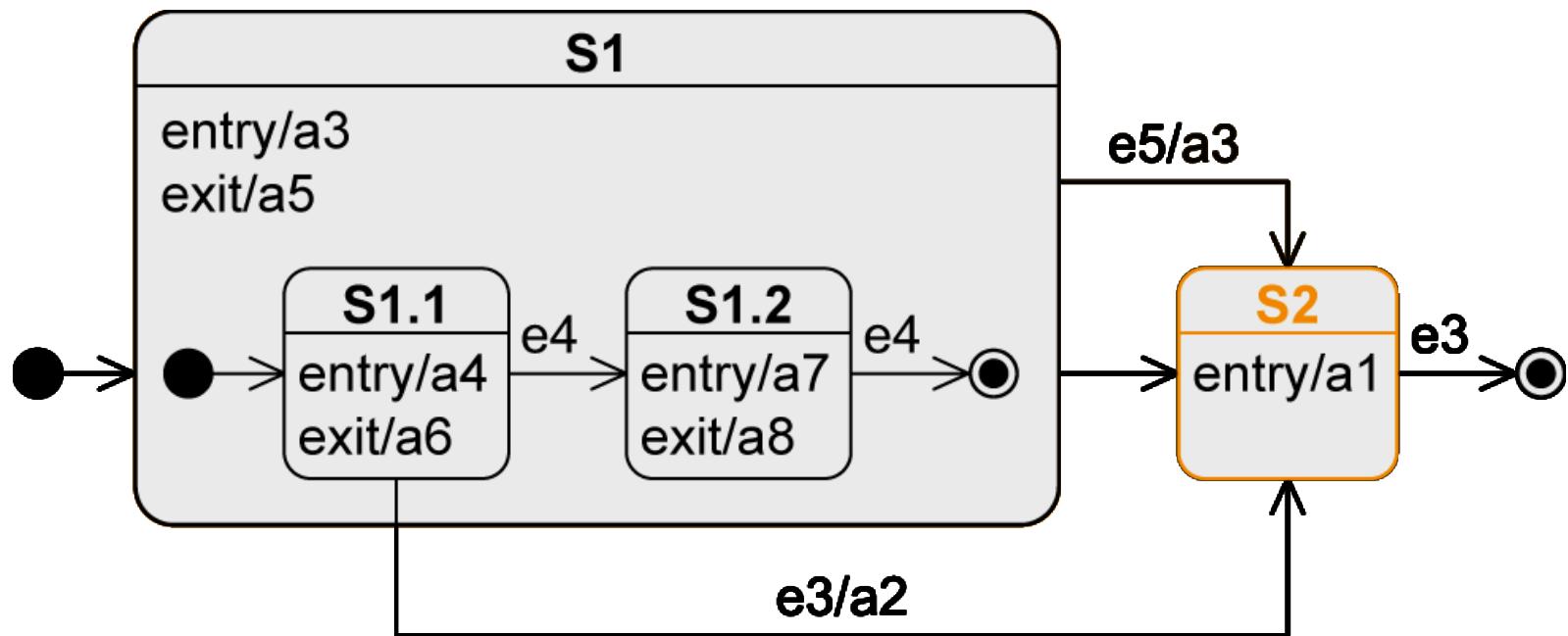


Ieșirea dintr-o stare compusă(2/3)

- Tranzitia din starea compusa

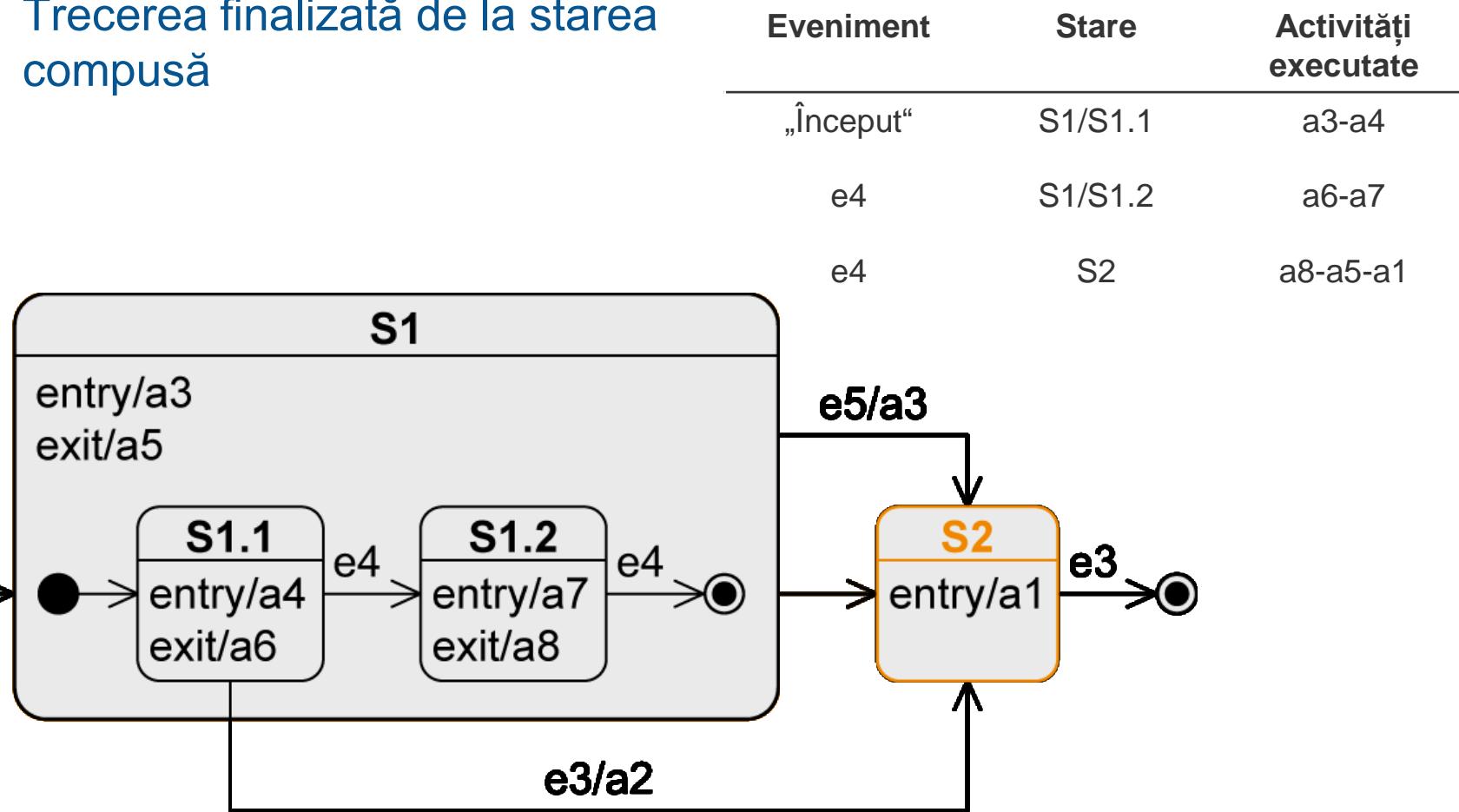
Indiferent ce substare a lui S1 este activă, îndată ce are loc e5, sistemul trece la S2

Eveniment	Stare	Activități executate
„Început“	S1/S1.1	a3-a4
e5	S2	a6-a5-a3-a1



Ieșirea dintr-o stare compusă (3/3)

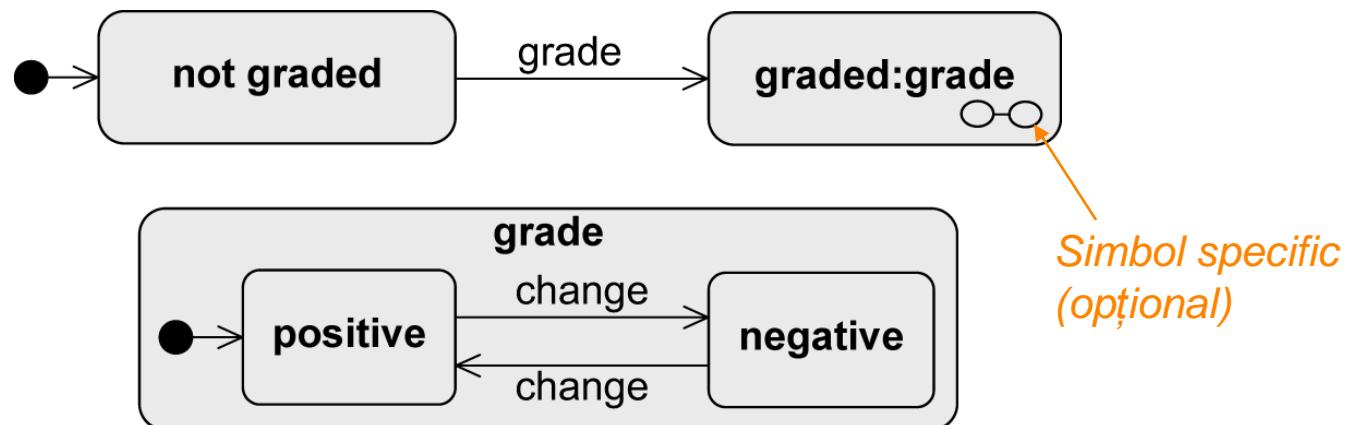
- Trecerea finalizată de la starea compusă





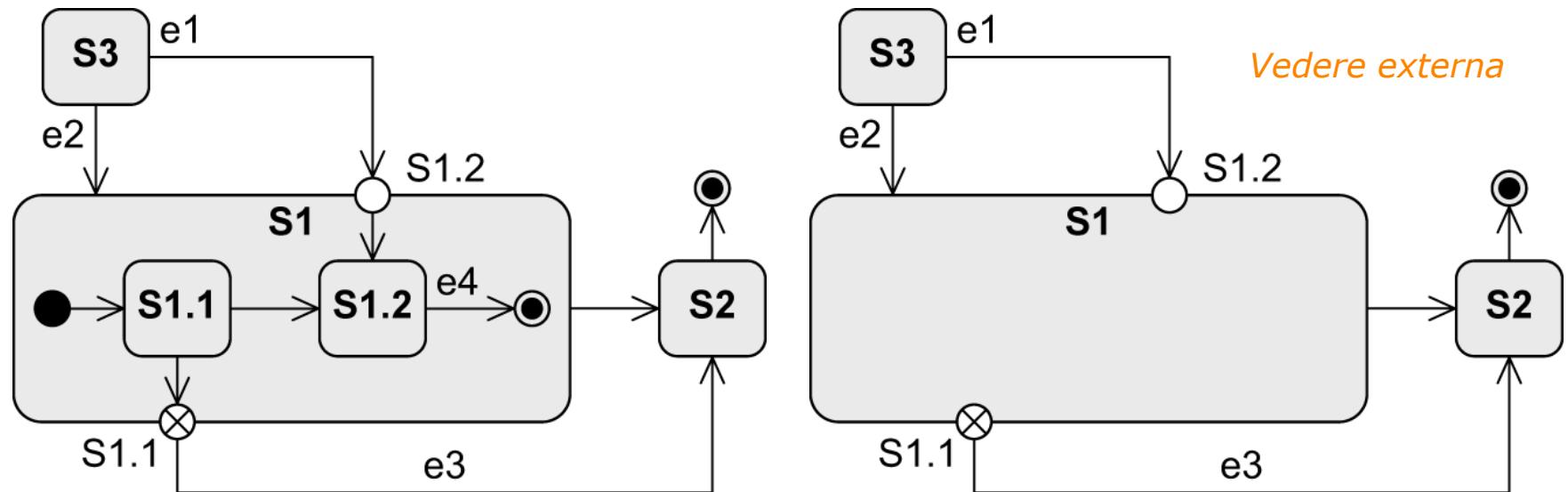
Starea submașină

- Permite reutilizarea părților diagramelor mașinilor de stare în alte diagrame ale mașinilor de stare
- Notatie: stare:stareSubmasina
- De îndată ce starea submașinii este activată, este executat comportamentul acesteia
 - Corespunde apelului unei subruteine în limbaje de programare

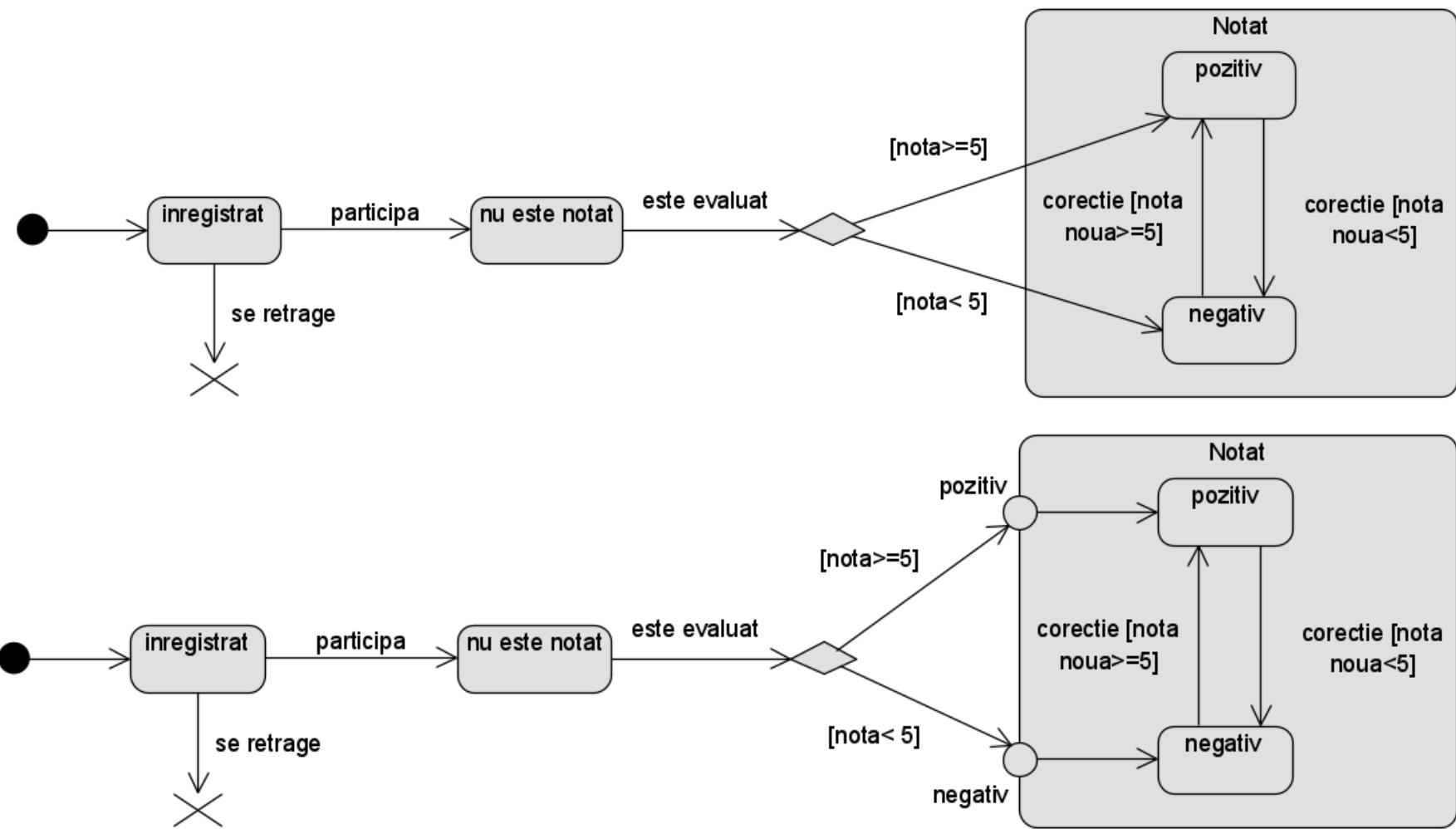


Puncte de intrare și ieșire

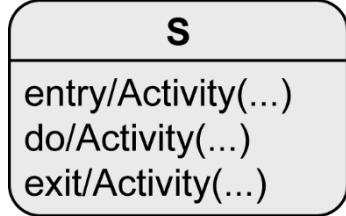
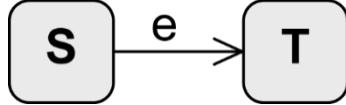
- Mecanism de încapsulare
 - Se va intra sau se va ieși dintr-o stare compusă printr-o altă stare decât stările inițiale și finale
 - Tranziția externă nu trebuie să cunoască structura stării compuse
 - Punctele de intrare și ieșire reprezintă un tip de mecanism de încapsulare



Exemplu: Puncte de intrare și ieșire

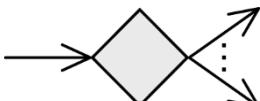
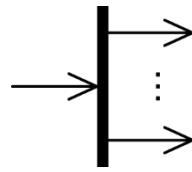
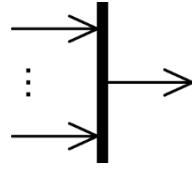


Elemente de notație (1/2)

Nume	Notație	Descriere
Stare		Descrierea unui „interval de timp” specific în care un obiect se regăsește pe parcursul „ciclului său de viață”. În cadrul unei stări, diferite activități pot fi executate de obiect.
Tranzitie		Tranzitia de la o stare sursa S la o stare țintă T
Stare inițială		Începutul unei diagrame de mașini de stări
Stare finală		Încheierea unei diagrame de mașini de stări
Nod de încheiere		Încheierea diagramei de mașini de stări a unui obiect



Elemente de notație (2/2)

Nume	Notație	Descriere
Nod decizional		Nod din care pot origina tranziții alternative multiple
Nod de paralelizare		Divizarea unei tranziții în mai multe tranziții paralele
Nod de sincronizare		Îmbinarea mai multor tranziții paralele într-o singură tranziție



Diagramele de interacțiu

*Modelarea
dinamică*

Introducere

- Modelarea comportamentului care implică mai multe obiecte
 - = presupune interacțiunea între obiecte
- Interacțiunea
 - Specifică modul cum mesajele și datele sunt schimbate între partenerii care interacționează
 - Partenerii interacțiunii
 - Uman (profesor, administrator, ...)
 - Non-uman (server, imprimantă, software executabil, ...)
- Example de interacțiuni
 - Conversații între persoane
 - Schimbul de mesaje între oameni și sisteme software
 - Protocole de comunicare
 - Secvențe de metode într-un program
 - ...

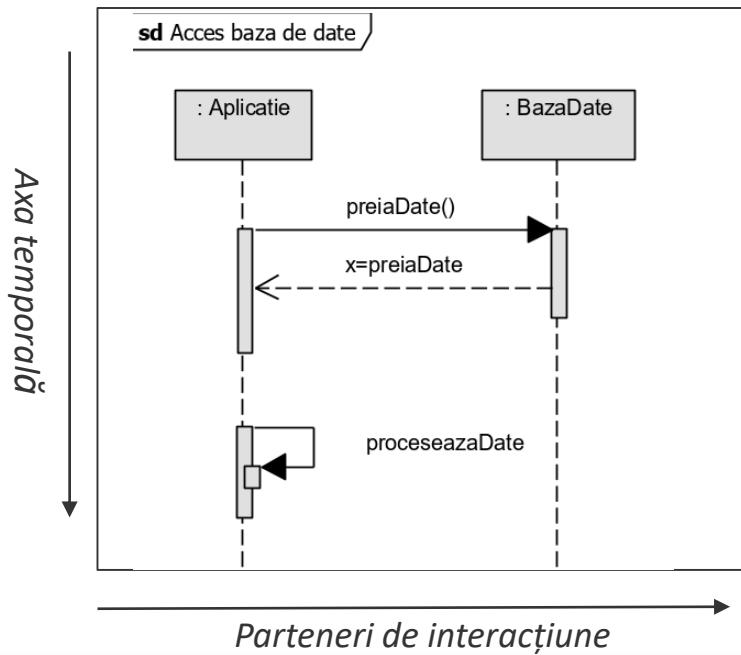
Diagramele de interacțiune

- Folosite pentru a specifica interacțiuni
- Modeleză scenarii concrete
- Descriu secvențe de comunicare la diferite nivele de detaliu

- Diagramele de interacțiune pot descrie:
 - Interacțiunea unui sistem cu mediul său
 - Interacțiunea dintre părți ale sistemului în scopul de a arăta cum poate fi implementat un caz de utilizare
 - Comunicarea dintre procese în care partenerii implicați trebuie să respecte anumite protocoale
 - Comunicarea la nivelul claselor (apeluri de operații, comportamentul interobiecte)

Diagrama de secvență

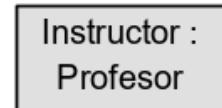
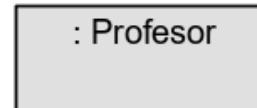
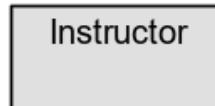
- Este o diagramă bidimensională
 - Axa orizontală: arată partenerii care interacționează
 - Axa verticală: ordinea cronologică a interacțiunilor
- Interacțiunea = secvență de specificații ale unor evenimente



Parteneri în cadrul interacțiunii

- Partenerii care interacționează sunt descriși sub forma unor linii de viață
- Partea superioară a liniei de viață
 - Dreptunghi care conține expresia **NumeRol:Clasă**
 - Numele de rol sunt un concept mai general decât obiectele
 - Obiectul poate juca diferite roluri de-a lungul liniei sale de viață
- Partea inferioară a liniei de viață
 - Linie verticală, de obicei, punctată
 - Reprezintă durata de viață a obiectului asociat acesteia

Partea superioară
a liniei de viață

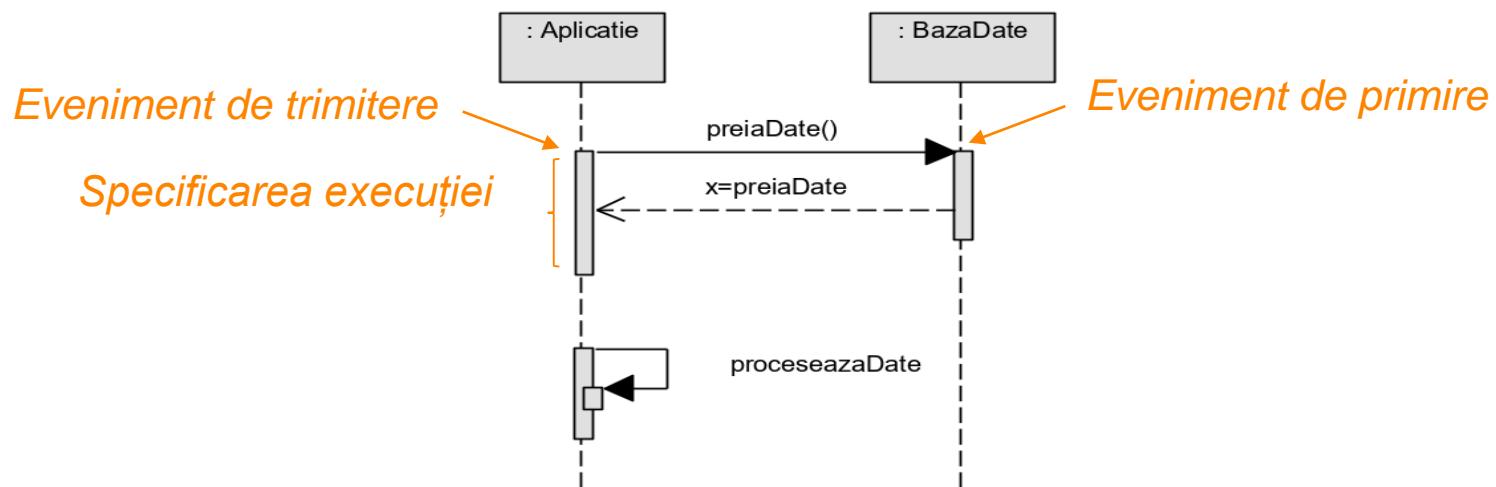


Partea inferioară a
liniei de viață



Schimbul de mesaje (1/2)

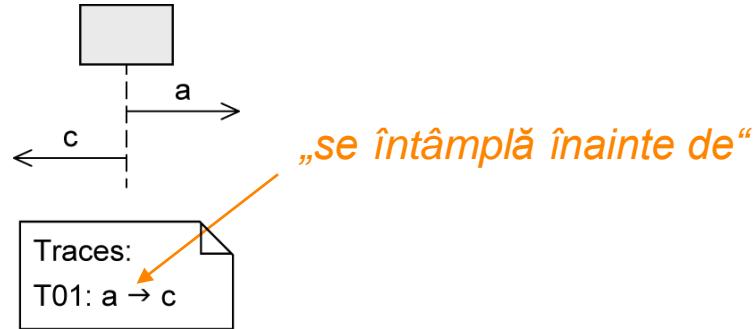
- Interacțiune: secvență de evenimente
- Mesajele sunt definite prin intermediul evenimentului de trimis și a evenimentului de primire
- Specificarea execuției
 - Reprezentată ca un dreptunghi subțire
 - Folosită pentru a vizualiza când un partener execută un anumit comportament



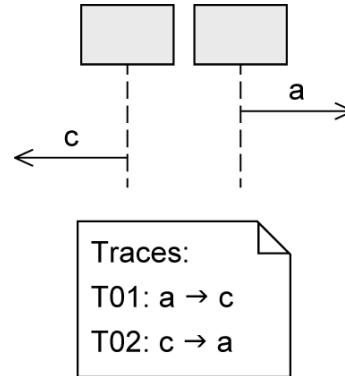
Schimbul de mesaje (2/2)

Ordinea mesajelor:

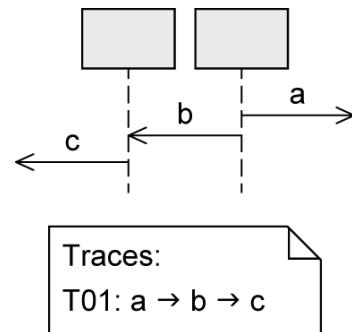
... pe o linie de viață



... pe linii de viață diferite



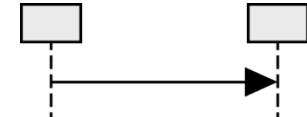
... pe linii de viață diferite care schimbă mesaje



Mesaje (1/3)

■ Mesaj sincron

- Emițătorul așteaptă până primește un mesaj de răspuns înainte de a continua
- Sintaxa numelui mesajului: **mesaj (par1 , par2)**
 - **mesaj**: numele mesajului
 - **par**: parametrii separați prin virgulă



■ Mesaj asincron

- Emițătorul continuă fără să aștepte un mesaj de răspuns
- Sintaxa numelui mesajului: **mesaj (par1 , par2)**



■ Mesaj răspuns

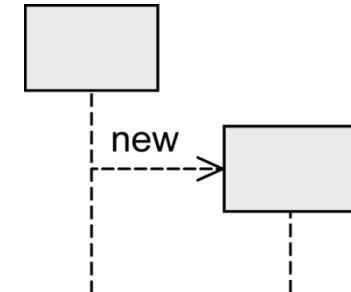
- Poate fi omis dacă locația sau contextul sunt evidente
- Sintaxa: **atrib=mesaj (par1 , par2) : val**
 - **atrib**: valoarea returnată poate fi atribuită optional unei variabile
 - **mesaj**: numele mesajului
 - **par**: parametrii separați prin virgulă
 - **val**: valoarea returnată



Mesaje (2/3)

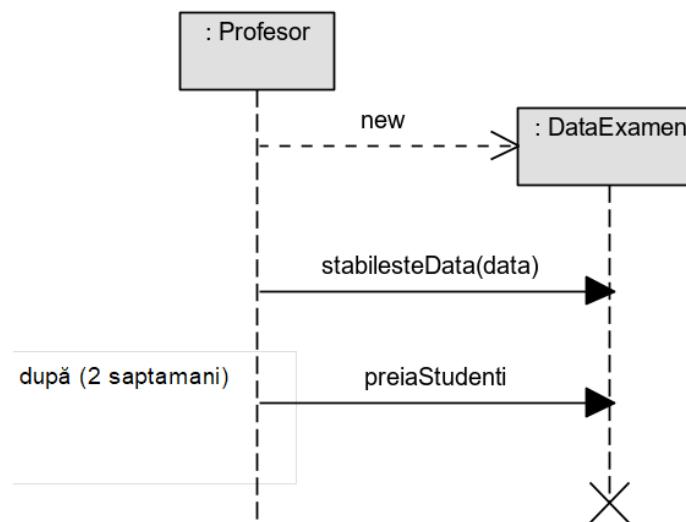
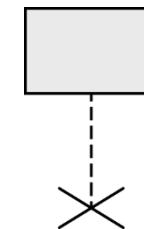
Crearea obiectului

- Reprezentată prin linie punctată
- Vârful săgeții indică înspre linia de viață a obiectului creat
- Se folosește cuvântul cheie **new**



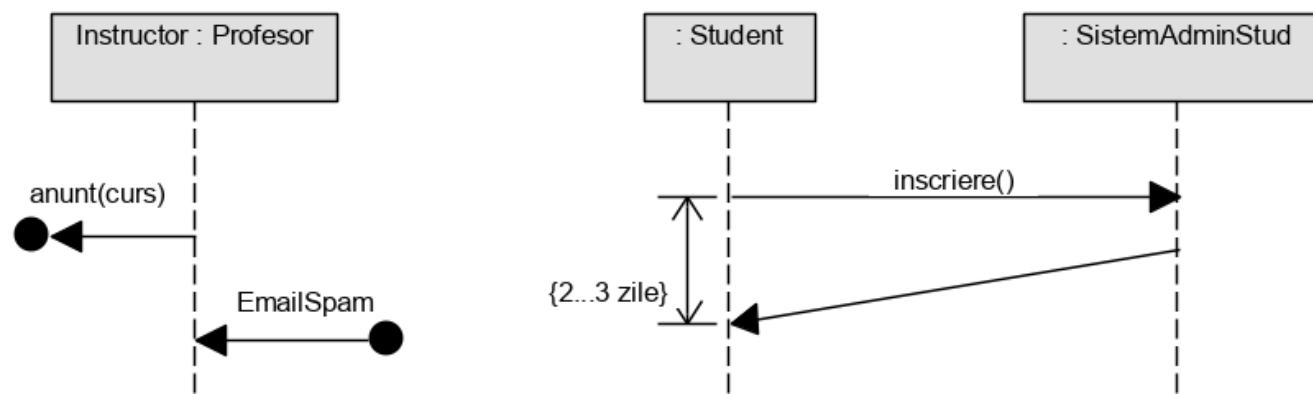
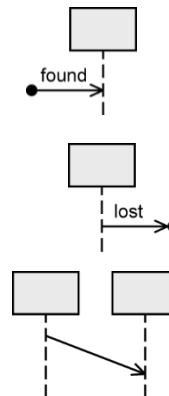
Distrugerea obiectului

- Obiectul este șters
- Reprezentată cu (×) la finalul liniei de viață



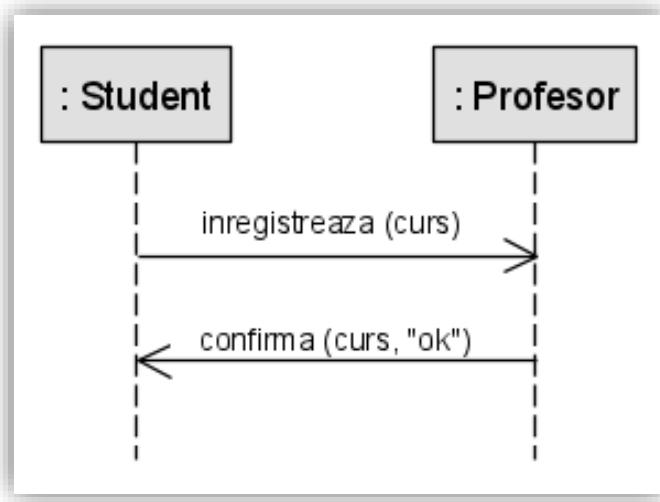
Mesaje (3/3)

- Mesaj găsit
 - Emitterul unui mesaj este necunoscut sau irrelevant
- Mesaj pierdut
 - receptorul unui mesaj este necunoscut sau irrelevant
- Mesaj consumator de timp
 - Poate fi numit și “mesaj cu durată”
 - De obicei, se presupune că mesajele se transmit fără nicio pierdere de timp
 - Exprimă faptul că trece un timp între trimiterea și primirea mesajului

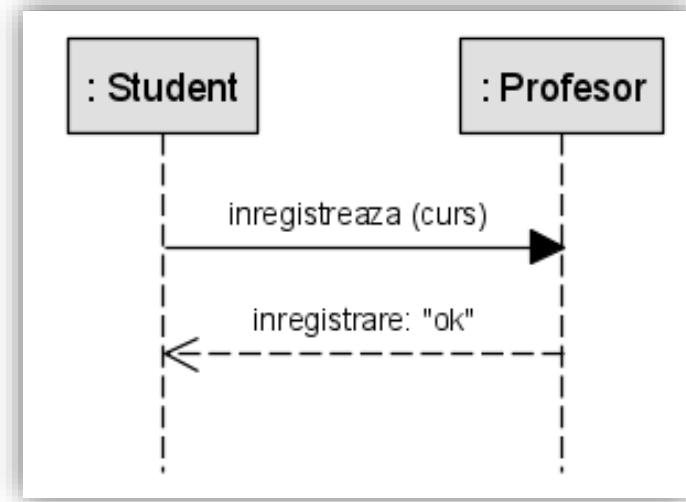


Mesaje asincron/sincron

A



B



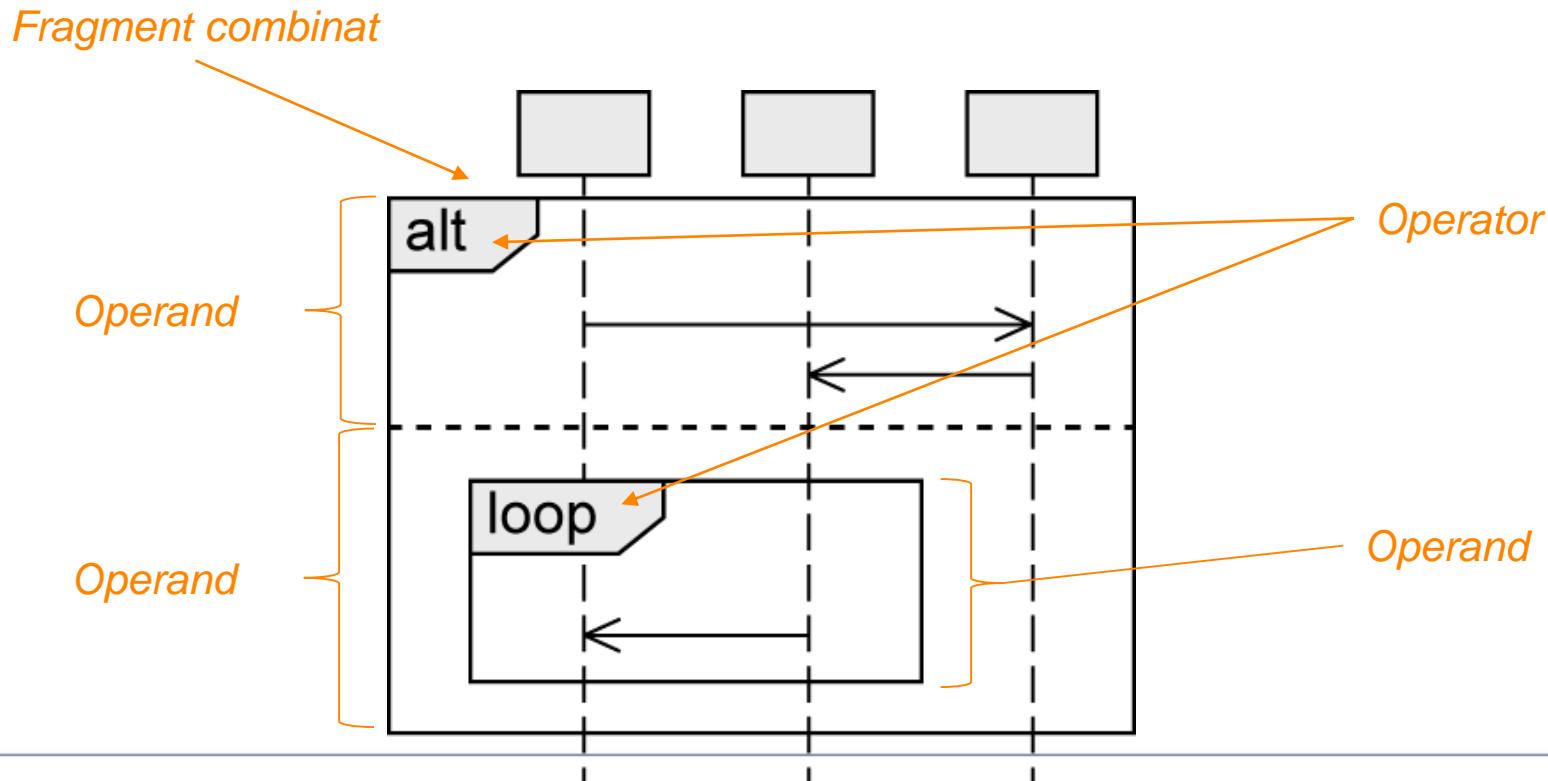
Comunicare prin două mesaje asincrone

Comunicare printr-un mesaj sincron și un mesaj răspuns

Student așteaptă primirea unui răspuns înainte de a continua

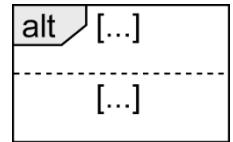
Fragmente combinate

- Modeleză diferite tipuri de structuri de control
- Permit să introducem logică procedurală în diagrama de secvență
- Există 12 tipuri predefinite de operatori



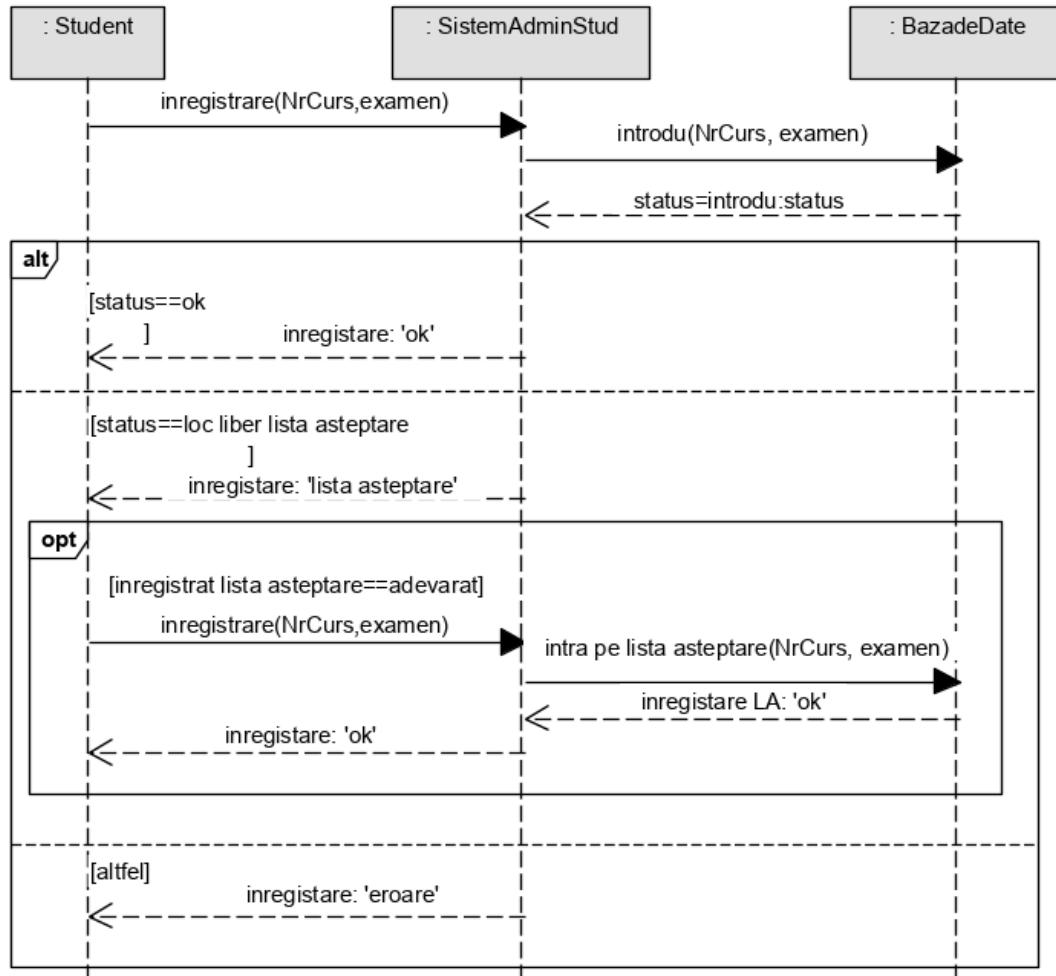
Tipuri de fragmente combinate

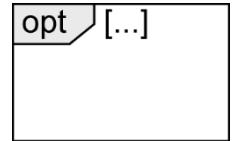
	Operator	Scop
Alternativ și iterativ	alt	Interacțiune alternativă
	opt	Interacțiune optională
	loop	Interacțiune repetată
	break	Interacțiune excepție
Conurență și ordonare	seq	Ordonare slabă
	strict	Ordonare strictă
	par	Interacțiune concurentă
	critical	Interacțiune atomică
Filtre și aserțiiuni	ignore	Interacțiune irelevantă
	consider	Interacțiune relevantă
	assert	Interacțiune confirmată
	neg	Interacțiune invalidă



Fragmentul alt

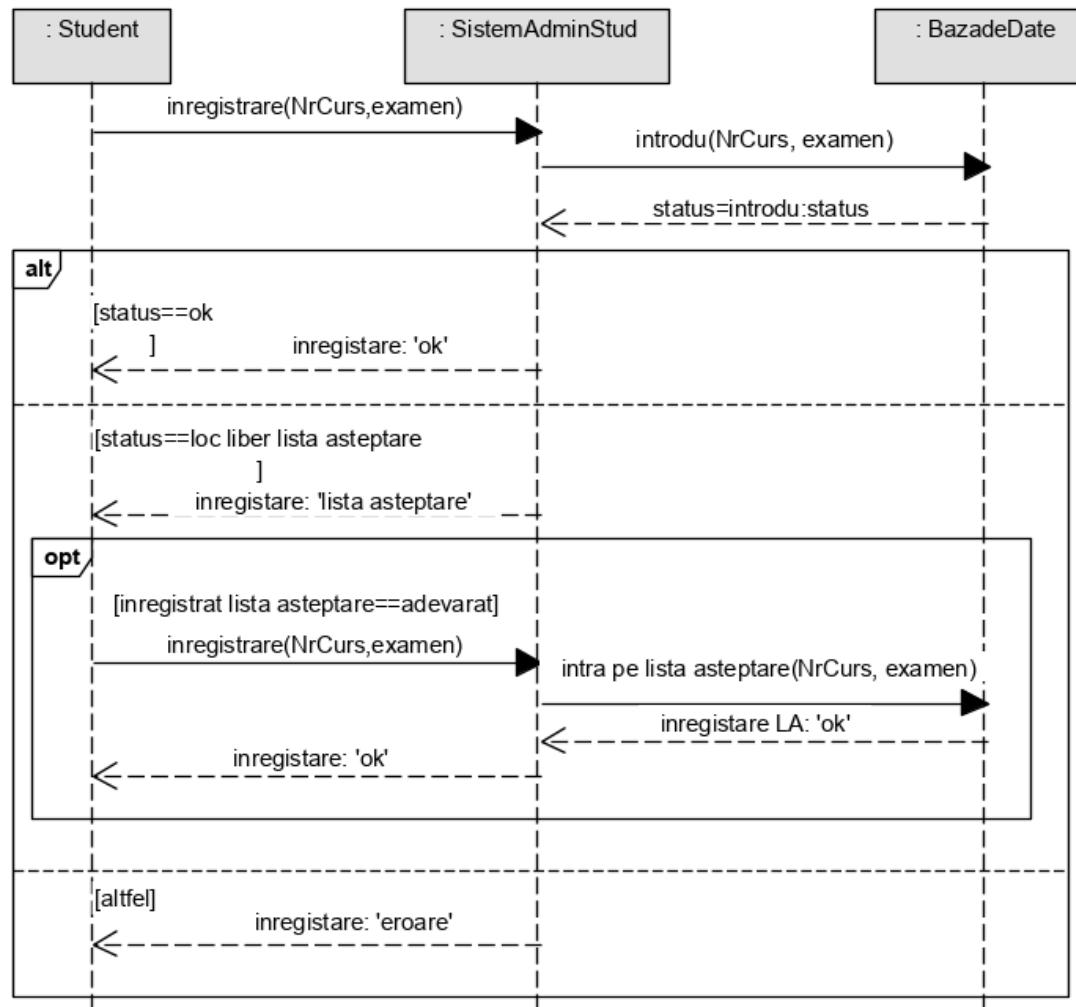
- Folosit pentru modelarea sevențelor alternative
- Similar structurii **switch** din Java
- Sunt folosite condiții pentru a selecta o singură cale de execuție
- Condițiile
 - Descrise între paranteze drepte
 - implicit: true
 - predefinită: [else]
- Pot exista operanzi mulți
- Condițiile trebuie să fie mutual exclusive





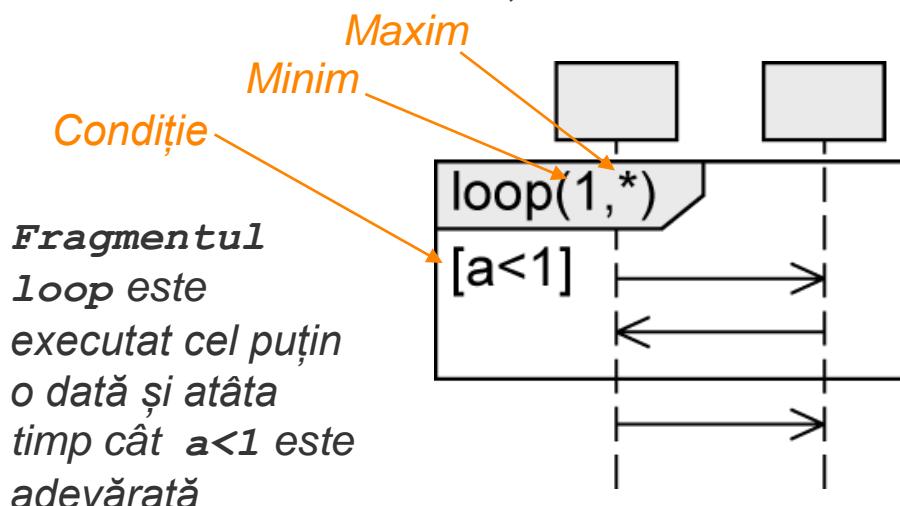
Fragmentul opt

- Modelează secvențe optionale
- Execuția la momentul rulării este dependentă de condiție
- Are doar un singur operand
- Similară structurii **if** fără ramura **else**
- Echivalent fragmentului **alt** cu doi operanzi, dintre care unul este gol



Fragmentul loop

- Exprimă o secvență care trebuie executată în mod repetat
- Are un singur operand
- Desemnat prin cuvântul cheie **loop**, urmat de numărul minim/maxim de iterații (**min..max**) sau (**min,max**)
 - implicit: (*) .. fără limită superioară
- Condiție
 - Evaluată imediat ce a fost atins numărul minim de operații
 - Verifică pentru fiecare iterare dacă se încadrează în limitele (**min,max**)
 - Atunci când condiția este evaluată ca falsă, execuția este terminată



Notări alternative:

$$\text{loop}(3, 8) = \text{loop}(3..8)$$

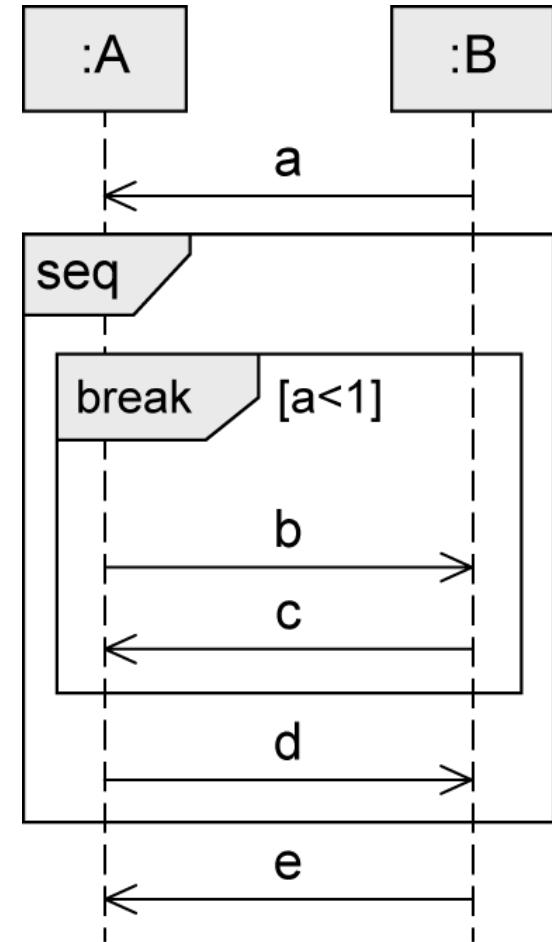
$$\text{loop}(8, 8) = \text{loop}(8)$$

$$\text{loop} = \text{loop}(\ast) = \text{loop}(0,\ast)$$

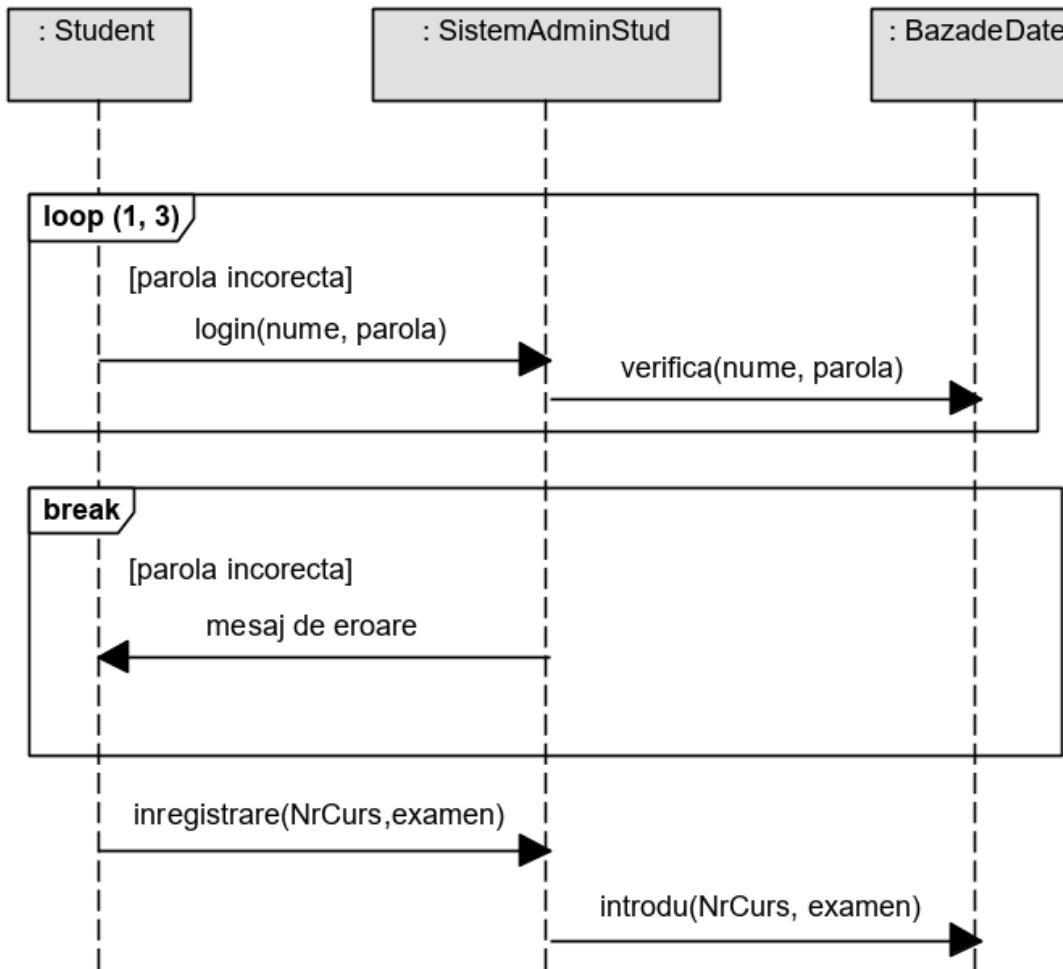
Fragmentul break

- Modalitate de tratare a exceptiilor
- Exact un singur operand cu o conditie
- Atunci cand conditia este adevarata:
 - Se execută interacțiunile din cadrul acestui operand
 - Restul operațiilor ale fragmentului din jurul său vor fi omise
 - Interacțiunile continuă în următorul fragment de nivel mai înalt
 - Comportament diferit față de opt

Nu se execută dacă este executat break

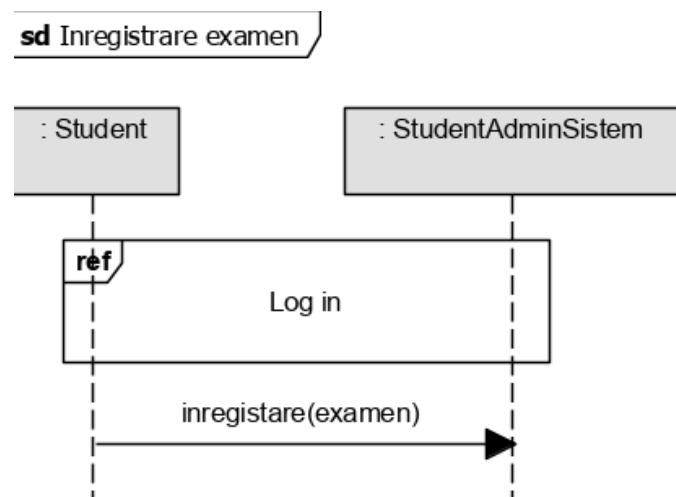
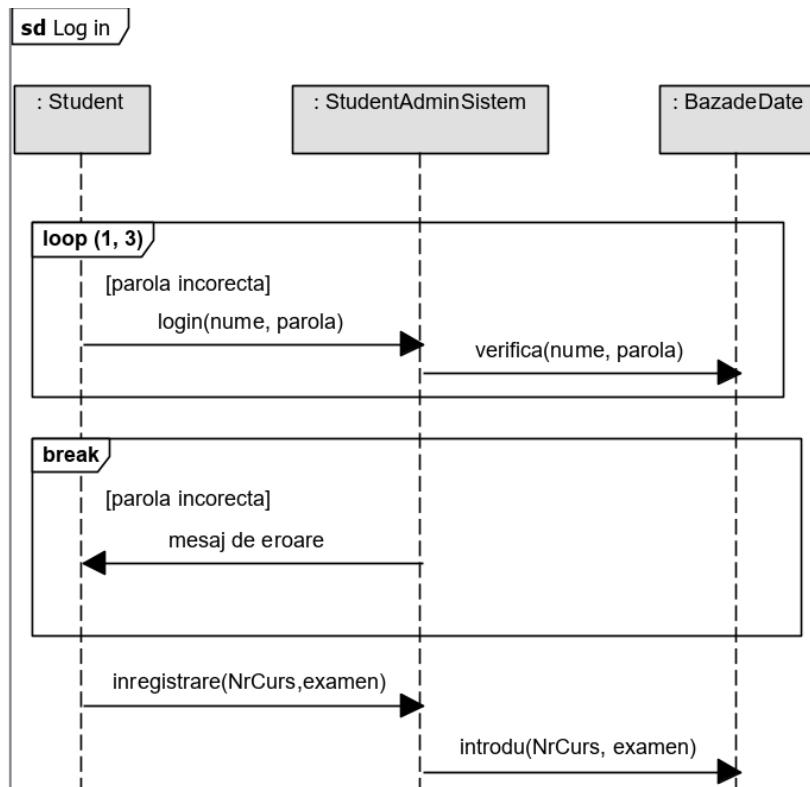


Fragmentele loop și break - Exemplu



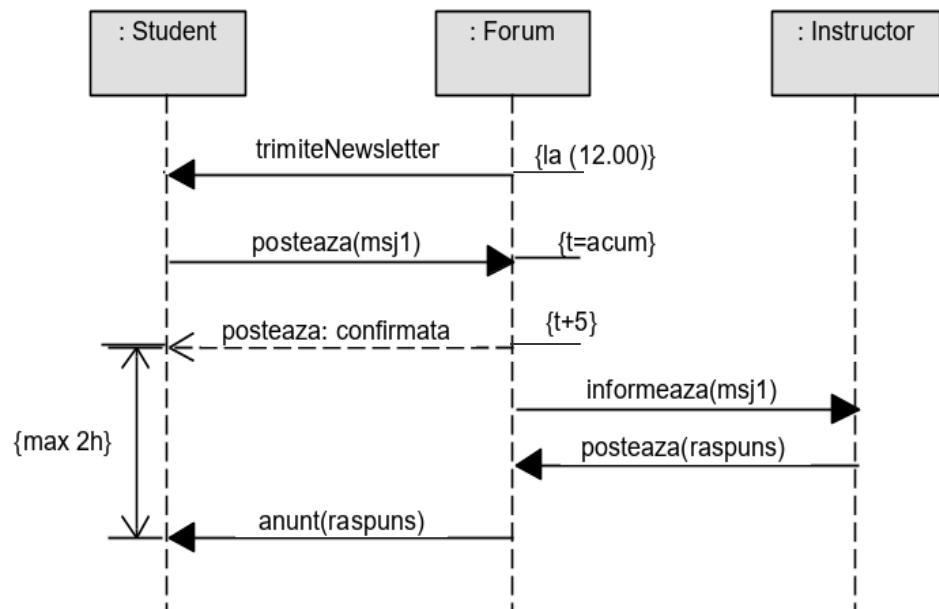
Referință interacțiunii

- Integrează o diagramă de secvență în altă diagramă de secvență



Constrângeri de timp

- Tipuri
 - Moment de timp pentru producerea unui eveniment
 - Relativ: ex. **după (5sec)**
 - Absolut: ex. **la (12.00)**
 - Perioadă de timp între două evenimente
 - **{min..max}**
 - Ex. **{12.00..13.00}**
- Acțiuni predefinite
 - **acum**: momentul curent
 - Poate fi alocată unui atribut și folosită apoi într-o constrângere de timp
 - Durata: calculul duratei transmiterii unui mesaj



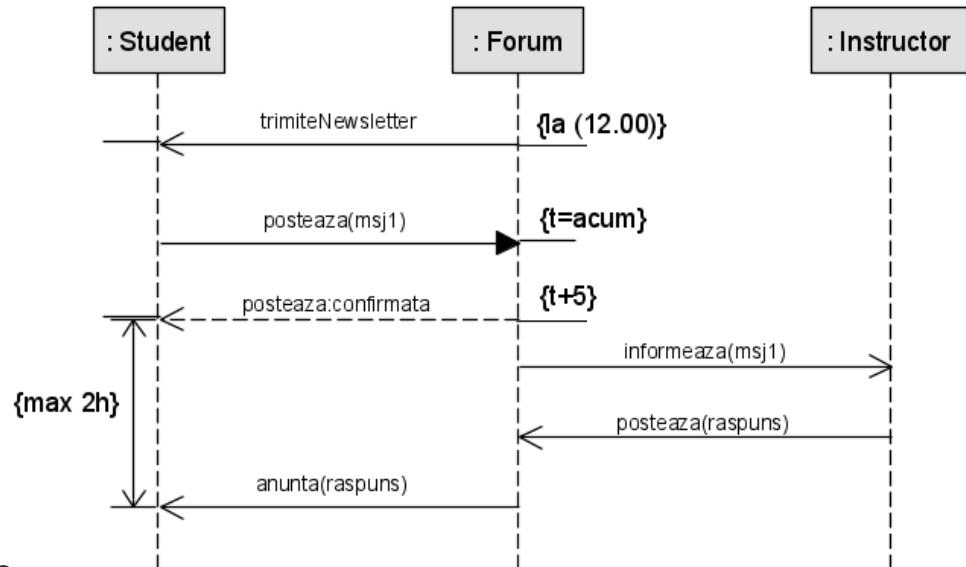
Constrângeri de timp

■ Tipuri

- Moment de timp pentru producerea unui eveniment
 - Relativ: ex. **după (5sec)**
 - Absolut: ex. **la (12.00)**
- Perioadă de timp între două evenimente
 - **{min..max}**
 - Ex. **{12.00..13.00}**

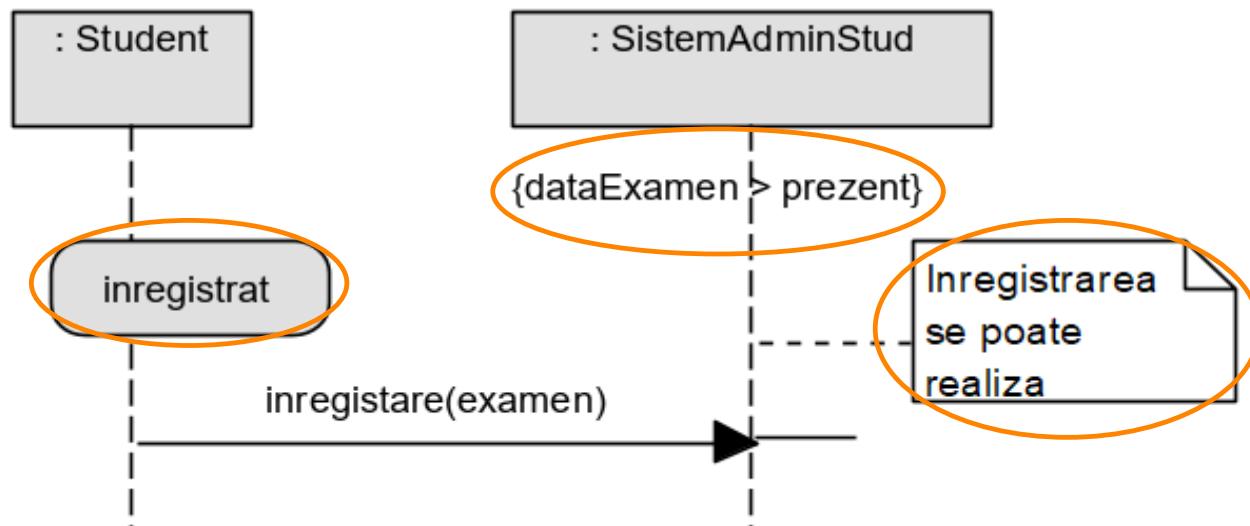
■ Acțiuni predefinite

- **acum**: momentul curent
 - Poate fi alocată unui atribut și folosită apoi într-o constrângere de timp
- Durata: calculul duratei transmiterii unui mesaj



Invarianță

- Arată că o anumită condiție trebuie îndeplinită la un moment de timp
- Este întotdeauna alocată unui linii de viață
- Este evaluată înainte de apariția unui eveniment ulterior
- Dacă invarianță nu este adevărată, atunci fie modelul, fie implementarea, sunt incorecte
- Trei notații alternative

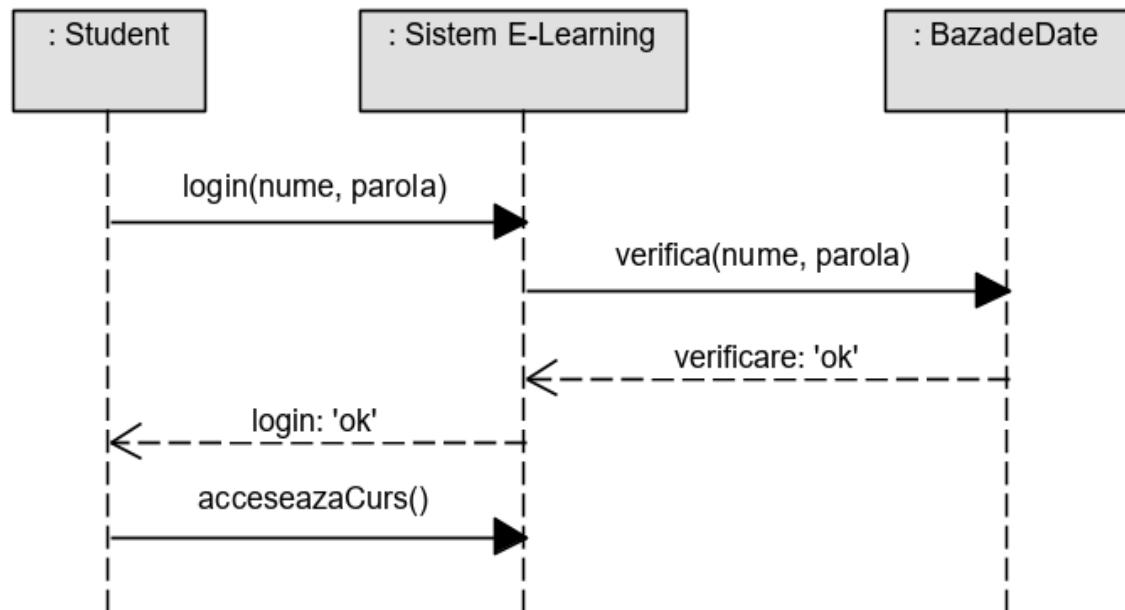


Patru tipuri de diagrame de interacțiuie (1/4)

- Bazate pe aceleași concepte
- În general sunt echivalente pentru interacțiuni simple, dar din perspective diferite

▪ Diagrama de secvență

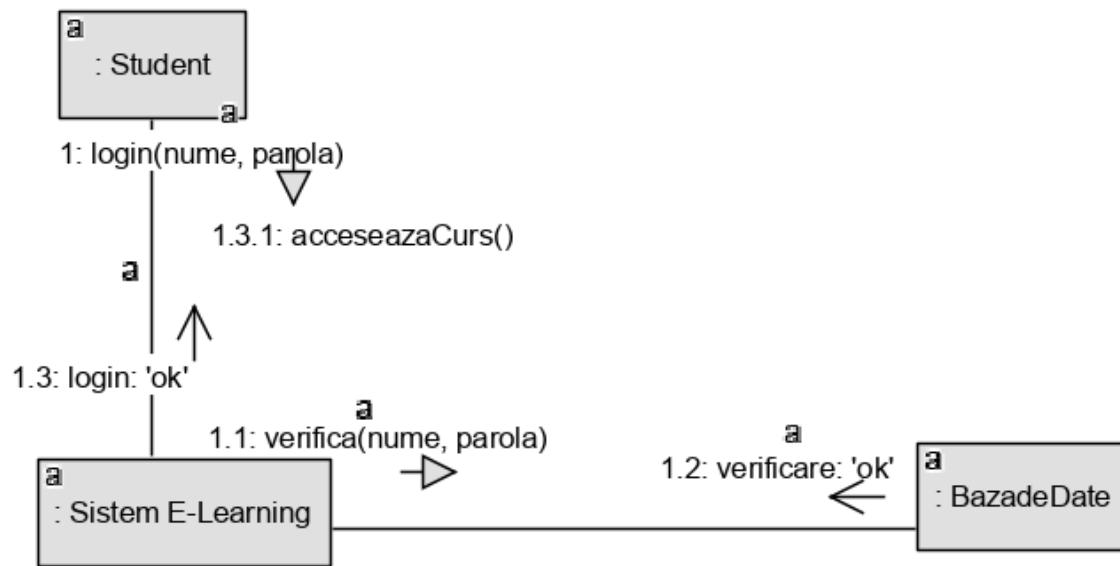
- Axa verticală:
ordinea cronologică
- Axa orizontală:
partenerii interacțiunii



Patru tipuri de diagrame de interacțiuie (2/4)

■ Diagrama de comunicare

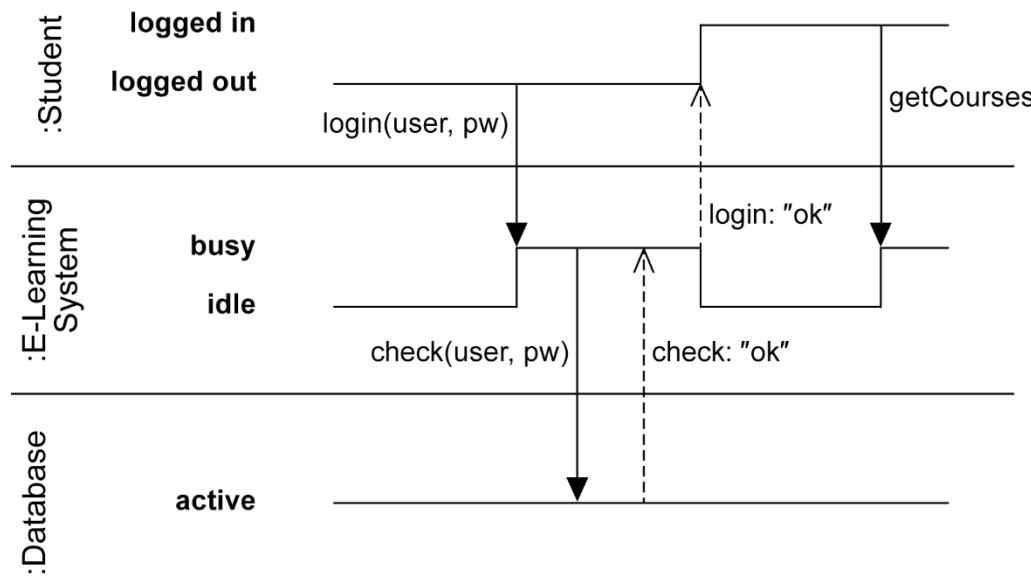
- Modeleză relațiile dintre partenerii care comunică
- Privesc interacțiunile din perspectiva: cine cu cine comunică
- Timpul nu este definit ca o dimensiune separată
- Ordinea mesajelor este dată de un prefix asociat acestora



Patru tipuri de diagrame de interacție (3/4)

■ Diagrama de timp

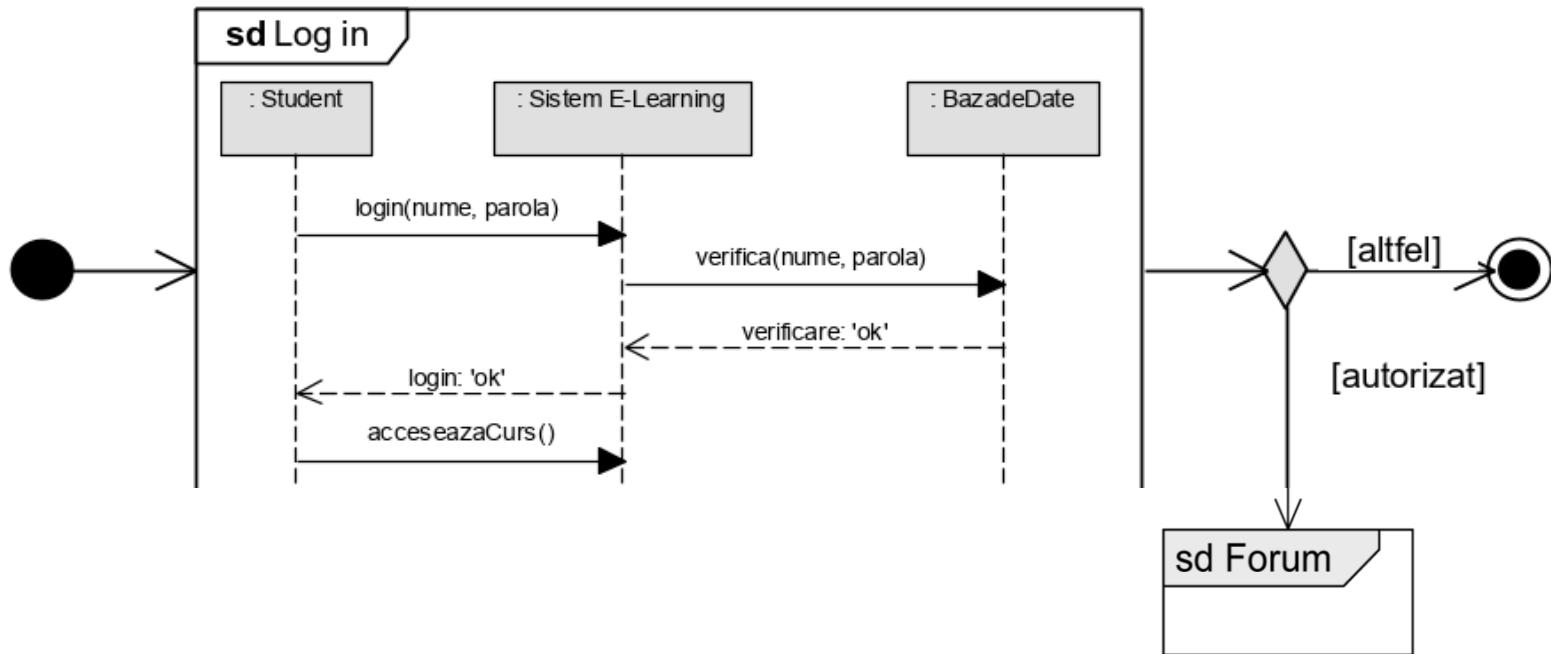
- Arată schimbările stărilor partenerilor care interacționează ca rezultat al apariției evenimentelor
- Axa verticală: partenerii care interacționează
- Axa orizontală: ordinea cronologică



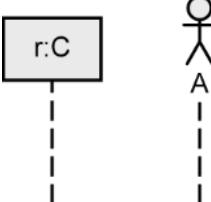
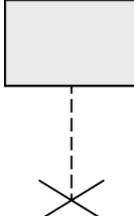
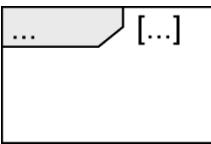
Patru tipuri de diagrame de interacțiune (4/4)

■ Diagrama interacțiunilor de ansamblu

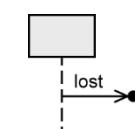
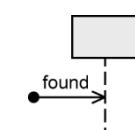
- Vizualizează ordinea diferitelor interacțiuni
- Permite aranjarea diferitelor diagrame de interacțiune într-o ordine logică
- Folosesc notațiile de bază din diagrama de activitate



Notății (1/2)

Nume	Notătie	Descriere
Linie de viață		Partenerii care interacționează și sunt implicați în comunicare
Mesaj de distrugere		Momentul în care un partener implicat în interacțiune încetează să existe
Fragment combinat		Construcții care modelează structuri de control

Notății (2/2)

Nume	Notăție	Descriere
Mesaj sincron	 A sequence diagram showing two participants. The first participant sends a solid horizontal arrow to the second participant. The second participant receives it via a dashed arrow.	Inițiatorul așteaptă un mesaj de răspuns
Mesaj răspuns	 A sequence diagram showing two participants. The second participant sends a dashed arrow back to the first participant, indicating a response to a previous message.	Răspuns la un mesaj sincron
Comunicare asincronă	 A sequence diagram showing two participants. The first participant sends a solid horizontal arrow to the second participant. The second participant receives it via a dashed arrow.	Emitătorul continuă activitatea sa după trimiterea mesajului asincron
Mesaj pierdut	 A sequence diagram showing two participants. The first participant sends a solid horizontal arrow to the second participant. The second participant receives it via a dashed arrow labeled "lost".	Mesaj către un receptor necunoscut
Mesaj găsit	 A sequence diagram showing two participants. The second participant sends a dashed arrow to the first participant labeled "found".	Mesaj de la un emițător necunoscut



Recapitulare

1. Care sunt elementele unei tranziții în diagrama de mașini cu stări?
2. Ce tipuri de activități interne se pot executa în interiorul unei stări?
3. Prin ce se caracterizează activitatea de tip **do**?
4. Pentru ce este folosit nodul decizional?
5. Dacă o stare are o tranziție de ieșire fără niciun eveniment specificat, prin ce mecanism se va produce tranziția?
6. Din ce este formată o stare compusă?
7. Ce pot descrie diagramele de interacțiune?
8. Care sunt dimensiunile diagramei de secvență?
9. Cum se reprezintă un partener al interacțiunii în diagrama de secvență?
10. Ce reprezintă un mesaj sincron? Dar un mesaj de distrugere a unui obiect?
11. Care este rolul fragmentelor combinate?
12. Ce structură de control modeleză fragmentul alt?
13. Explicați diferența dintre fragmentele opt și break.
14. Prin ce diferă diagrama de comunicare de cea de secvență?



Modelarea proceselor de afaceri – limbajul BPMN

Modelarea proceselor de afaceri – limbajul BPMN

Cuprins

- ✓ Introducere
- ✓ Managementul și modelarea proceselor de afaceri
- ✓ Limbajul BPMN
- ✓ Elemente ale limbajului BPMN
- ✓ Obiecte de flux
- ✓ Obiecte de conectare
- ✓ Obiecte de partiționare
- ✓ Date
- ✓ Artefacte
- ✓ Tipuri de diagrame



Introducere - 1

Managementul proceselor de afaceri se bazează pe observația că fiecare produs pe care o companie îl oferă pe piață este **rezultatul mai multor activități** desfășurate în cadrul companiei.

Procesele de afaceri sunt instrumentul cheie pentru **organizarea** acestor activități și pentru **îmbunătățirea înțelegерii relațiilor** dintre acestea.

Tehnologia informației în general și sistemele informatiche în special joacă un rol important în gestiunea proceselor de afaceri, deoarece tot mai multe activități pe care le desfășoară o companie au **suportul unor sisteme informatiche**.

Activitățile unui proces de afaceri pot fi realizate **manual** de către angajații companiei sau **cu ajutorul sistemelor informaticе**. Există, de asemenea, activități ale unui proces de afaceri care pot fi realizare **automat** de sistem, fără nicio implicare umană.

Introducere - 2

În multe companii există un **decalaj** între aspectele organizaționale ale afacerii și tehnologia informațională care este implementată.

Reducerea acestui decalaj între organizație și tehnologie este importantă, deoarece companiile sunt constrânse să ofere clienților lor produse mai bune și specifice.

La nivel organizațional, procesele de afaceri sunt esențiale pentru **întellegerea** modului în care funcționează companiile. Au un rol important în **proiectarea** și **realizarea** sistemelor informatiche **flexibile**.

Acste sisteme informatiche oferă **baza tehnică** pentru crearea **rapidă de noi funcționalități** care realizează produse noi și pentru **adaptarea** funcționalității existente în scopul satisfacerii noilor cerințe ale pieței.

Managementul proceselor de afaceri este influențat de **concepții și tehnologii** din diferite domenii ale administrării afacerilor și informatică.

Procese de afaceri



Un proces de afaceri (engl. *business process*) constă dintr-un set de activități care sunt efectuate în mod coordonat într-un mediu organizațional și tehnic.



Aceste activități realizează în comun un **obiectiv** de afaceri.



Fiecare proces de afaceri este adoptat de o **singură organizație**, dar poate interacționa cu procesele de afaceri efectuate de către **alte** organizații.



Tehnologia informației **influențează** gestiunea proceselor de afaceri, tot mai multe activități desfășurate în cadrul unei organizații **necesită automatizare**, beneficiind astfel de suportul unor sisteme informatiche.

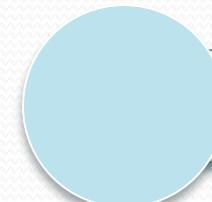
Managementul proceselor de afaceri



Managementul proceselor de afaceri (engl. *business process management*) include concepte, metode, și tehnici necesare pentru a sprijini **proiectarea, administrarea, configurarea, implementarea și analiza proceselor de afaceri.**



Baza managementului proceselor de afaceri este **reprezentarea explicită** a procesele de afaceri cu activitățile lor și constrângerile de execuție dintre acestea.



Odată definite procesele de afaceri, acestea pot fi supuse **analizei, îmbunătățite și implementate.**



Un sistem de gestiune a proceselor de afaceri (engl. *business process management system*) este un sistem software generic care folosește **reprezentări explice** ale proceselor de afaceri pentru a **coordona implementarea** acestora.

Modelarea proceselor de afaceri

Există diferite niveluri ale modelării proceselor:

Hărți de procese – simple fluxuri de lucru ale activităților

Descrieri ale proceselor – fluxuri de lucru extinse cu informații adiționale, dar nu suficiente pentru a defini procesul în detaliu

Modele de procese – fluxuri de activități extinse cu informații suficiente pentru a putea analiza, simula și/sau executa procesul

BPMN suportă toate aceste niveluri

BPMN = Business Process Model and Notation

Limbajul BPMN - caracteristici

Standard creat și întreținut de OMG

Înainte de standardizare, fiecare companie sau producător de instrumente de modelare avea definită **propria metodă** de reprezentare

Este **independent** din punct de vedere tehnologic

Are un limbaj **expresiv** și un vocabular de bază simplu

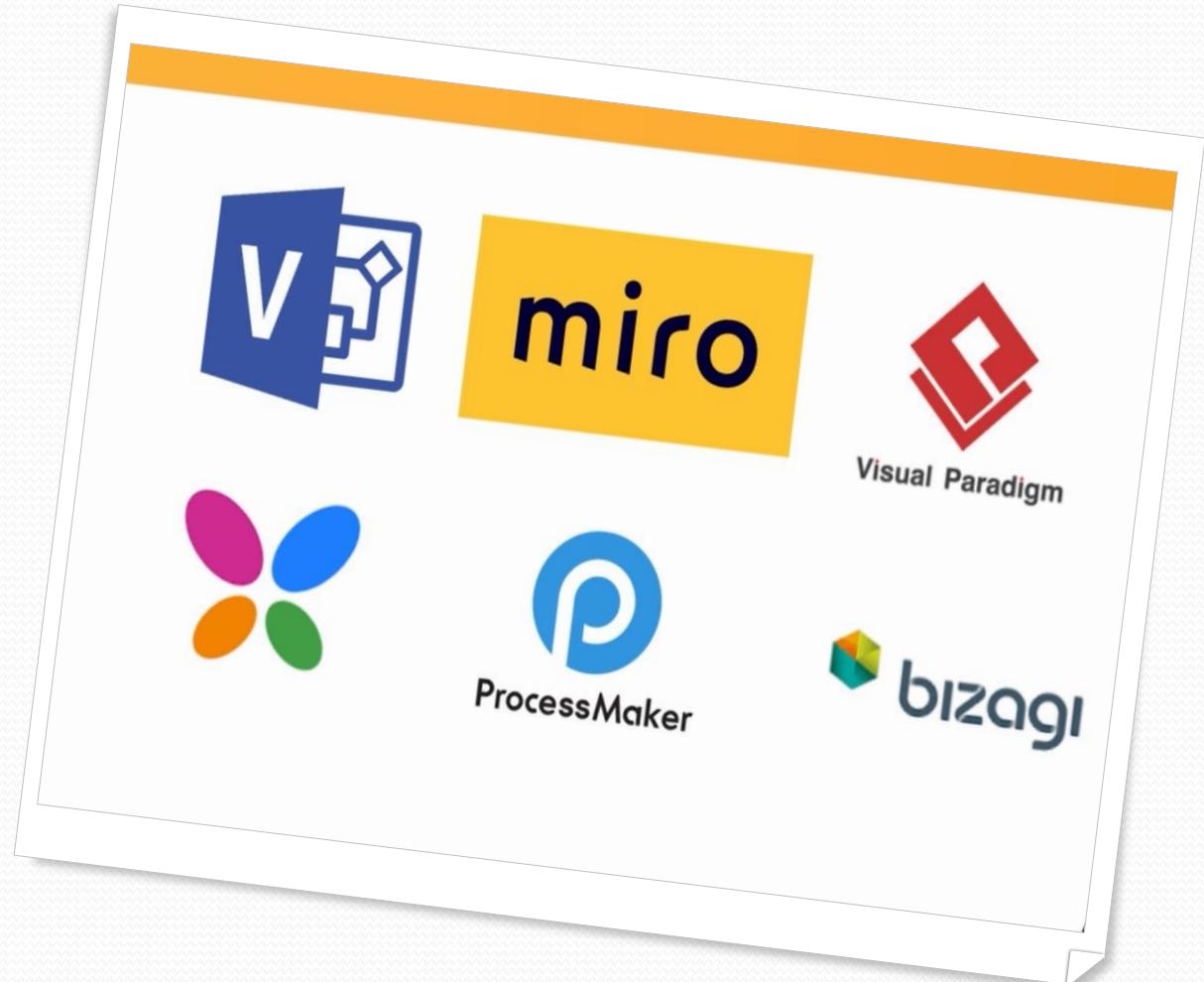
Oferă construcții care definesc şabloane pentru tratarea **excepțiilor**

Permite **îmbunătățirea** proceselor din lumea reală (prin simulare și optimizare)

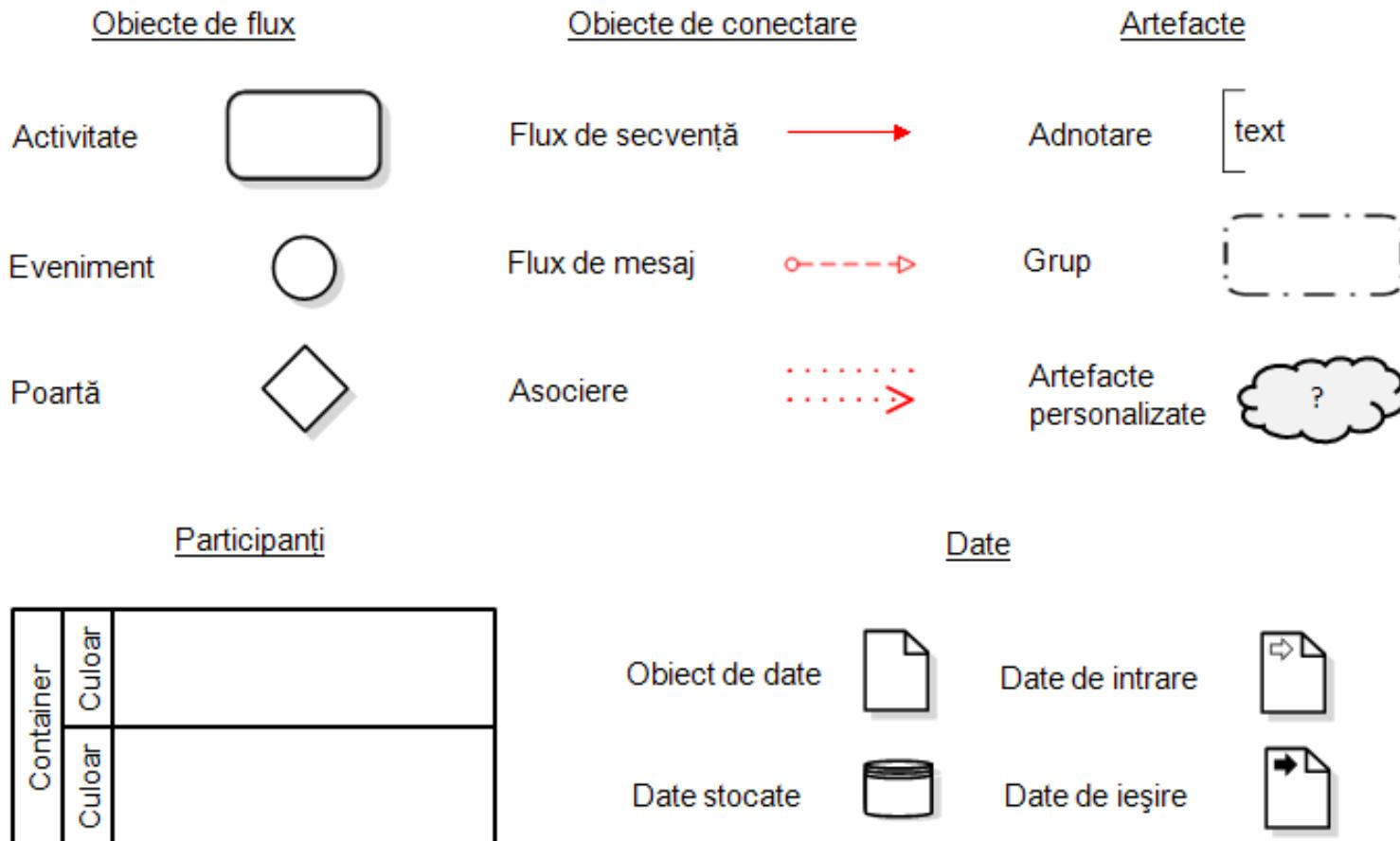
Diferența dintre BPMN și alte limbaje pentru fluxuri de lucru

1. Standard cu **sintaxă și reguli de formare** a modelelor valide
 - ✓ Verificarea sintactică a modelelor se poate face pe parcurs, în timpul creării modelelor sau se poate valida modelul la final ca precondiție pentru simulare/automatizare.
2. Este **ierarhic** și permite **imbricarea** modelelor
3. Pentru fiecare element de modelare, BPMN definesc multe **atribute adiționale** care nu apar în diagramă (spre exemplu, intrări/ieșiri sau listeneri).
 - ✓ Acestea sunt parte a modelului BPMN, dar nu apar în diagramă. Mare parte a acestor atrbute sunt detalii **necesare pentru automatizarea** procesului folosind un instrument dedicat (motor de procese).
 - ✓ Majoritatea modelelor BPMN **nu sunt create**, însă, cu scopul de a fi **executate**.

Instrumente specifice



Elemente ale limbajului BPMN



Elemente ale limbajului BPMN

Obiecte de flux (flow objects) reprezintă elementele de bază ale diagramei de proces. La rândul lor, acestea se pot încadra în una din categoriile: Eveniment (event), Activitate (activity), Poartă sau Ieșire (gateway).

Obiecte de conectare (connectig objects) au rolul de a conecta obiectele de flux între ele sau cu alte tipuri de obiecte. Cele trei tipuri de obiecte de conectare sunt: Flux de secvență (sequence flow), Flux de mesaje (message flow) și Asociere (association).

Obiectele de partiționare (swimlanes) stabilesc subgrafuri în fluxul de proces, cu scopul de a separa logic anumite porțiuni ale acestuia, în funcție de entitățile participante la realizarea procesului. Ele pot fi de două tipuri: Container (pool) și Culoar (lane).

Elemente ale limbajului BPMN

Datele (data) sunt necesare pentru a scoate în evidență datele de care au nevoie activitățile sau care sunt produse de acestea. Datele se pot încadra în patru categorii: Obiect de date (data object), Date de intrare (data input), Date de ieșire (data output) și Date stocate (data store).

Artefactele (artifacts) sunt create cu scopul de a oferi informații adiționale în cadrul unei diagrame. Există două tipuri de artefacte standard: Grupul (group) și respectiv Adnotările textuale (annotation), dar atât limbajul, cât și instrumentele de modelare oferă posibilitatea de a adăuga orice alte artefacte personalizate de utilizator necesare pentru înțelegerea modelului.

1. Obiecte de flux

- Reprezintă elementele grafice principale care definesc comportamentul unui proces.
- Tipuri de obiecte de flux:
 - **Activitate** - termen generic pentru a desemna ceva ce se realizează în cadrul unui proces. Activitățile pot fi atomice (acțiuni) sau non-atomice (compuse).
 - **Eveniment**: ceva ce se întâmplă în timpul unui proces de afaceri. Aceste evenimente afectează fluxul unui model și au, de obicei, o cauză (declanșator) sau un impact (rezultat). Există trei tipuri de evenimente, pornind de la momentul în care acestea afectează fluxul:



- **Poartă**: Elemente de modelare folosite pentru a controla divergența sau convergența unor fluxuri de activități. Sunt considerate elemente de decizie.

Task

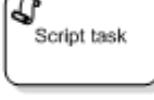
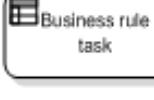
Sub-Process



Gateway

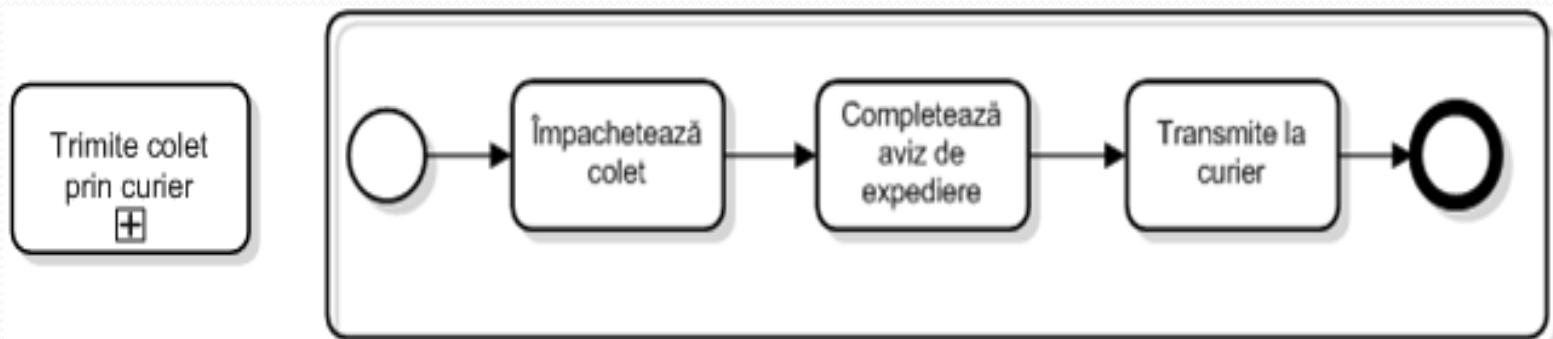
Acțiuni

Acțiunea este o activitate atomică ce nu mai poate fi descompusă pentru a-i descrie comportamentul intern.

Simbol	Descriere
	O <i>acțiune a utilizatorului</i> este frecvent întâlnită în cadrul unui flux de lucru, desemnând faptul că este necesară o intervenție umană în sistemul informatic (spre exemplu, completarea unui formular).
	O <i>acțiune manuală</i> este realizată de o persoană fără ajutorul unui motor de procese sau a unei alte aplicații informatiche (de exemplu, livrarea unui colet).
	O <i>acțiune de tipul serviciu</i> nu necesită intervenție umană și implică apelarea unui anumit tip de serviciu, cum ar fi o aplicație software sau serviciu web.
	O <i>acțiune de tipul trimitere</i> are rolul de a transmite un mesaj unui participant extern procesului, care poate fi un sistem sau o persoană.
	O <i>acțiune de tipul primire</i> are rolul de a aștepta un mesaj de la un participant extern procesului, care poate fi un sistem sau o persoană.
	O <i>acțiune de tipul script</i> este o secvență de cod executată de un motor de procese de afaceri. Analistul sau dezvoltatorul definesc aceste instrucțiuni într-un limbaj pe care motorul de procese știe să îl interpreteze.
	O <i>acțiune de tipul regulă de afaceri</i> oferă procesului un mecanism prin intermediul căruia furnizează date de intrare unui motor pentru reguli de afaceri și obține de la acesta ieșiri sub forma rezultatelor calculelor efectuate (cum ar fi aplicarea de cote diferite de reduceri pentru clienți).

Subprocese

- Sunt sunt activități compuse incluse în interiorul unui proces.
- Pot fi imbricate în mod ierarhic până la orice nivel de detaliere. Este necesar pentru a descrie complet un proces.
- Pot fi reprezentate atât în mod condensat, cât și extins.
- Orice descriere extinsă a unui subprocess trebuie să conțină evenimente de început și de sfârșit pentru care nu se specifică un comportament particular.



Evenimente

- Evenimentele sunt elemente care **nu au durată**.
- Exemple de evenimente:
 - Sosirea unui mesaj, cum ar fi e-mail sau scrisoare
 - Se ajunge la un anumit moment în timp (“începutul lunii”, “ora 10.00”).
 - Trece o anumită perioadă de timp (“două zile”, “timp de coacere complet”)
 - O condiție devine adevărată (“Temperatura externă este de cel puțin 30°C”)
 - Se produce o eroare (“Mesajul nu poate fi livrat”)

Categorii de evenimente

- Se reprezintă ca cercuri cu centrul gol pentru a permite folosirea de marcatori interni pentru a diferenția diferiți declanșatori sau rezultate.
 1. Eveniment de **început** care **recepționează** un mesaj.
 2. Eveniment **intermediar** care **recepționează** un mesaj.
 3. Eveniment **intermediar** care **trimite** un mesaj.
 4. Eveniment **final** care **trimite** un mesaj.
 5. Eveniment de **început** care **recepționează** un mesaj fără a **întrerupe** o altă activitate.
 6. Eveniment **intermediar** care **recepționează** un mesaj fără a **întrerupe** o altă activitate.



1



2



3



4



5



6

Calificatori pentru evenimente- 1

Simbol	Calificator	Semnificație
	Mesaj	Primește sau trasmite mesaje. Exemple: primește cerere de rezervare, trimite formular completat.
	Timp	Este întotdeauna de tipul “Primește” și este folosit pentru a arăta că se așteaptă ca o anumită condiție bazată pe timp să fie evaluată ca adevărată. Exemplu: așteaptă 5 minute, în fiecare zi la ora 8 ⁰⁰ .
	Semnal	Se folosește pentru a recepționa sau transmite semnale către unul sau mai mulți participanți. Exemplu: buget de cheltuieli aprobat, comandă planificată pentru producție.
	Eroare	Este folosit pentru tratarea excepțiilor și poate avea loc doar la sfârșitul unui proces. Exemplu: informații insuficiente.
	Condițional	Procesul este declanșat atunci când o anumită regulă de afaceri este evaluată ca adevărată. Exemplu: prețul unei acțiuni a variat cu 10% față de prețul de la deschiderea bursieră, nivel minim de stoc.
	Escaladare	Desemnează un flux alternativ prin care se încearcă rezolvarea unei probleme. Are avantajul că oferă posibilitatea de a nu întrerupe executarea unei activități. Este folosită cu precădere în sau cu subprocese. Exemplu: livrare întârziată.

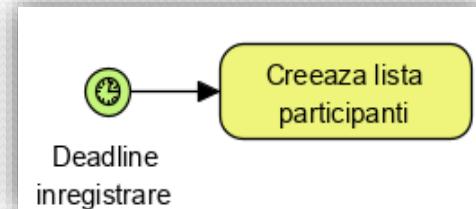
Calificatori pentru evenimente -2

	“Primește”	“Trimite”	Nu îintrerup
Mesaj			
Timp			
Eroare			
Escaladare			
Anulare			
Compensare			
Condițional			
Legătură			
Semnal			
Terminare			
Multiplu			
Multiplu papalel			

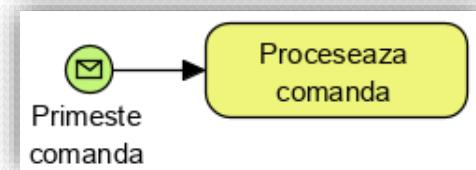
Evenimente de început



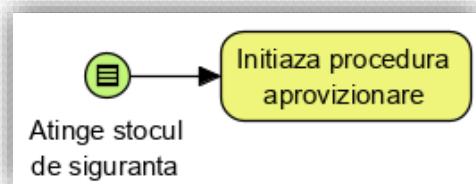
Eveniment de început **necalificat**. Declanșatorul nu este important, nu se cunoaște sau poate fi dedus din context



Eveniment de început care așteaptă să se ajungă la un anumit **moment de timp**



Eveniment de început care așteaptă **primirea unui mesaj**



Eveniment de început care are loc atunci când este **îndeplinită o condiție**

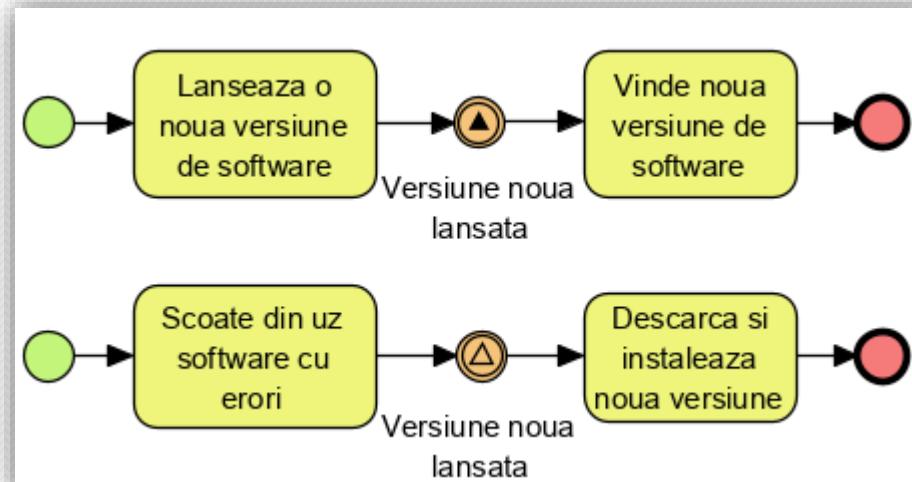
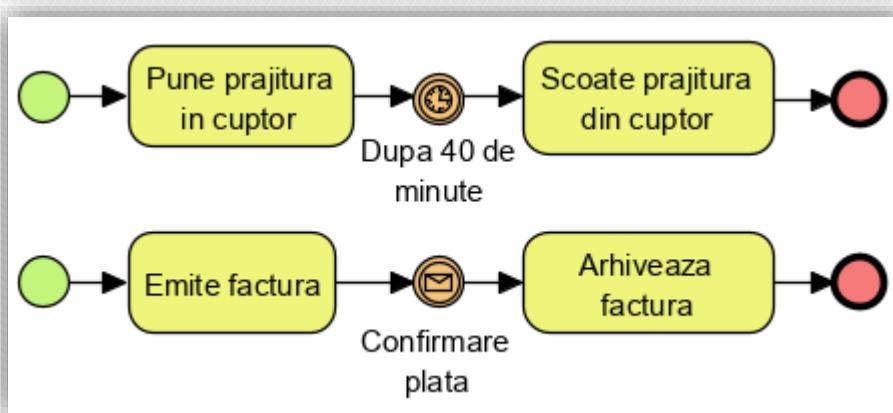


Evenimente de început cu **declanșatori multipli**: SAU – o singură condiție îndeplinită, **ȘI** – toate condițiile îndeplinite

Evenimente intermediare

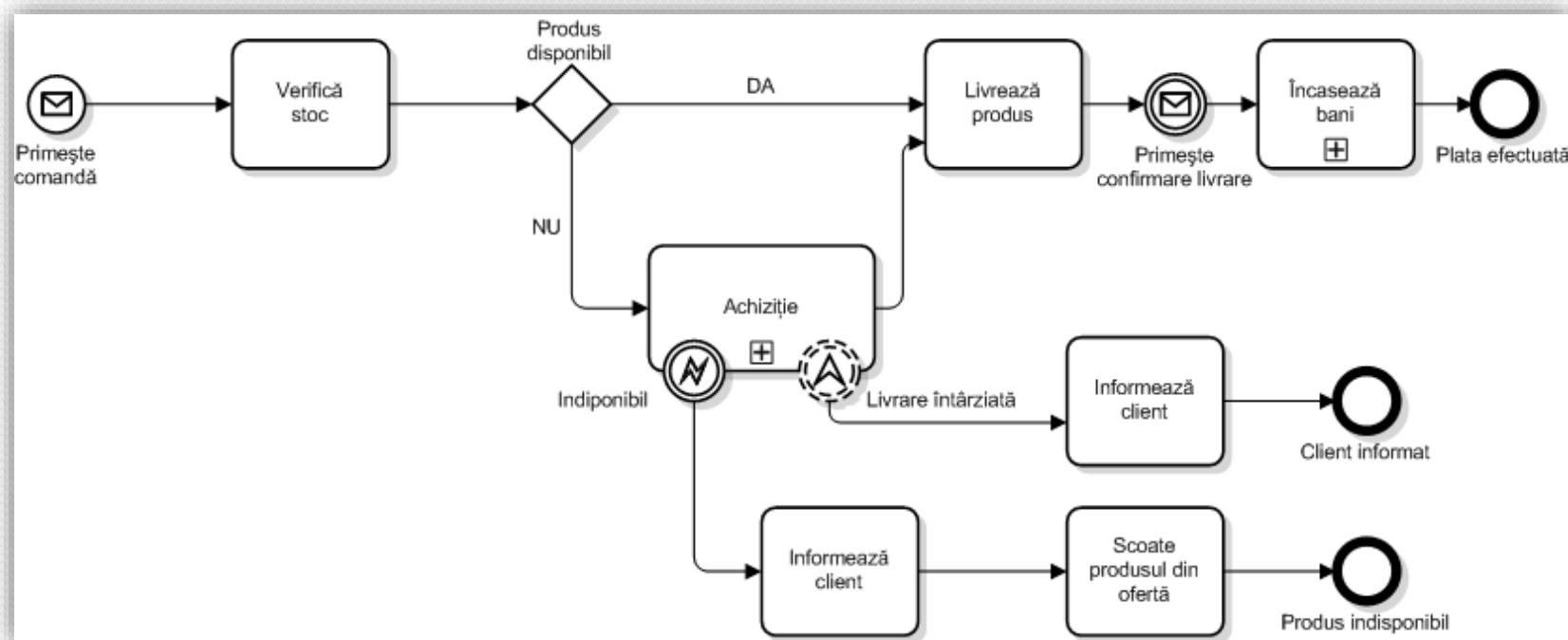
De obicei, evenimentele intermediare sunt modelate atunci când:

1. Un eveniment relevant pentru alți participanți **este declanșat** în cadrul unui proces (de exemplu, este trimis un mesaj sau un semnal).
2. Există o reacție la un eveniment în cadrul unui proces (de exemplu, atunci când este primit un mesaj sau se ajunge la un anumit moment de timp timp).
3. Se dorește modelarea unor excepții sau posibile erori ale sistemului.



Evenimente de tip eroare și escaladare

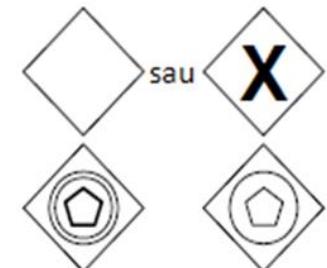
- Evenimentele intermediare de eroare nu pot fi utilizate în fluxuri de secvențe normale, ci doar atașate la **limitele unei activități**.
- În timp ce evenimentele de eroare sunt utilizate în principal pentru **probleme tehnice**, evenimentele de escaladare identifică **probleme la nivelul organizației**, de exemplu, dacă o sarcină nu este finalizată, un obiectiv nu este atins sau nu se realizează un acord necesar.



Porti - categorii

- Controlează fluxul de proces
- Directionează logica sistemului
 - Nu efectuează acțiuni/activități
 - Nu iau decizii, dar pot testa decizii

Exclusivă



Bazată pe evenimente



Paralelă
bazate pe evenimente



Inclusivă



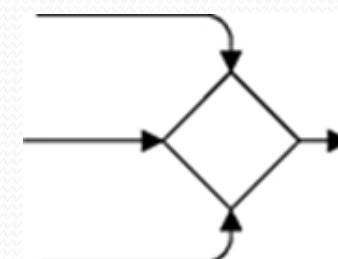
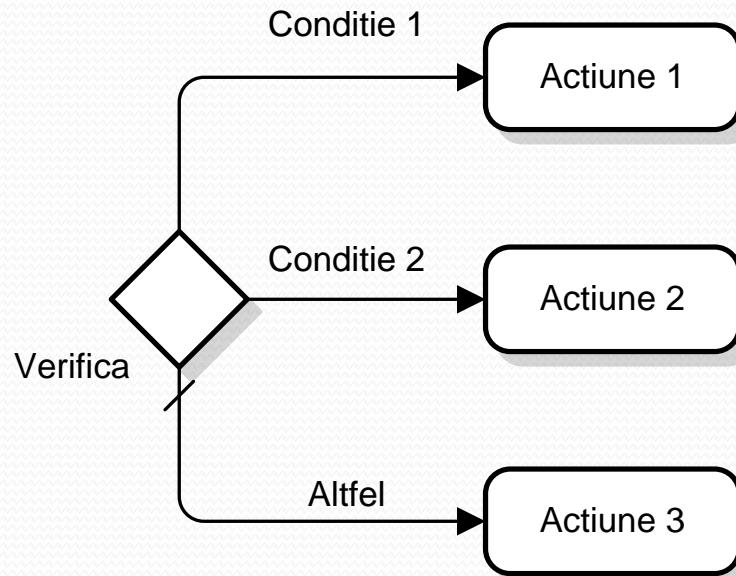
Complexă



Paralelă

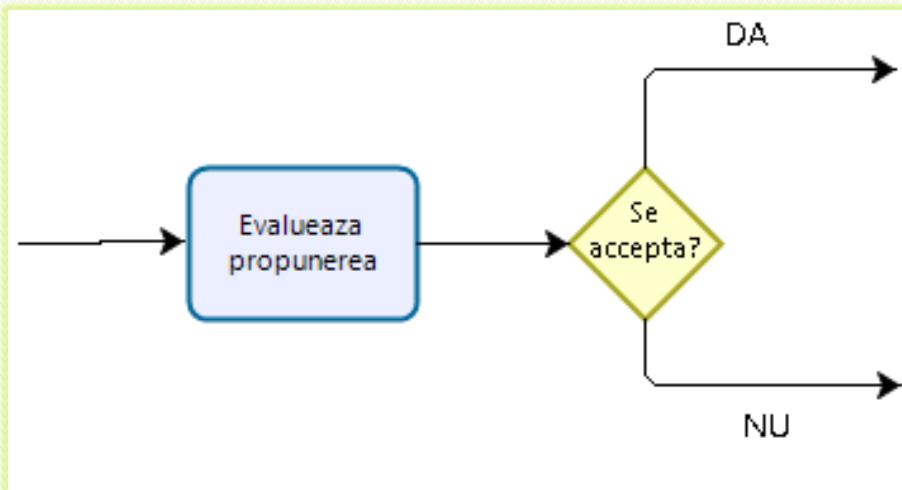
Porti exclusive

- Cunoscute și sub denumirea de decizii, sunt puncte din interiorul unui proces de afaceri unde fluxul de secvențe poate urma una dintre două sau mai multe căi alternative.
- Numai una dintre posibilele căi de ieșire poate fi urmată atunci când procesul este rulat. Corespunde unui „SAU exclusiv” logic.
- Se folosesc pentru divergența sau convergența fluxurilor mutual exclusive.

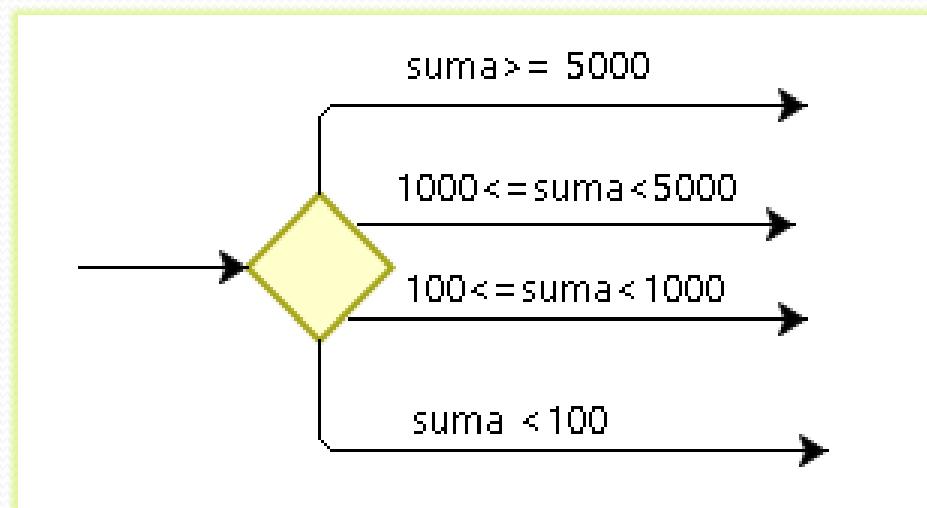


Poartă exclusivă convergentă

Porti exclusive – reprezentare (1)



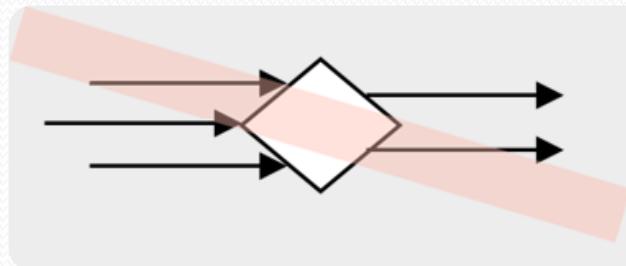
Reprezentare având întrebarea în interiorul portii



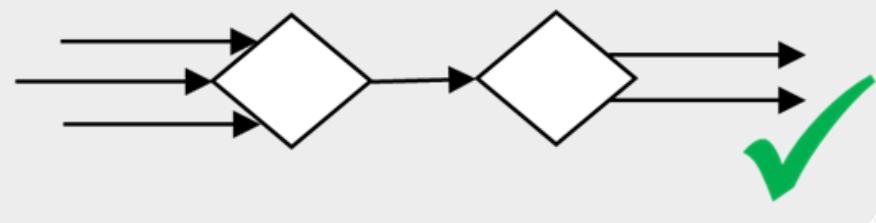
Reprezentare având condiții pe fluxuri

Porti exclusive – reprezentare (2)

Reprezentare folosind perechi de porti

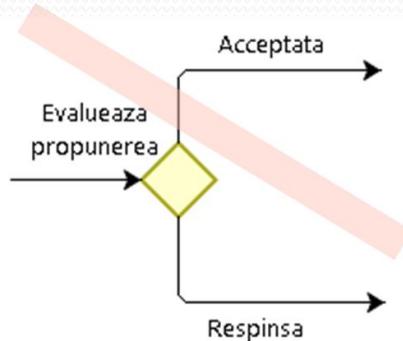


Nu este recomandat

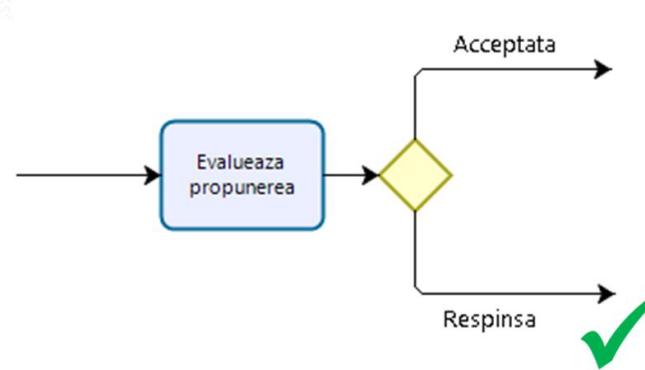


Preferabil

Portile reprezintă doar logică procedurală



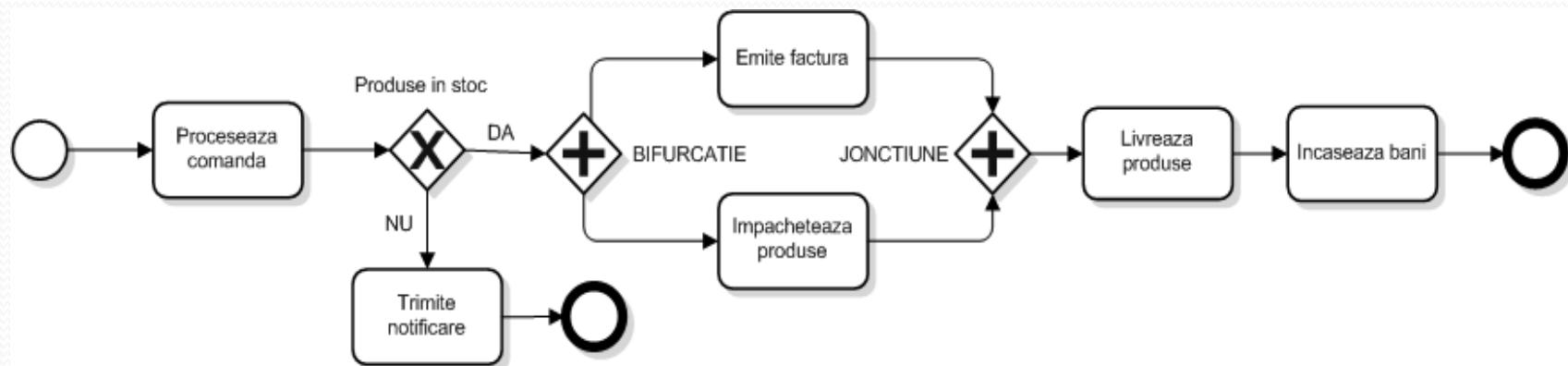
Incorect



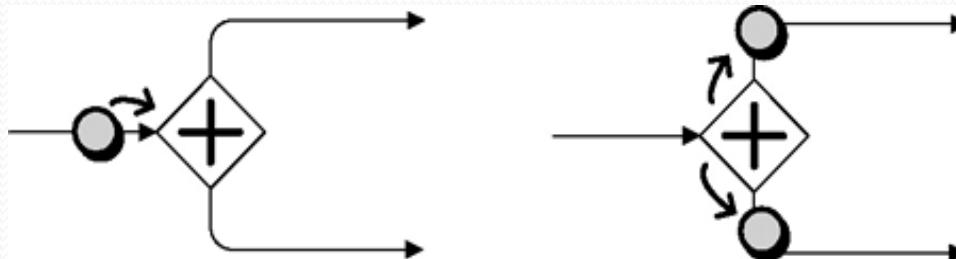
Corect

Porti paralele

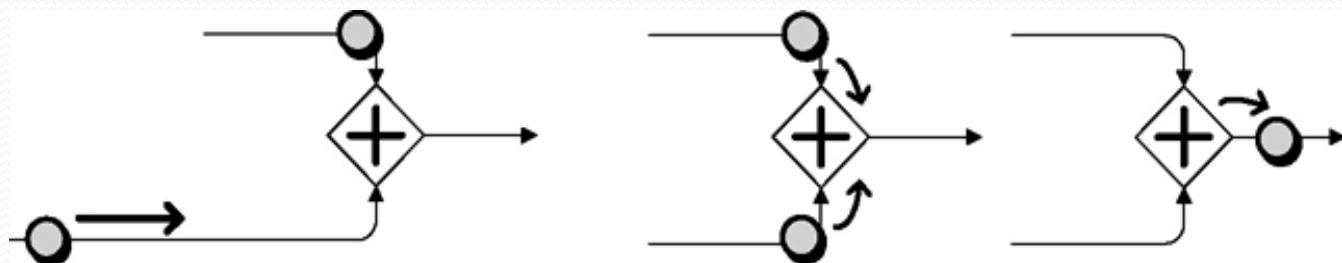
- Crează fluxuri de ieșire paralele fără a verifica nicio condiție care să ducă la declanșarea acestora.
- Sunt folosite pentru a sincroniza (combina) fluxuri paralele sau pentru a desemna începutul unor fluxuri paralele.
- În acest fel se reprezintă executarea activităților concurente. Nu este specificată o ordine a execuției activităților, acestea pot fi executate și simultan.
- Corespunde unui „SI” logic.



Logica portilor paralele



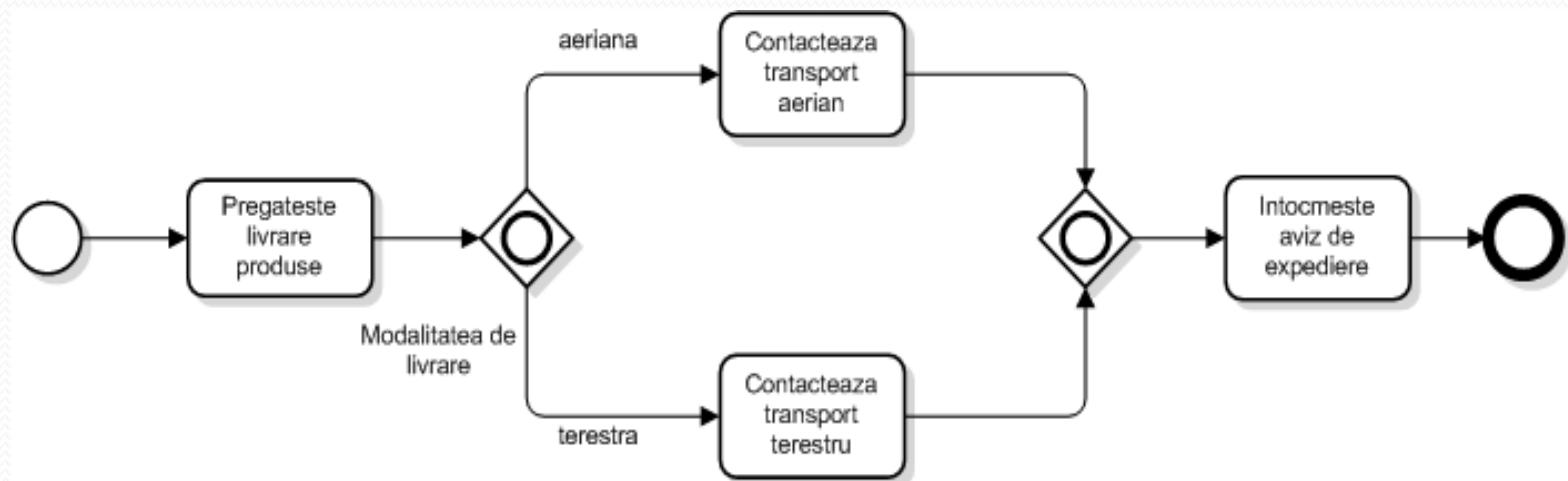
O poartă paralelă divergentă duplică jetonul



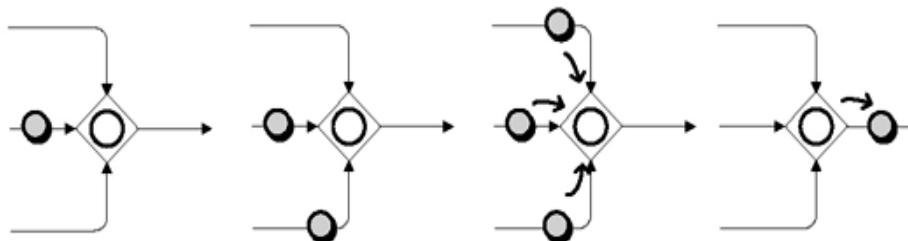
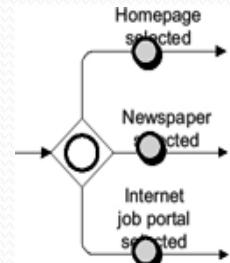
O poartă paralelă convergentă aşteaptă toate jetoanele care sosesc și le îmbină într-un singur jeton

Porți inclusive

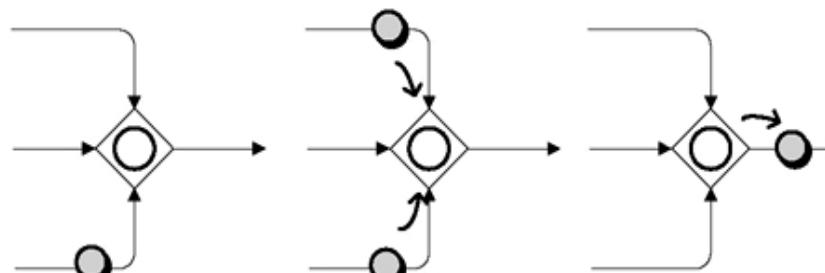
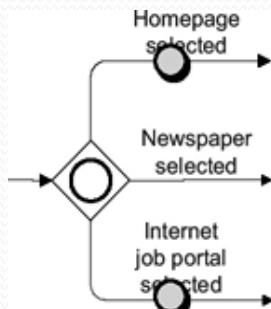
- Porțile inclusive pot declanșa mai mult de un rezultat, deci pot avea mai multe fluxuri de ieșire.
- Toate condițiile de ieșire sunt evaluate indiferent dacă există deja unul sau mai multe fluxuri de ieșire ale căror condiții au fost evaluate anterior ca fiind adevărate.
- În cadrul unui model acestea sunt urmate, de obicei, de poarta inclusivă de îmbinare corespunzătoare.
- Corespunde unui „SAU inclusiv” logic.



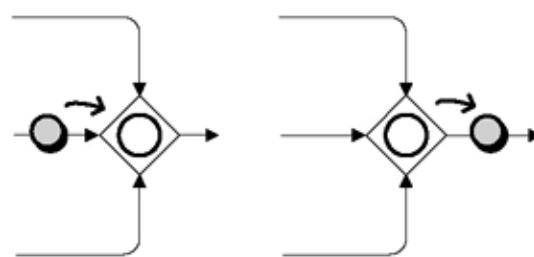
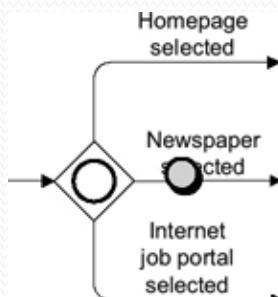
Logica porților inclusive



Toate variantele selectate



Două variante selectate

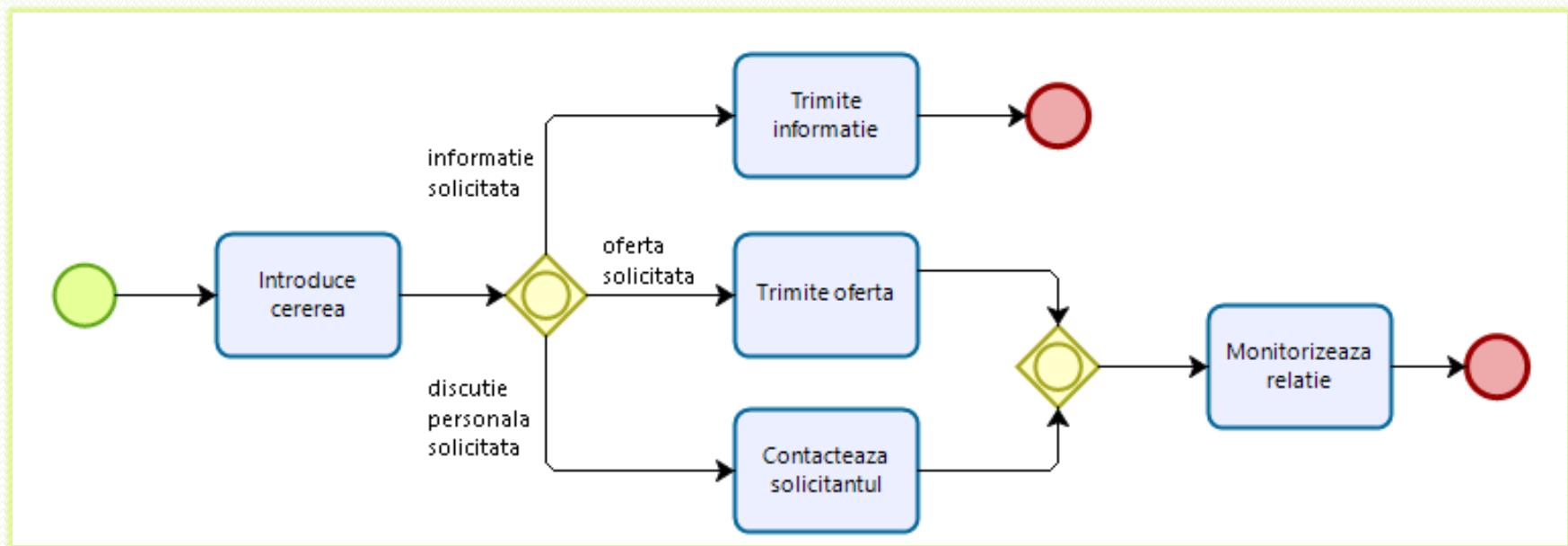


O variantă selectată

O poartă inclusivă știe întotdeauna câte și ce fluxuri de secvențe au fost selectate. Deci, știe ce jetoane sunt așteptate de poarta convergentă înainte de a putea îmbina jetoanele.

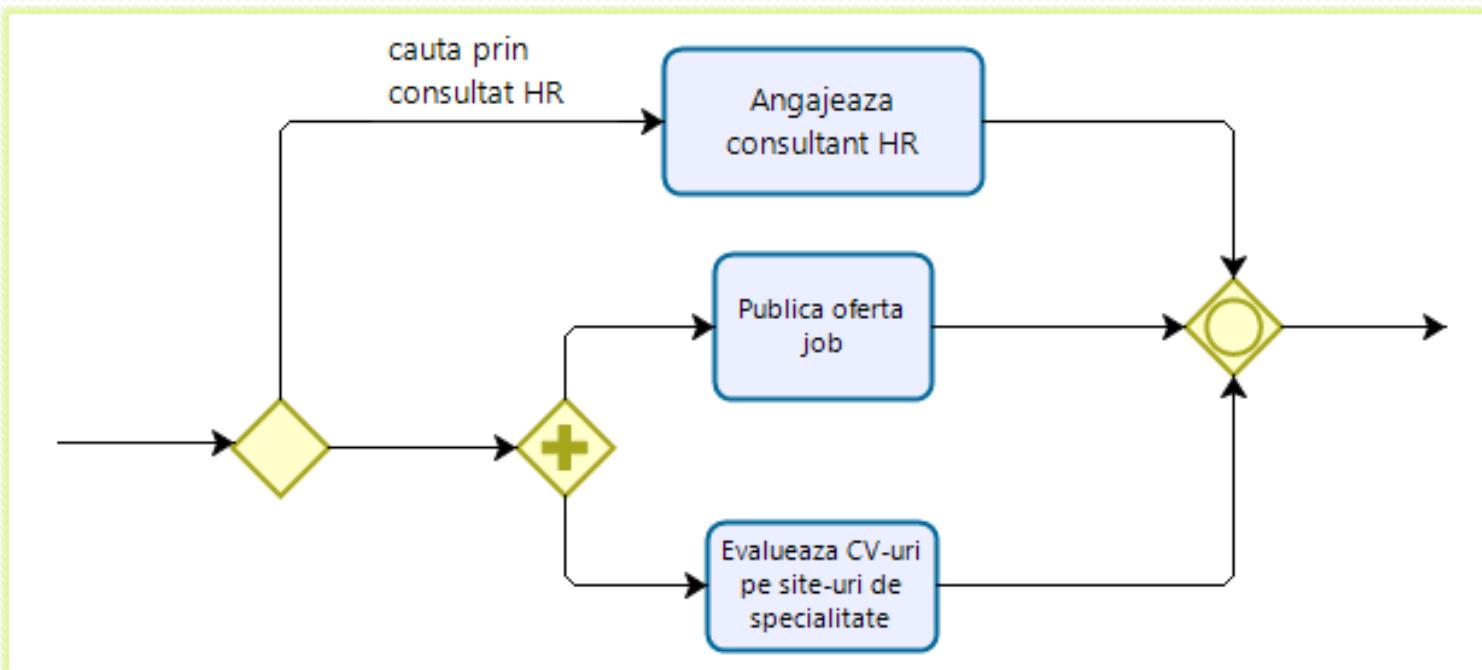


Porti inclusive – cazuri particular (1)



Unul dintre fluxuri nu ajunge la poarta convergentă

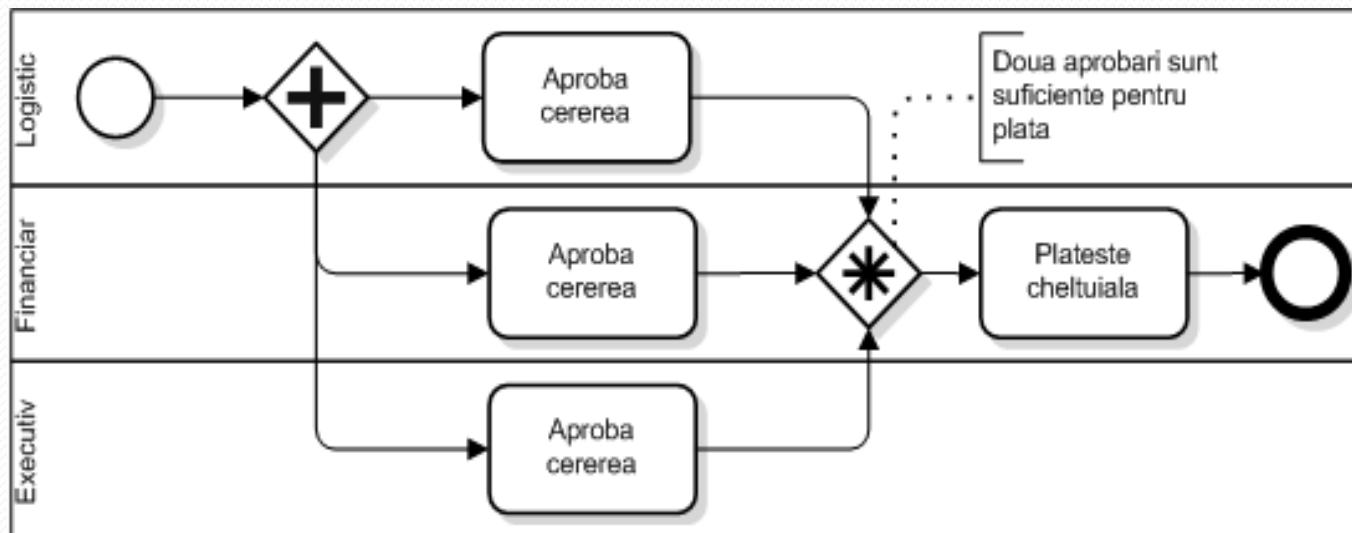
Porti inclusive – cazuri particular (2)



Convergența fluxurilor provenind de la o combinație de porti de tipuri deferite

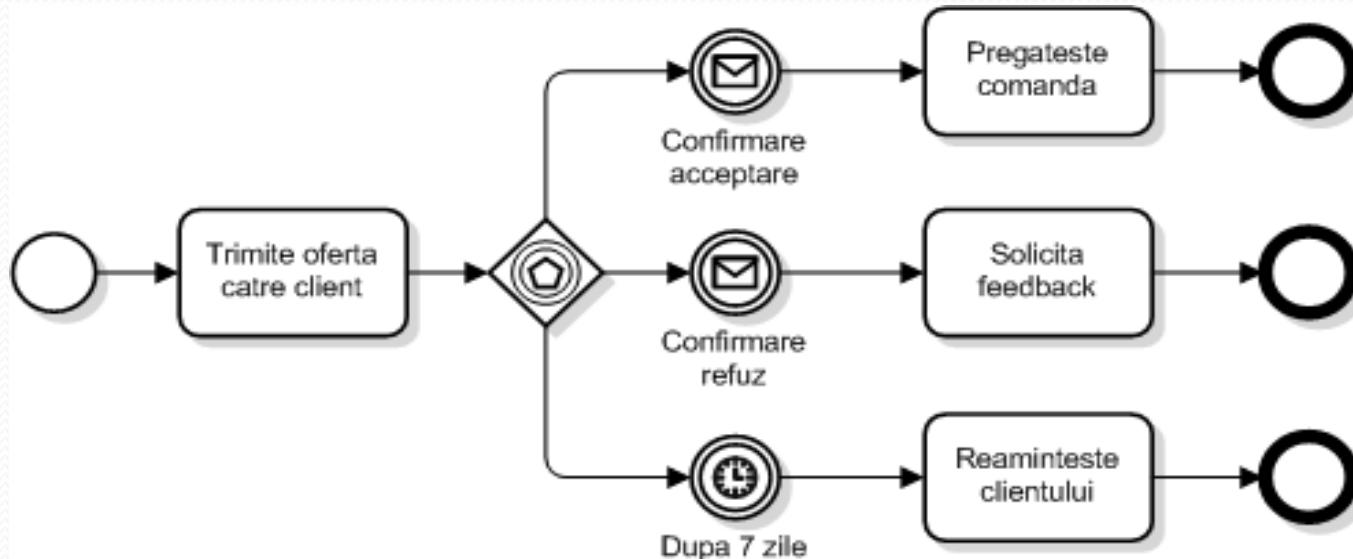
Porti complexe

- Se folosesc atunci când este necesară modelarea unui comportament care presupune condiții de sincronizare care nu pot fi descrise prin intermediul mecanismelor prezentate anterior.
- Pot avea asociate oricără reguli arbitrare definite de utilizator prin care să se specifică modul în care va fi tratată sincronizarea sau divizarea fluxurilor de secvențe.



Porti bazate pe evenimente

- Reprezintă un punct de ramificație al procesului unde fluxurile de ieșire se bazează pe producerea unor evenimente și nu pe evaluarea unor expresii folosind date, aşa cum se întâmplă în cazul portilor exclusive și inclusive.
- Un eveniment specific care constă, de obicei, în primirea unui mesaj ce determină calea care va trebui urmată.
- Decizia este luată de către un alt participant, pe baza unor date care nu sunt accesibile procesului analizat.



2. Obiecte de conectare

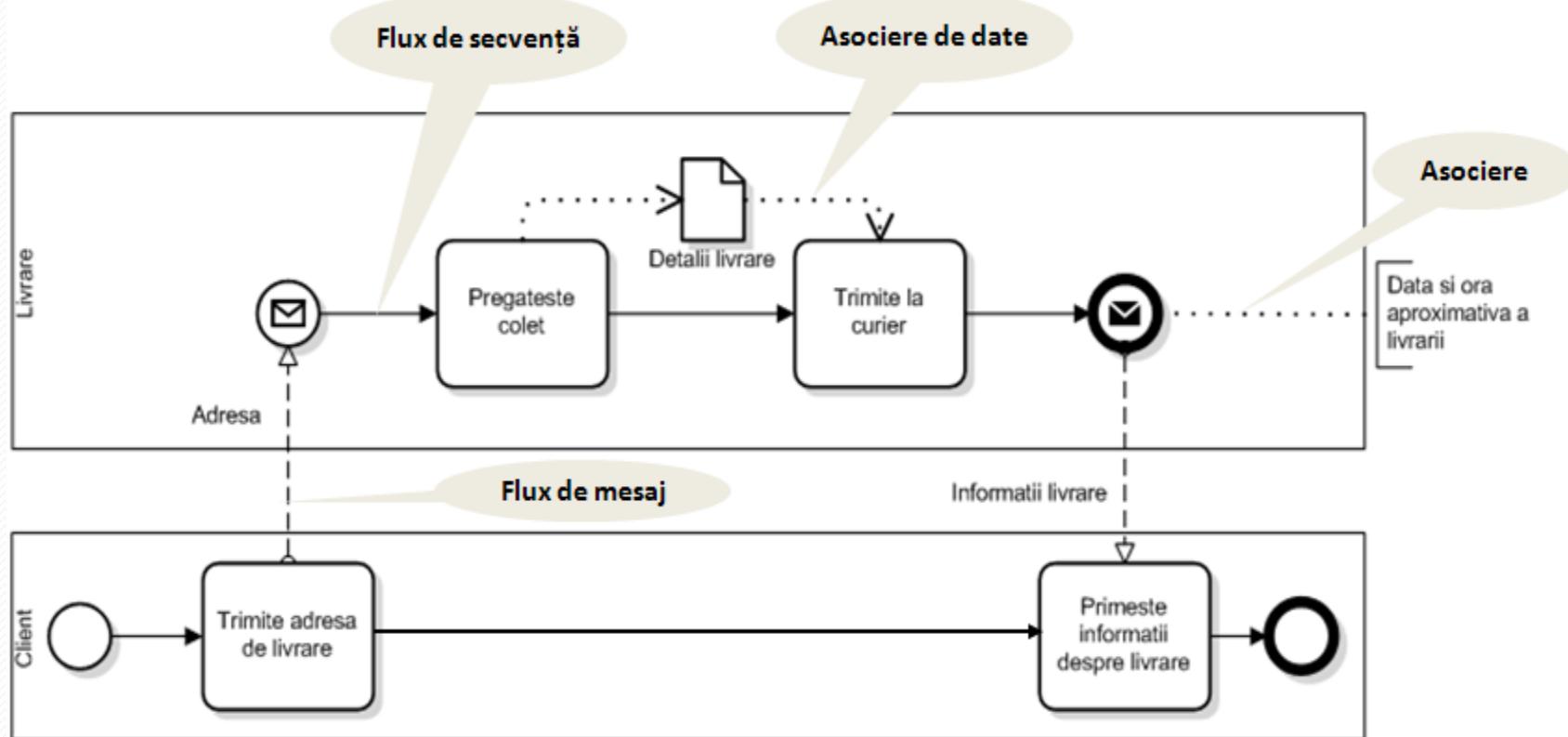
- **Un flux de secvență** este utilizat pentru a descrie ordinea elementelor din flux în modelele de proces și coregrafie.
- **Un flux de mesaj** are rolul de a arăta fluxul de mesaje între doi participanți care sunt capabili să trimită și să primească mesaje.
- **O asociere de date** este folosită pentru a arăta fluxul de informații dintre activitățile unui proces de afacere.
- **O asociere** leagă artefactele cu alte elemente grafice ale BPMN.

Flux de secvență 

Flux de mesaj 

Asociere 

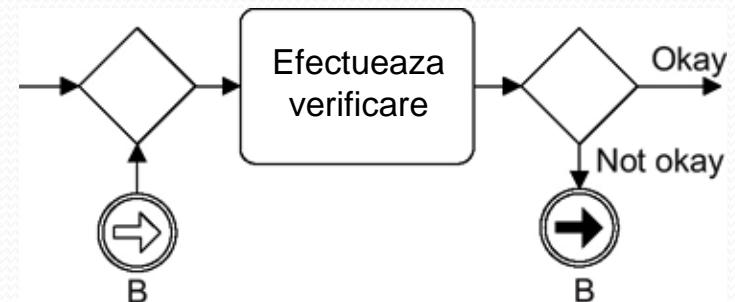
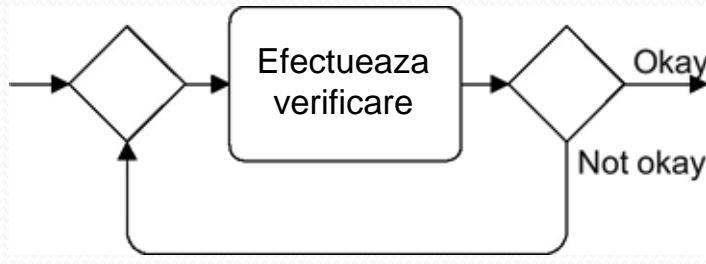
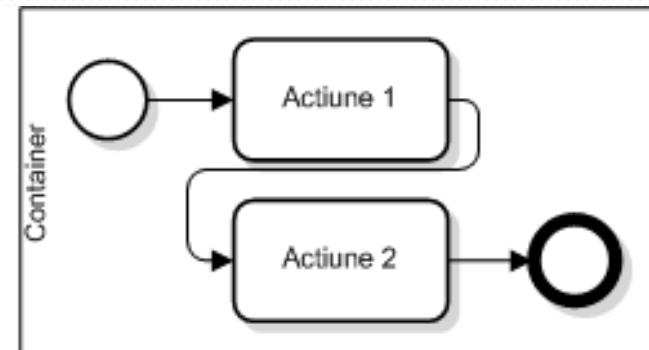
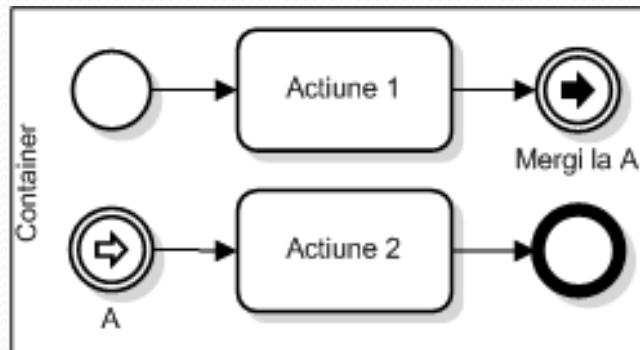
Exemple de obiectele de conectare



Fluxuri de secvență

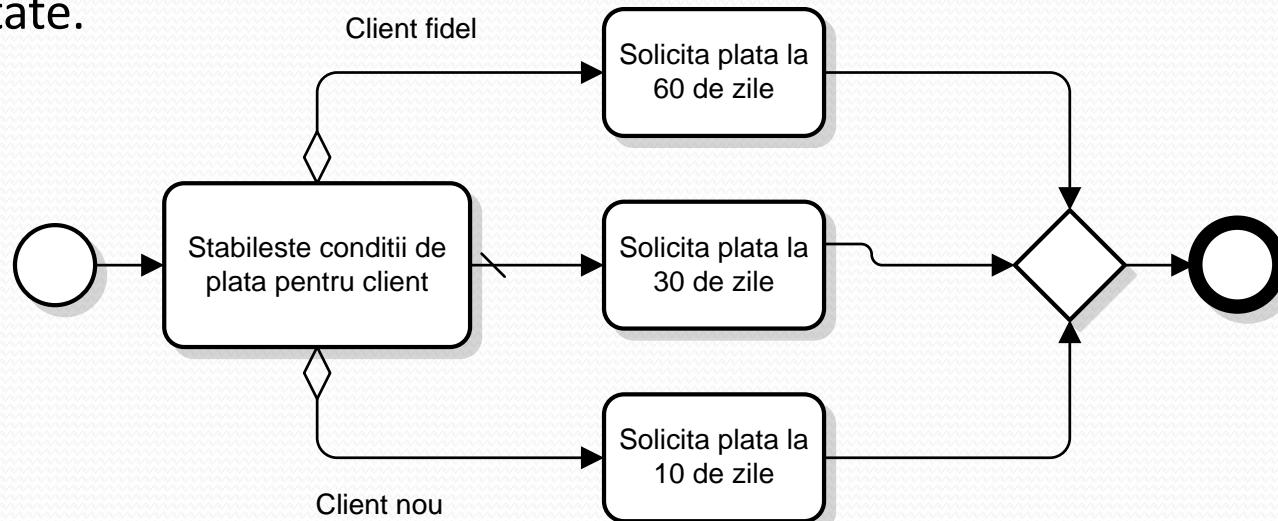
- Pot conecta următoarele tipuri de elemente: evenimente (de început, intermediare și de sfârșit), acțiuni, subprocese și porți,
- Limite ale unui flux de secvență:
 - nu poate reprezenta o intrare pentru un eveniment de început;
 - nu poate reprezenta o ieșire pentru un eveniment de sfârșit;
 - nu poate conecta în mod direct o acțiune a unui proces cu o acțiune a unui subproces, legătura trebuind realizată în mod corect între acțiune și subproces;
 - sunt permise numai în interiorul unui container, pentru interacțiunile dintre containere trebuie utilizate fluxurile de mesaj;
 - nu pot fi utilizate pentru a conecta artefacte la alte elemente ale modelului, în acest caz fiind folosite asocierile;
 - pot fi substituite prin evenimente intermediare de legătură, cu specificația că ambele evenimente intermediare de legătură trebuie să aparțină aceluiași container.

Utilizarea evenimentelor de legătură – exemple

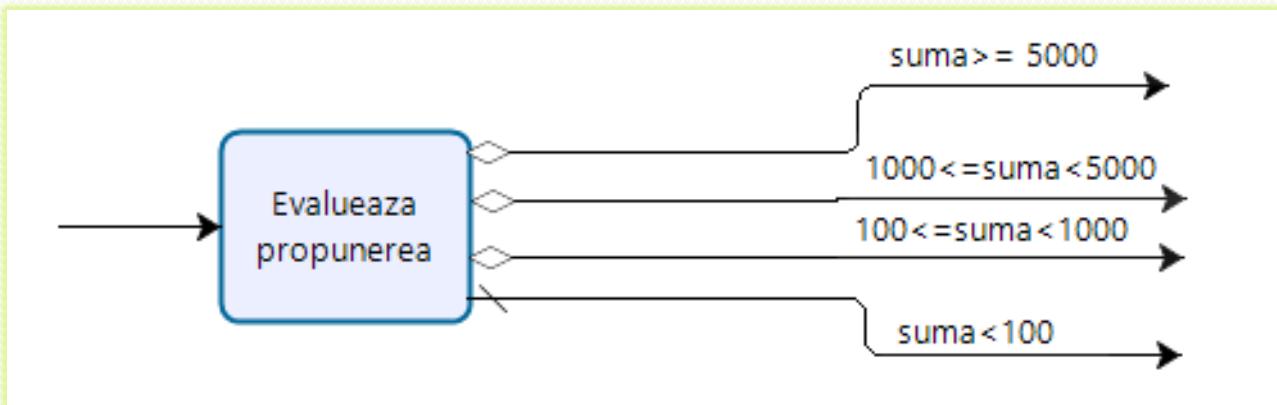
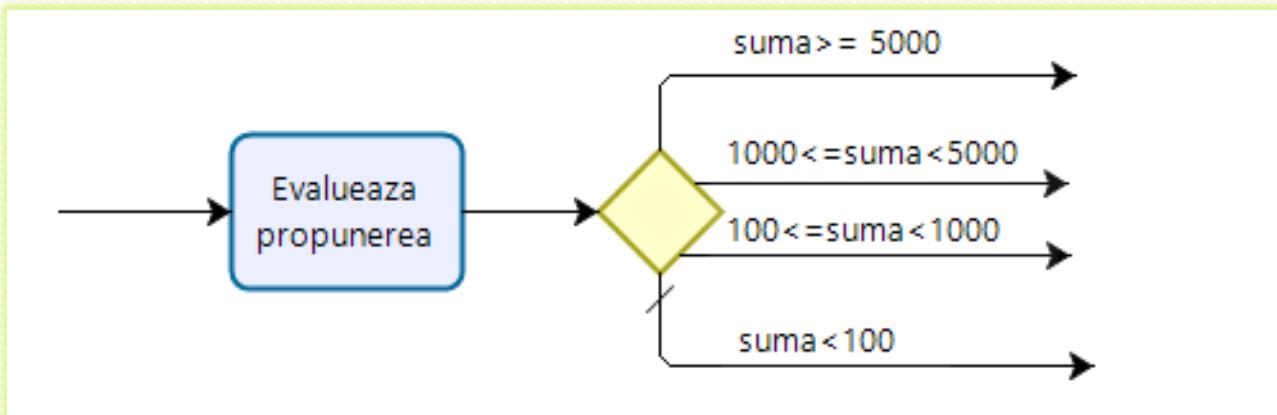


Fluxurile de secvență condiționale

- Atunci când conectează o poartă inclusivă sau exclusivă sau o activitate, un flux de secvență poate defini o condiție și atunci va purta denumirea de flux de secvență condițional. Semnul distinctiv este rombul.
- Există și fluxuri de secvență implicate, pentru situațiile în care nicio altă condiție nu este îndeplinită.
- La folosirea fluxurilor de secvență condiționale trebuie să se aibă întotdeauna în vedere ca mulțimea condițiilor reprezentate de fluxurile de ieșire să conducă la un rezultat valid de fiecare dată când se realizează o activitate.



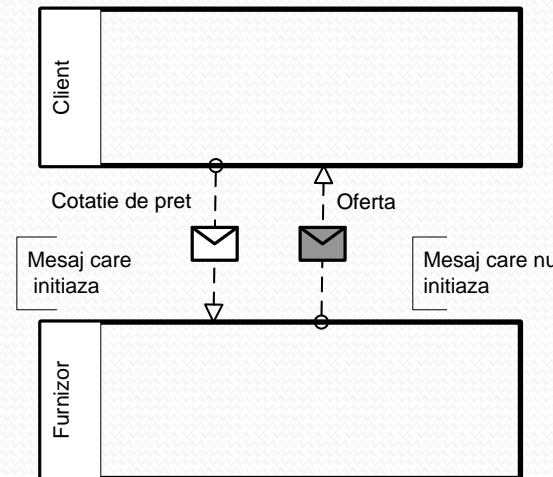
Divergența fluxurilor exclusive fără a utiliza porti



Flux de secvență implicit la nivel de poartă și de activitate

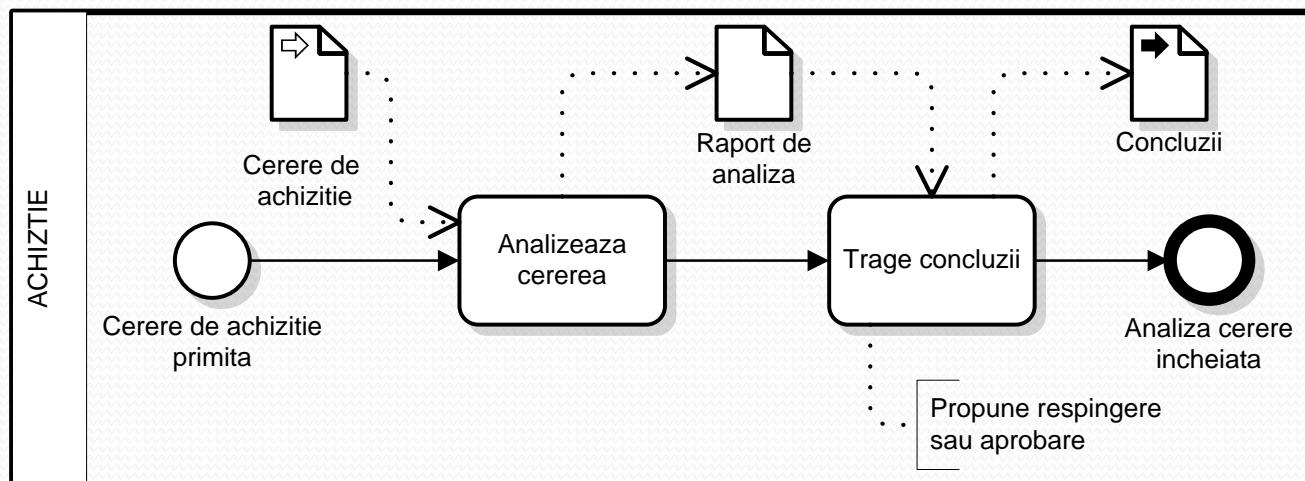
Fluxuri de mesaj

- Un flux de mesaj este folosit pentru a reprezenta transmiterea de mesaje între doi participanți care sunt pregătiți să trimită și să primească aceste mesaje. În BPMN, două containere separate din cadrul unei diagrame de colaborare vor reprezenta cei doi participanți.
- Optional, fluxurile de mesaje pot fi extinse cu un obiect de tip mesaj (*message object*), care va fi legat de fluxul de mesaj sau suprapus peste acesta. Obiectul de tip mesaj descrie în mod explicit conținutul comunicației între cei doi participanți.



Asocieri de date

- Pentru a reprezenta fluxurile de date din cadrul unui proces, BPMN folosește ca și notație asocierile de date, care este o asociere direcțională. Asocierile de date sunt folosite pentru a transfera date între procese sau acțiuni.
- Asocierile de date nu produc nici un efect asupra fluxului de acțiuni din cadrul procesului, rolul lor fiind acela de a arăta care este necesarul de date pentru un anumit proces sau acțiune, precum și care sunt datele pe care acestea le produc sub formă de rezultate.



3. Obiecte de partitioнare

- Reprezintă un mecanism de organizare a activitătilor în categorii vizuale separate în scopul evidențierii diferitelor capacitați funcționale sau responsabilități.
 - Container (Pool)**: reprezintă un participant în proces. Implică unități organizaționale sau participanți separați fizic.
 - Culoar (Lane)**: este folosit pentru a organiza și a împărți activitățile. Sunt plasate în interiorul unui container și pot fi imbricate.

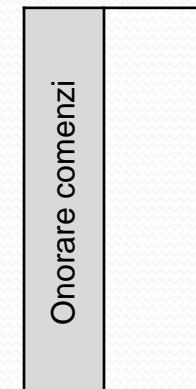
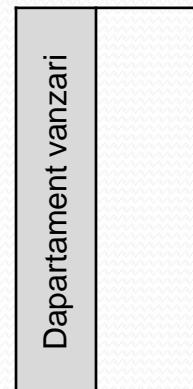
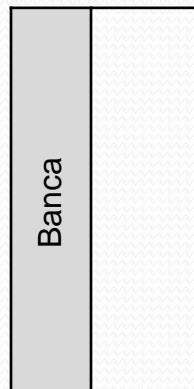


Participanți

- Elementul de tip **participant** constituie o entitate identificată la nivelul modelului de afacere, care:
 - **execută** sau **are anumite responsabilități** în executarea activităților din cadrul unui proces;
 - joacă rolul de **participant în cadrul unei colaborări**.
- Din perspectiva limbajului BPMN, un participant este reprezentat vizual sub forma unui container (*pool*)
- BPMN face distincție între două niveluri de participare:
 - **unitatea organizațională**, care reprezintă grupul de interes intern sau extern organizației, precum compania sau departamentul;
 - **rolul** asociat execuției unei activități, cum ar fi client, furnizor, producător etc.

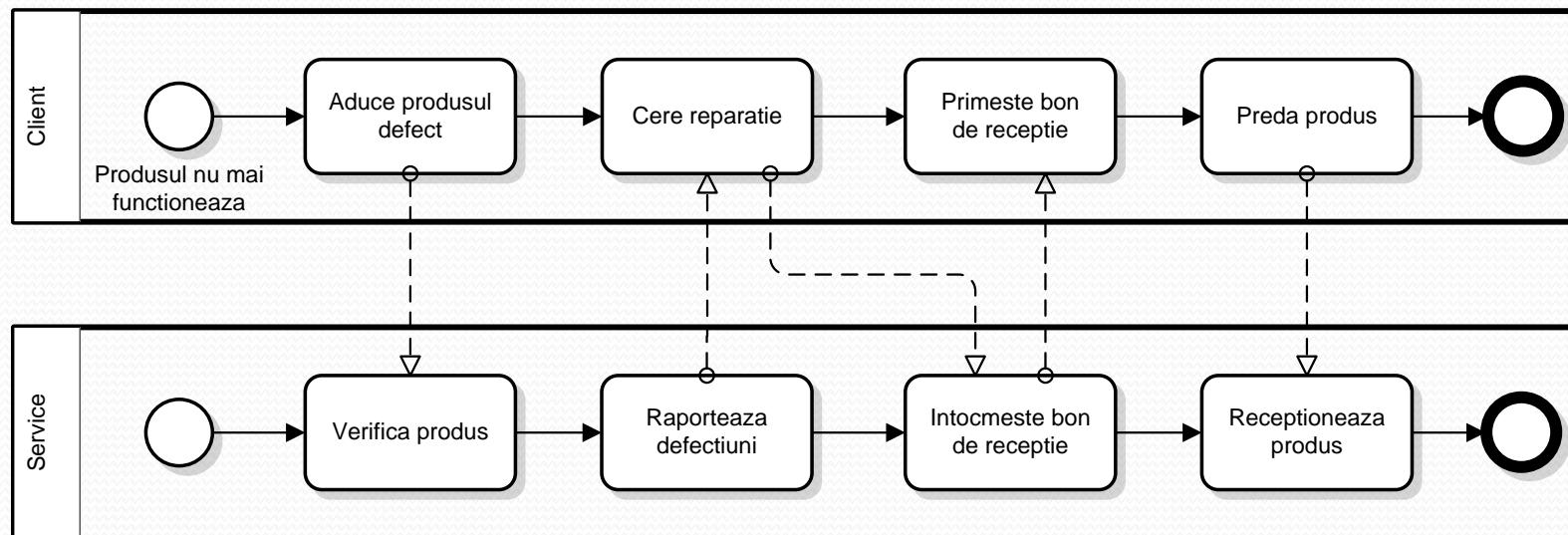
Containere

- Un container poate fi denumit după **numele unui participant** sau după **numele procesului** în sine
- Reprezentarea unui container în jurul procesului este **optională** în diagrama de proces
- Chiar dacă nu există container într-o diagramă de proces, acesta, de fapt, există, fiind **implicit**
- Se recomandă, însă, includerea containerelor deoarece **ofere claritate și ne permit să denumim procesul**



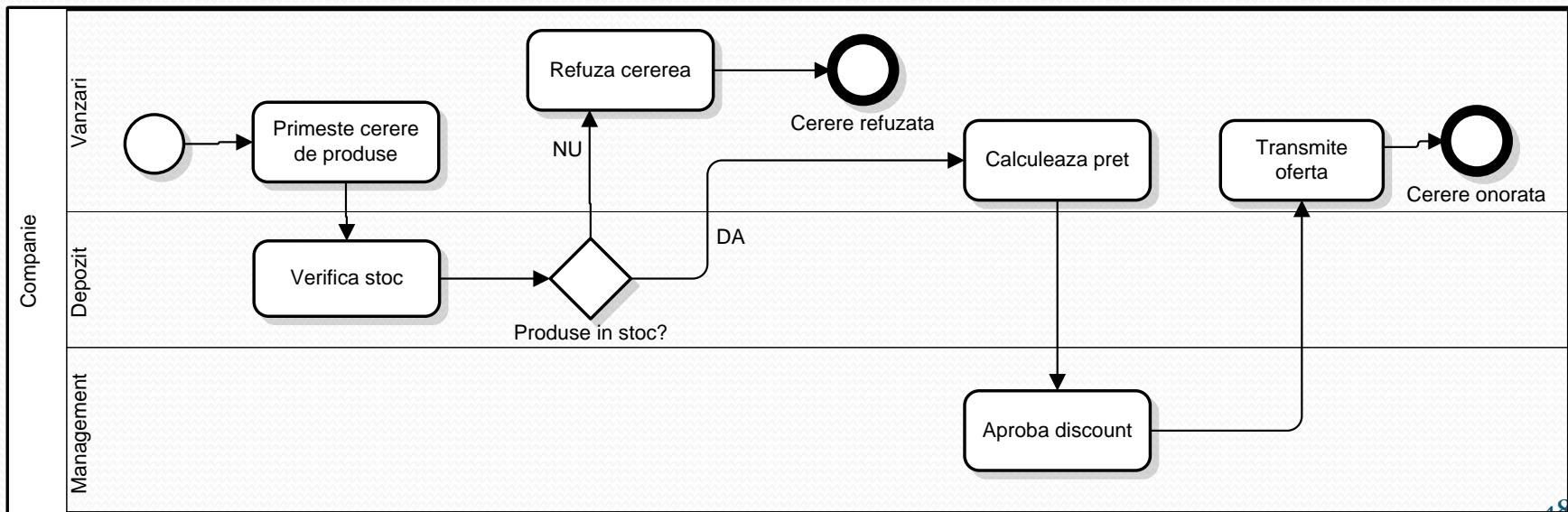
Fluxuri de secvență și de mesaj

- Un container încapsulează secvența de activități a unui proces, ceea ce înseamnă că fluxurile de secvență nu pot traversa granițele unui container.
- Numele containerului nu este obligatoriu să semnifice o unitate organizațională, acesta poate să desemneze și numele procesului în sine, cum ar fi “Recepție produse” sau “Solicitare reparație”.



Partitionarea unui container prin culoare

- Culoarele ajută la identificarea responsabilităților în cadrul unui proces de afaceri.
- Pot reprezenta activități tehnologice sau umane.
- Culoarele se aplică doar activităților, nu și porților sau evenimentelor.
- Fluxul de secvență poate traversa culoarele pentru a duce la îndeplinire activitățile specifice unui proces.

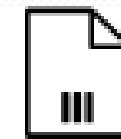
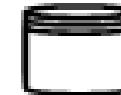
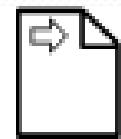
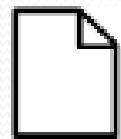


4. Date

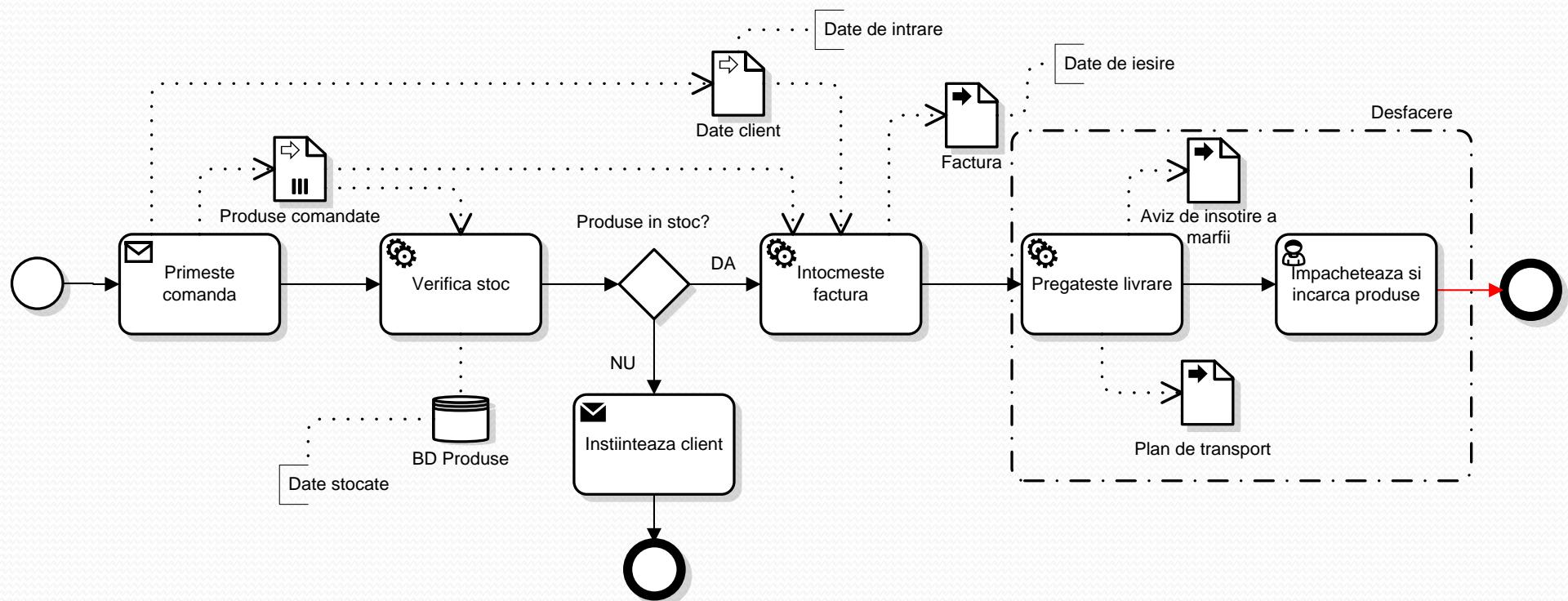
- Când se execută un proces, acesta **folosește și creează** date, informații, fișiere, documente etc.
- Un flux de secvențe este adesea **însorit de transfer de date**.
- De asemenea, unul din scopurile principale ale fluxului de mesaje este **schimbul de date**.
- Datele sunt mecanisme prin care sunt evidențiate datele **necesare sau produse de activități**. Sunt conectate la alte elemente prin asocieri de date.

Categorii de date

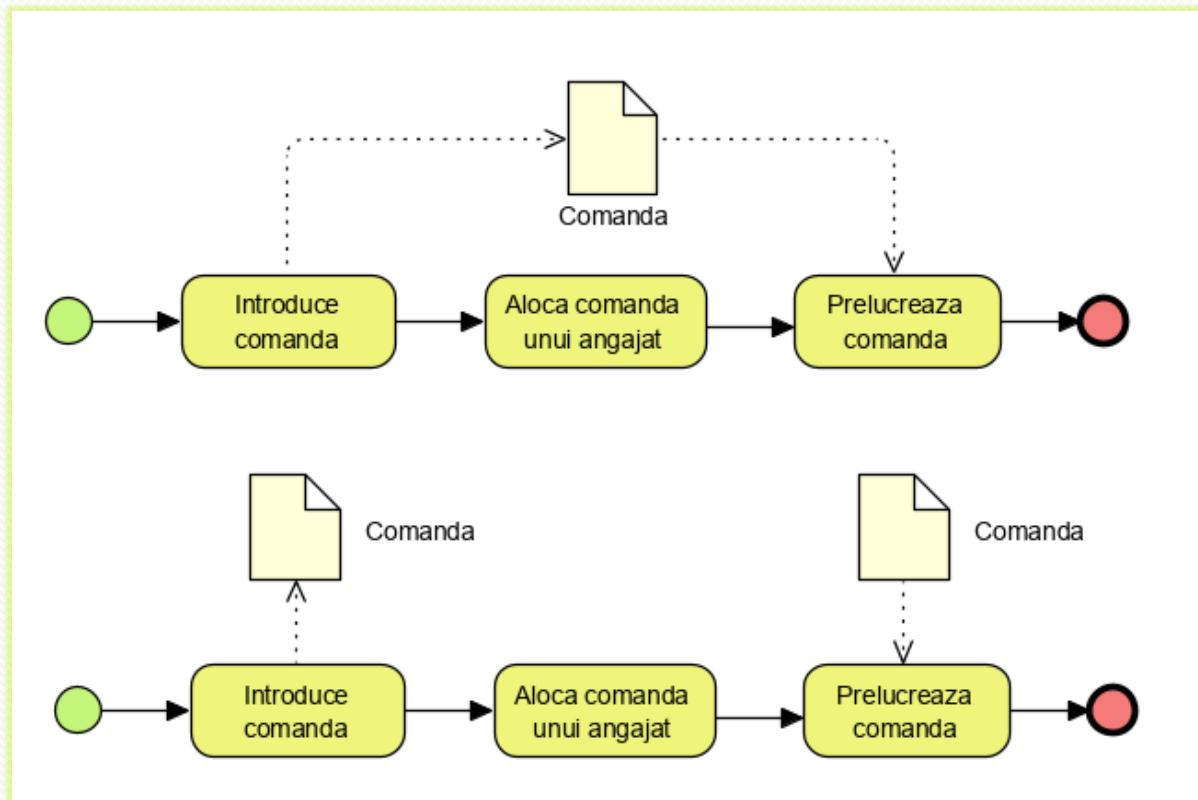
- **Obiect de date** - reprezintă informații care se transmit în interiorul procesului, cum ar fi documente comerciale, e-mailuri sau scrisori. Pot fi fizice sau electronice. Sunt frecvent utilizate pentru modelarea datelor în BPMN.
- **Intrare date** - o intrare externă pentru întregul proces; un fel de parametru de intrare.
- **Date de ieșire** - rezultatul datelor întregului proces; un fel de parametru de ieșire.
- **Colecție de date** - reprezintă o colecție de informații, de exemplu, o listă de articole comandate. Se aplică celor trei categorii anterioare
- **Date stocate** – element în care în procesul poate citi sau scrie date; de exemplu, o bază de date sau o colecție de dosare cu oferte. Persistă dincolo de durata de viață a instanței procesului.



Categorii de date - exemplu



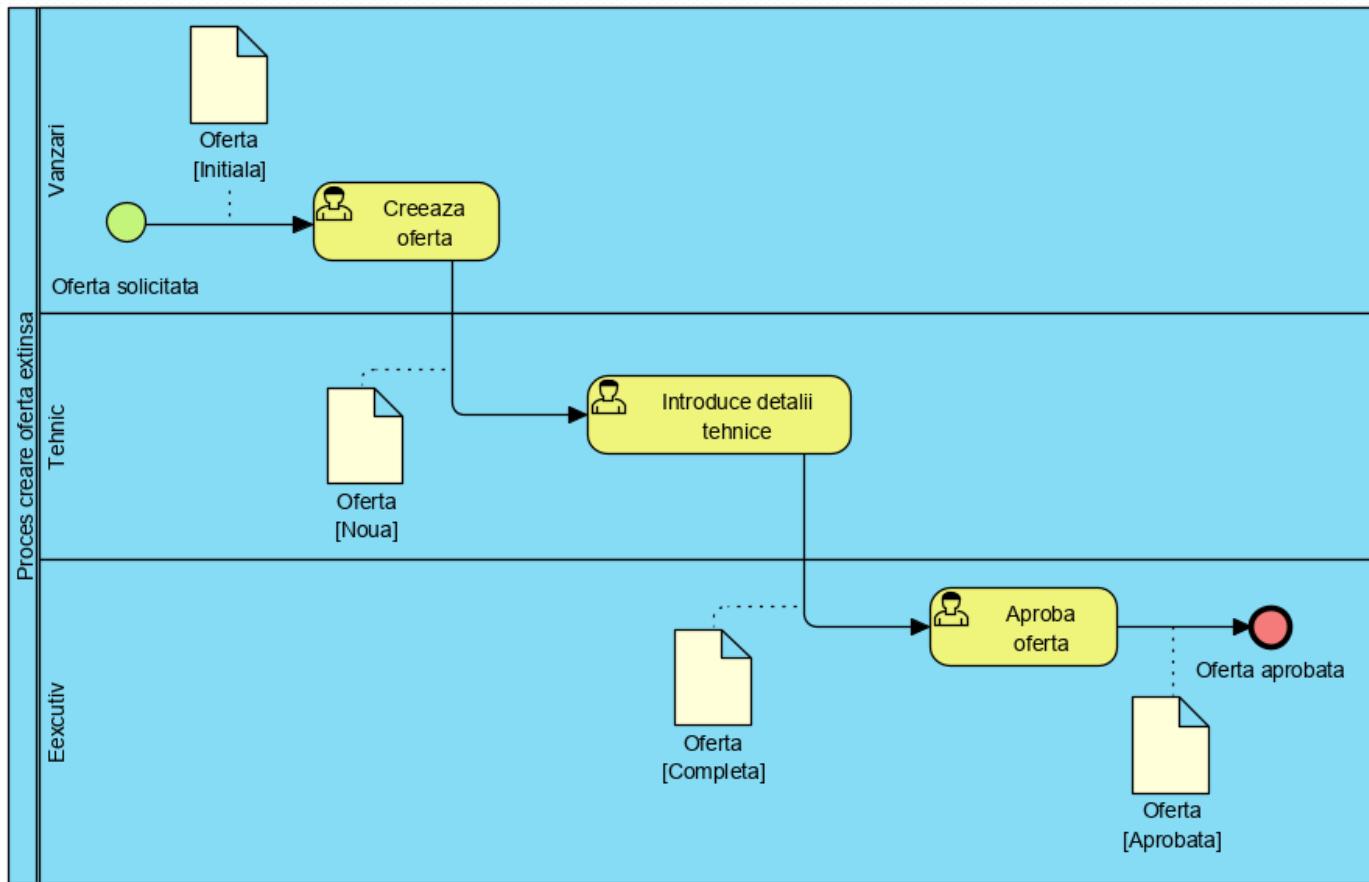
Obiecte de date – notații alternative



Utilizarea ulterioară a unui obiecte de date

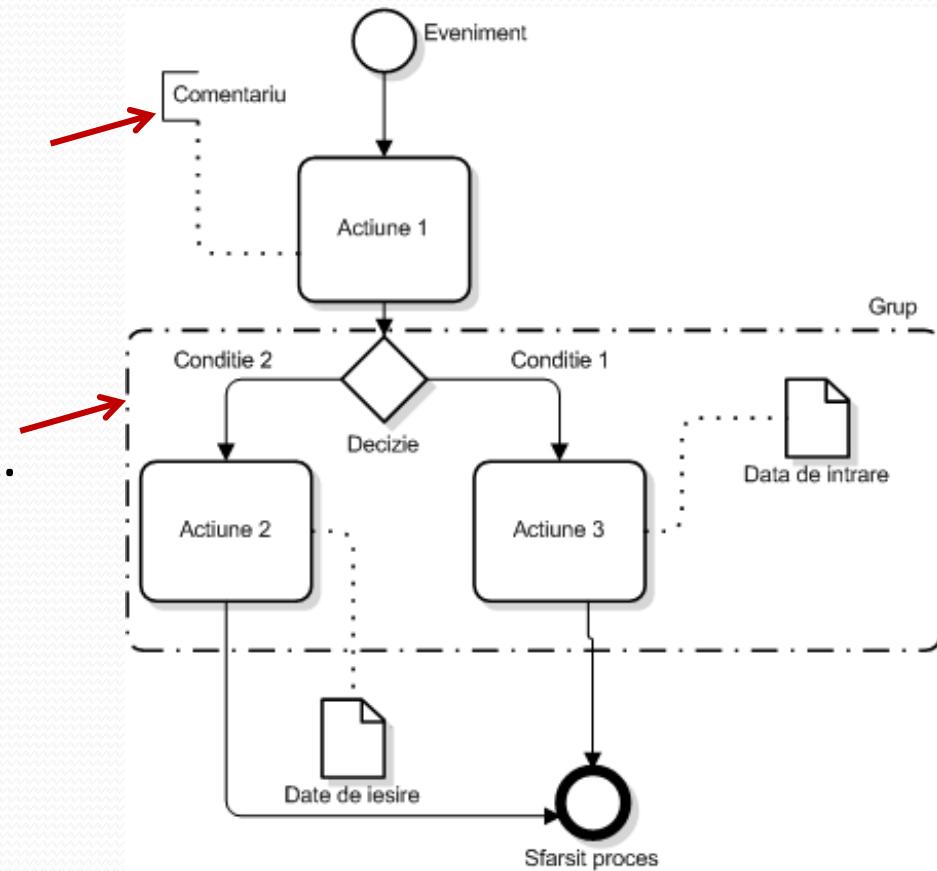
Date - stare

- În timpul transmiterii datelor printr-un proces, acestea își pot **schimba starea** pe parcurs.
- Starea datelor apare **între paranteze drepte** sub numele acestora.



5. Artefacte

- **Adnotări:** mecanism folosit pentru a adăuga informații adiționale în model.
- **Grup:** un element de grupare folosit în scopuri de documentare și analiză care nu afectează secvența de flux.



6. Tipuri de diagrame

- Un model de proces de afaceri nu este un concept uniform, având notații singulare.
- Specificația BPMN 2.0 conține patru tipuri de astfel de modele, și anume:
 - diagrama de procese de afaceri – conține un singur container
 - diagrama de colaborare – mai multe containere
 - diagrama de coregrafie
 - diagrama de conversație
- Fiind cea mai detaliată dintre acestea, diagrama de procese de afaceri este și cea mai uzitată în practică, celelalte trei tipuri de diagrame putând fi considerate o reprezentare sintetică a cunoștințelor specifice despre procesele de afaceri



Recapitulare -1

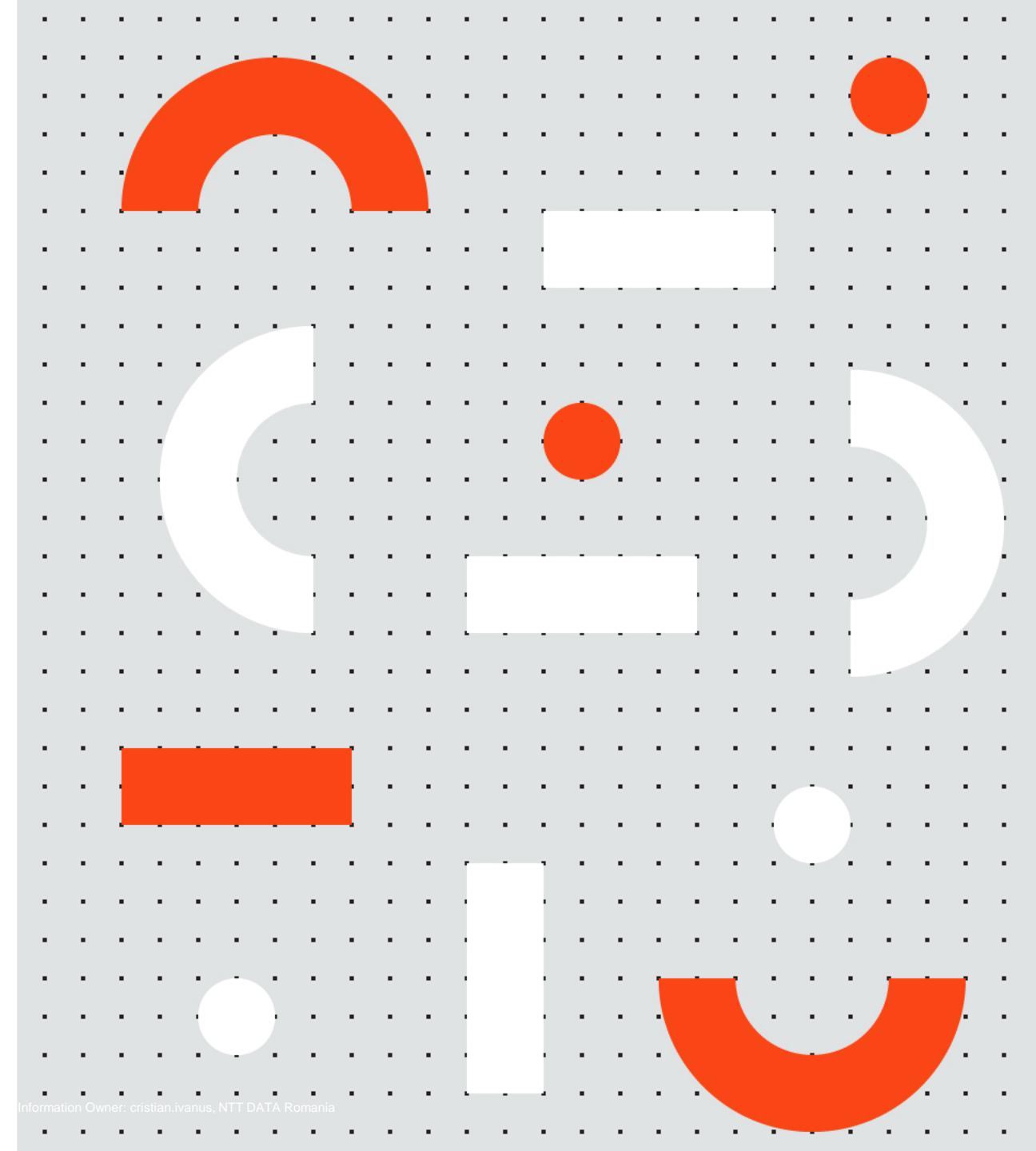
- 1) Care sunt elementele de bază ale limbajului BPMN?
- 2) Evenimentele de început, sfârșit și intermediare au reprezentări diferite. Cu ce simbol grafic se reprezintă un eveniment intermediar?
- 3) Ce tip de structură de control din programare modelează porțile exclusive?
- 4) Porțile paralele verifică îndeplinirea unor condiții?
- 5) În figura de la pagina 30 pot avea și transport aerian și transport terestru, sau se exclud?
- 6) Pentru care tip de porți decizia este luată de pe baza unor date care nu sunt accesibile procesului analizat?



Recapitulare- 2

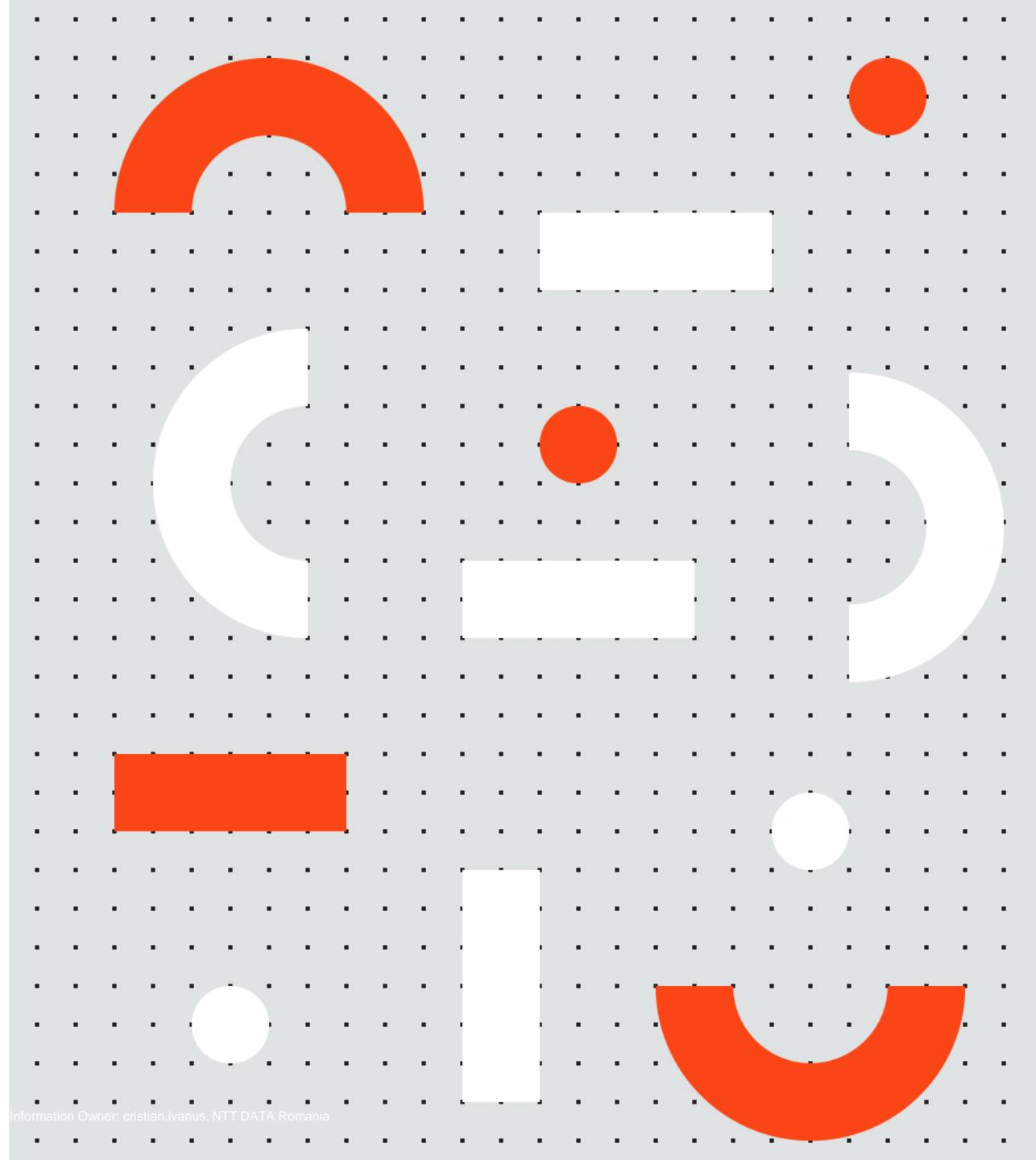
- 7) Când folosim porțile complexe?
- 8) Ce tipuri de obiecte de conectare oferă limbajul?
- 9) Când folosim fluxurile de secvență? Dar pe cele de mesaj?
- 10) Prin ce se diferențiază containerele de culoare?
- 11) Fluxul de secvență poate traversa containere?
- 12) Ce tip de obiecte de date sunt “Produsele comandate” și “Plan de transport” din figura de la pagina 51?

Introduction to RPA



Module 1:

Introduction to RPA



Agenda

01

Introduction to RPA

02

Need for RPA

03

Benefits of RPA in a business environment

04

Industries & domains where RPA can be deployed

05

Processes best-suited for RPA

06

Types of Robots

07

Ethics & best practices of RPA



Learning Objective

01

To understand Robotic Process Automation and its application

02

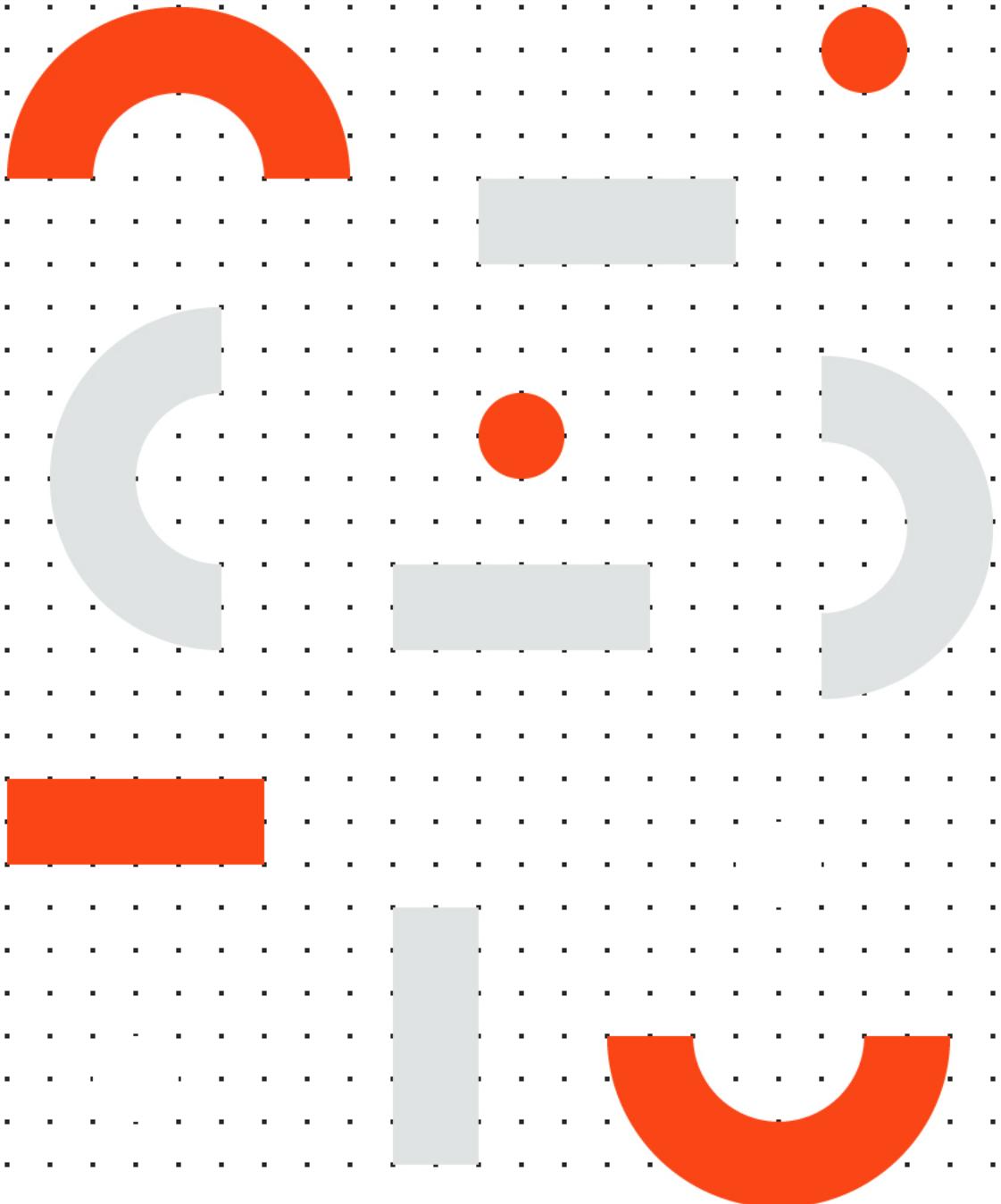
To describe processes that can be automated

03

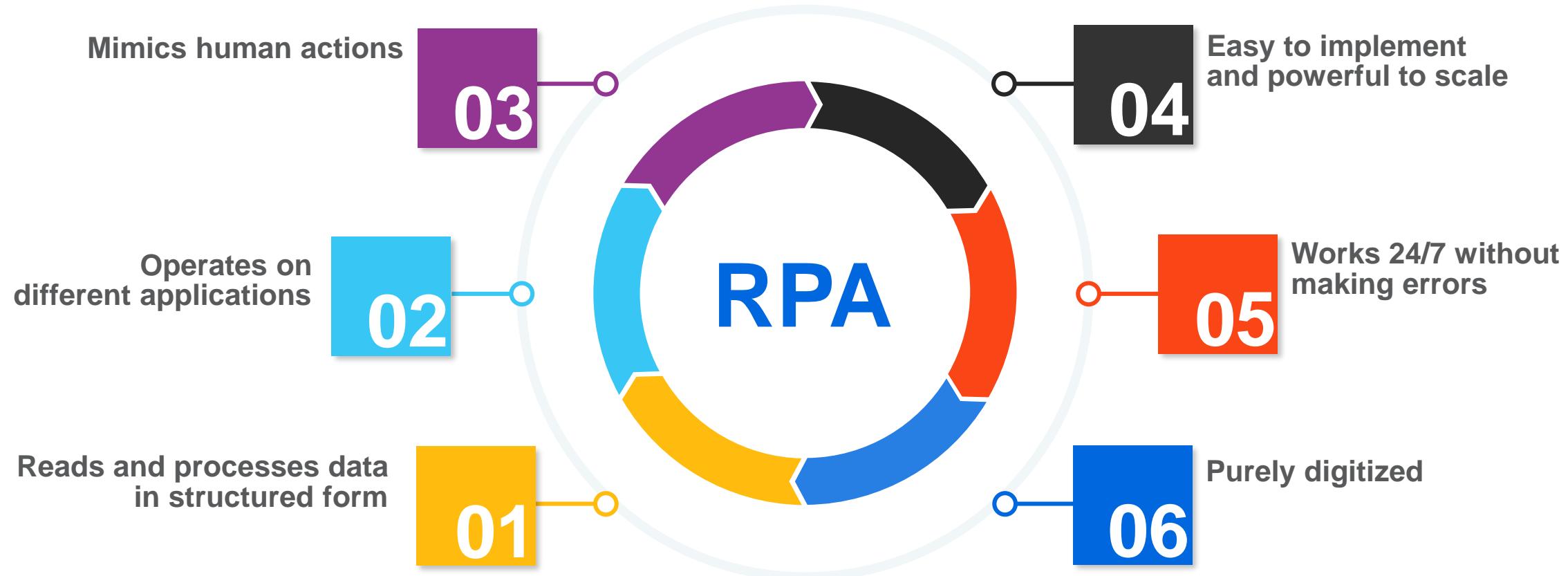
To learn RPA ethics and best practices



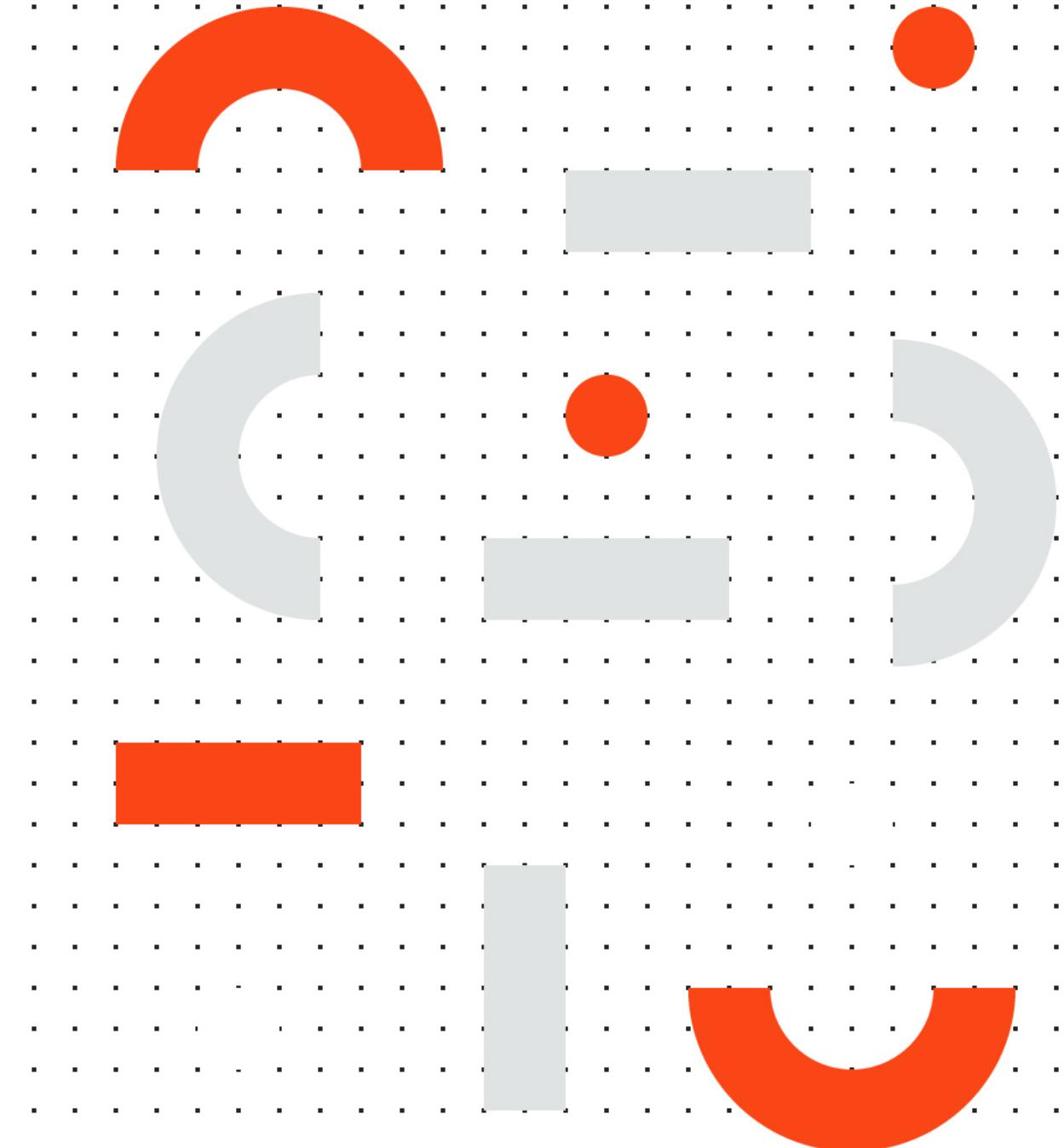
What is RPA?



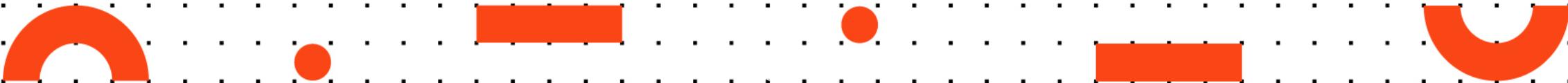
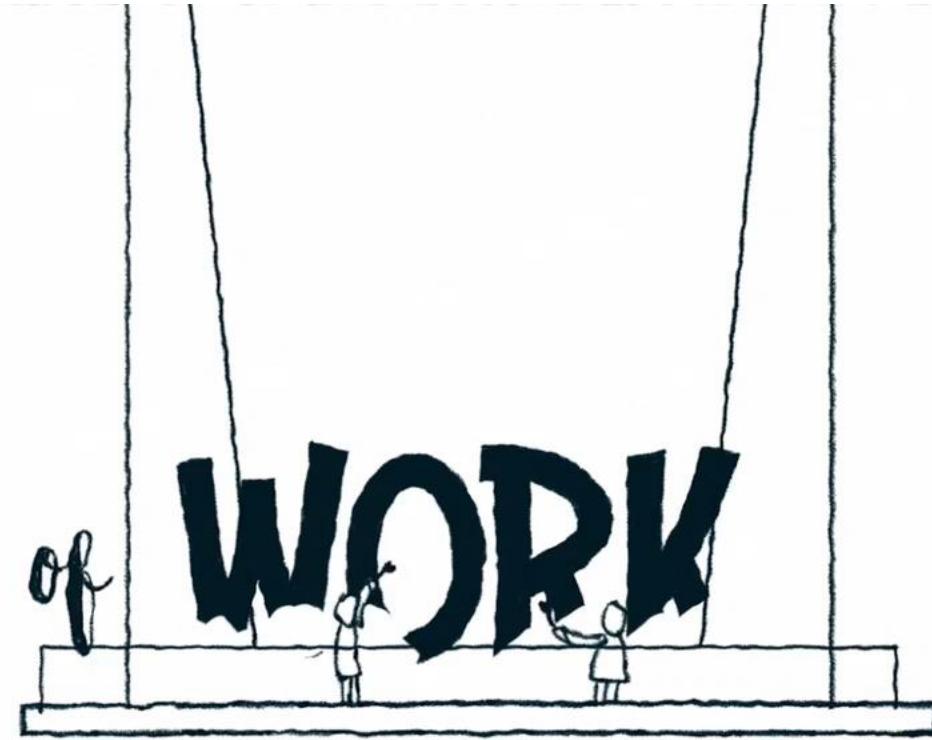
Robotic Process Automation



Need for RPA



the **STORY** of **WORK**



RPA in Business

Businesses want to:

- upgrade service quality
- minimize costs

Technology helps:

- increase productivity
- stay competitive

Goals achieved:

- revenue maximization
- optimum resource utilization



01

RPA can do repetitive jobs more quickly, accurately, and tirelessly than humans



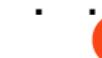
02

RPA helps in achieving operational efficiency by reducing operating costs

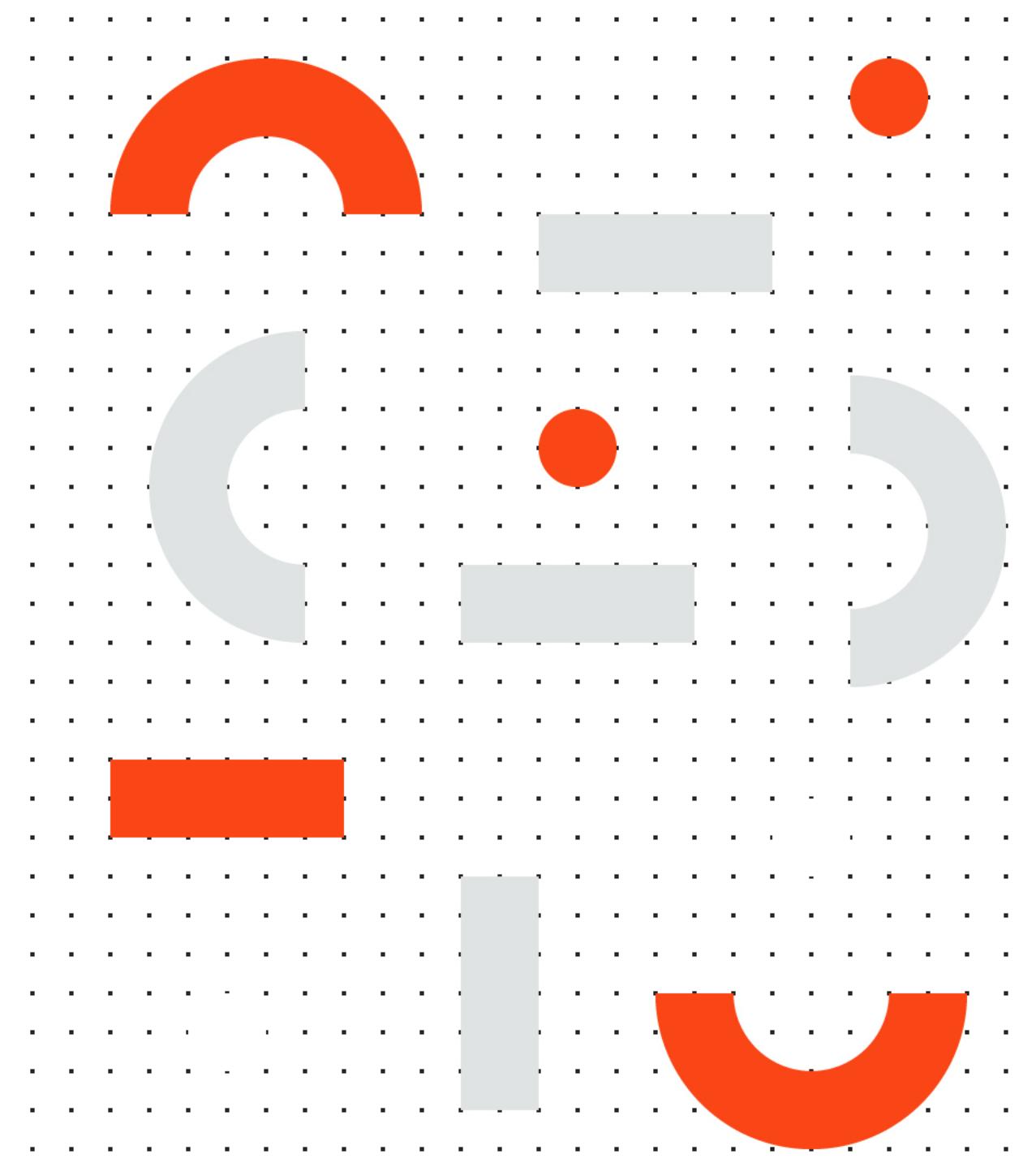


03

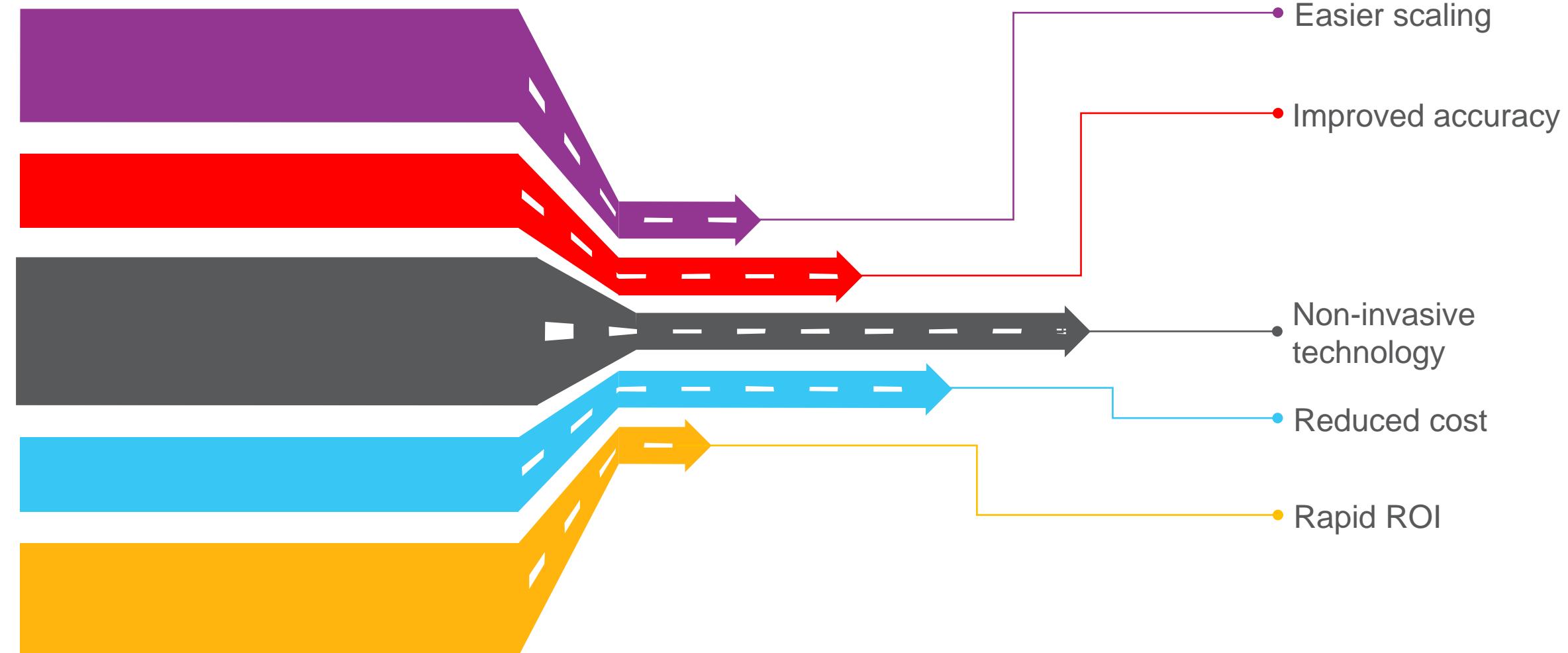
RPA helps organizations liberate their workforce to focus on high-value tasks



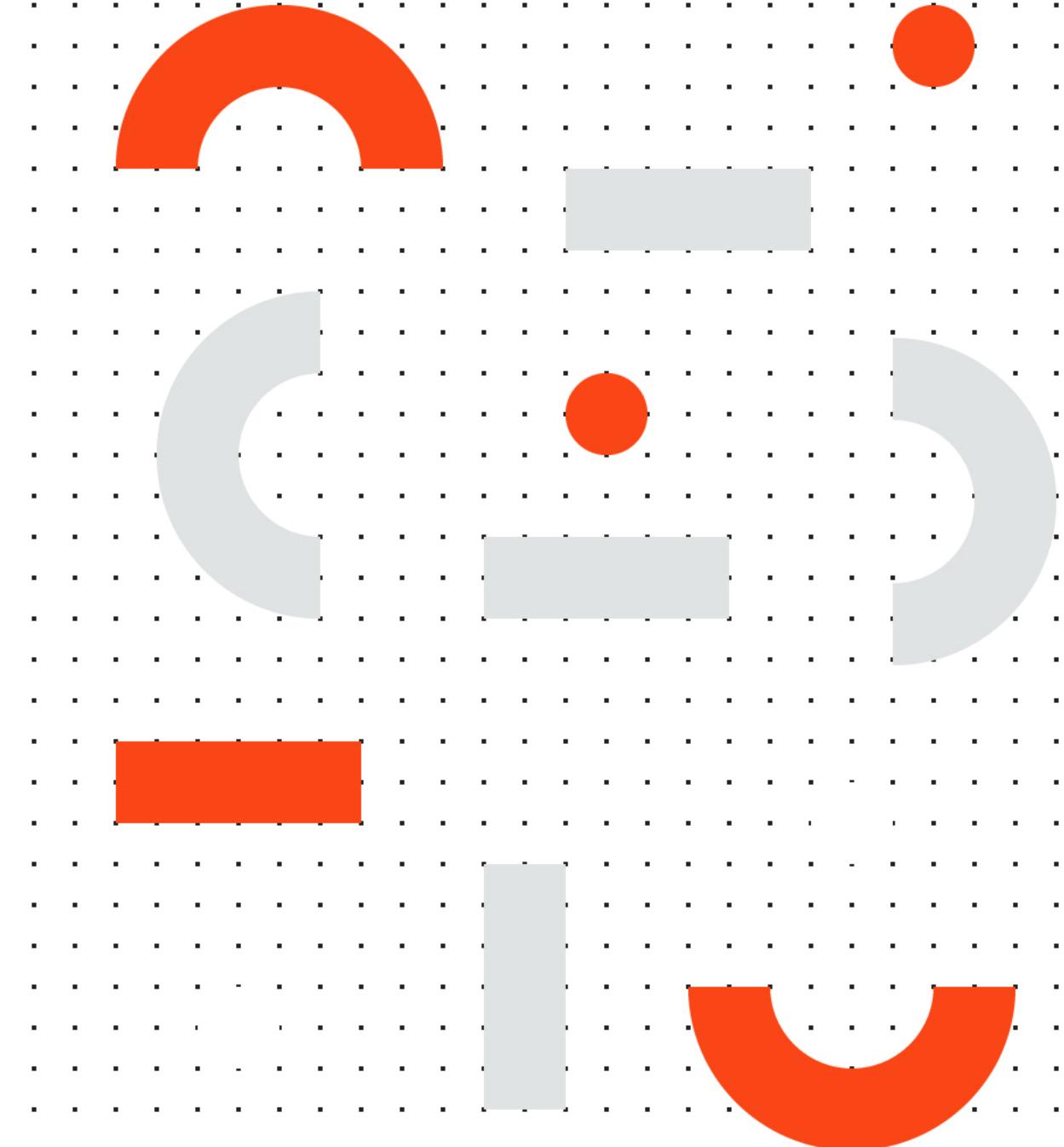
Benefits of RPA in a Business Environment



Benefits of RPA in a Business Environment



Industries & domains where RPA can be deployed



Industries & Domains where RPA can be Deployed

Insurance



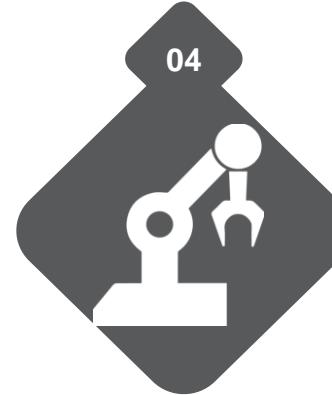
Banking



Healthcare



Manufacturing



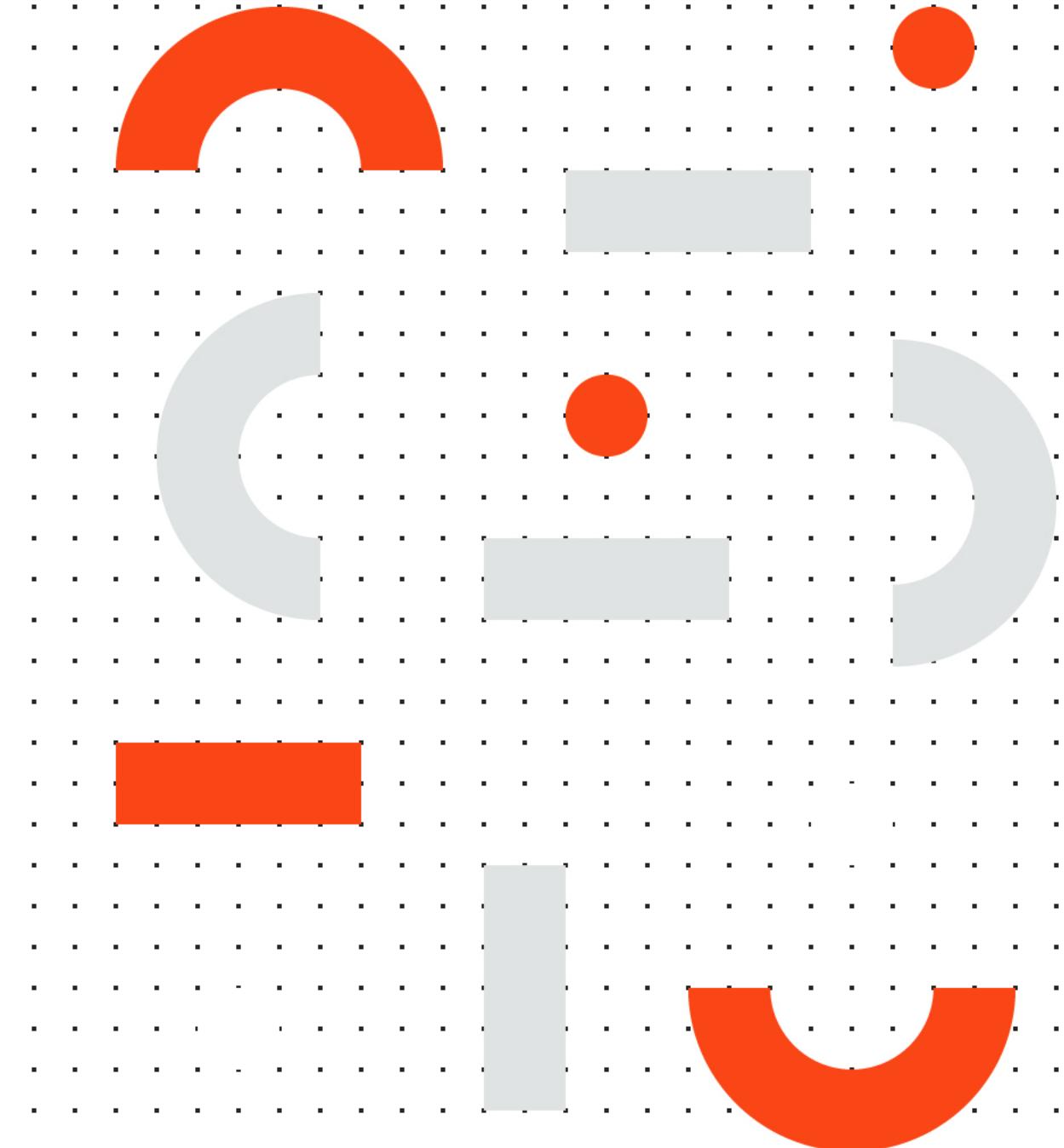
Customer service



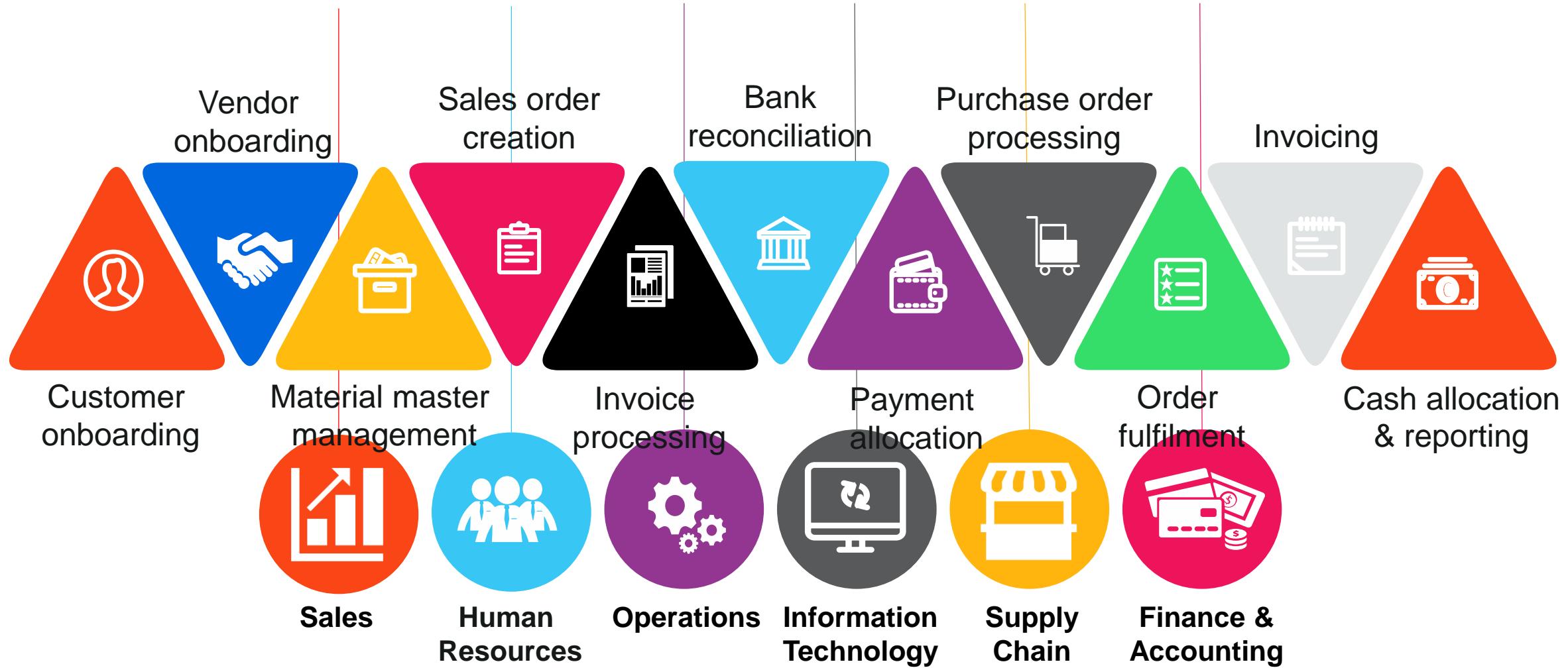
Travel



Processes that can be automated



Departments & Processes suited for RPA



Processes best-suited for RPA

Rules-driven

The processes which are rules-based and consistent are a good choice for automation

Voluminous

Tasks which have high volumes of transactions are always a good choice

Data intensive

Tasks which require a lot of data manipulations and crunching are always a good choice

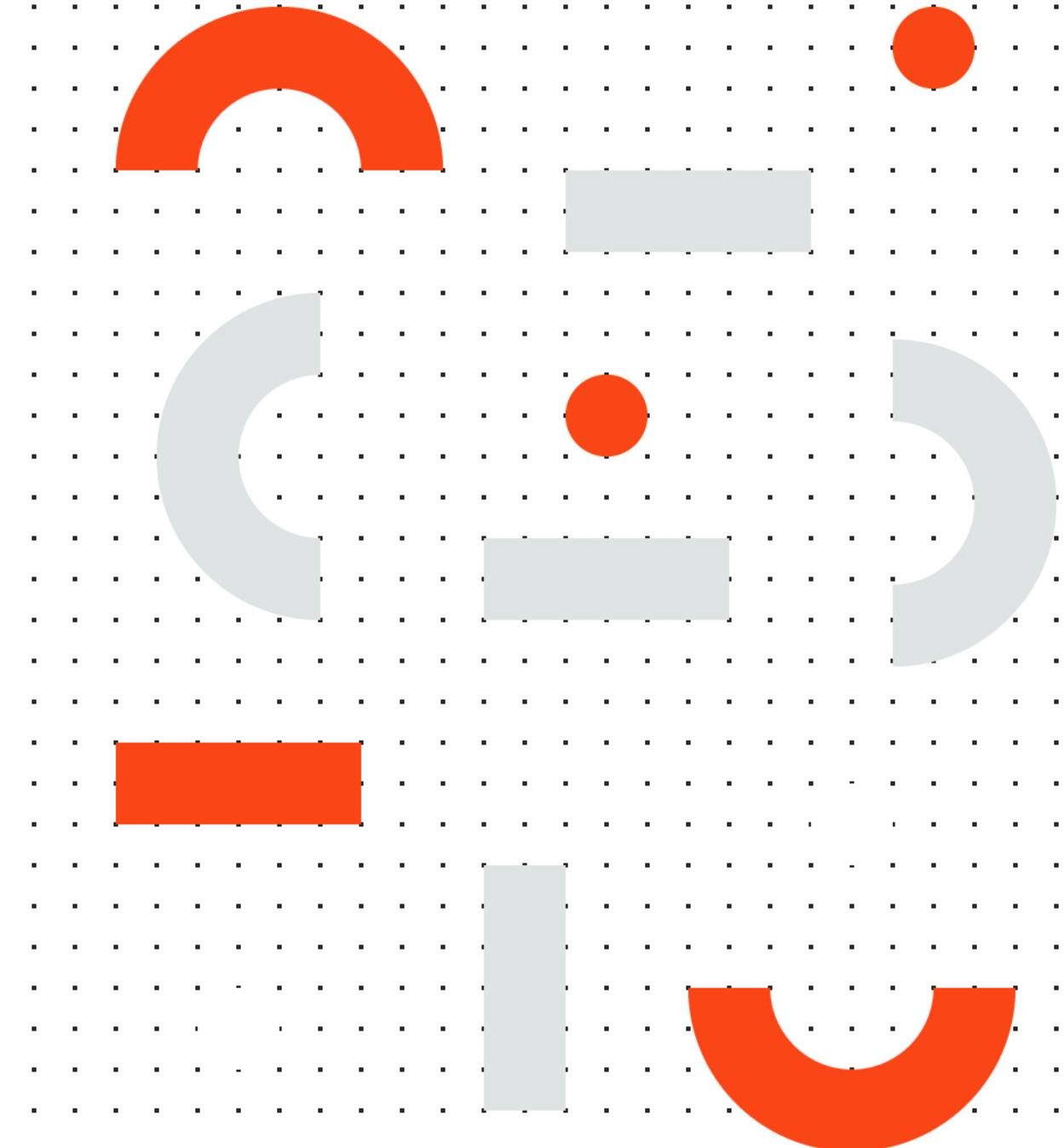
Repetitive in nature

Processes that involve manual and repetitive tasks are the right processes

Driven by electronic inputs

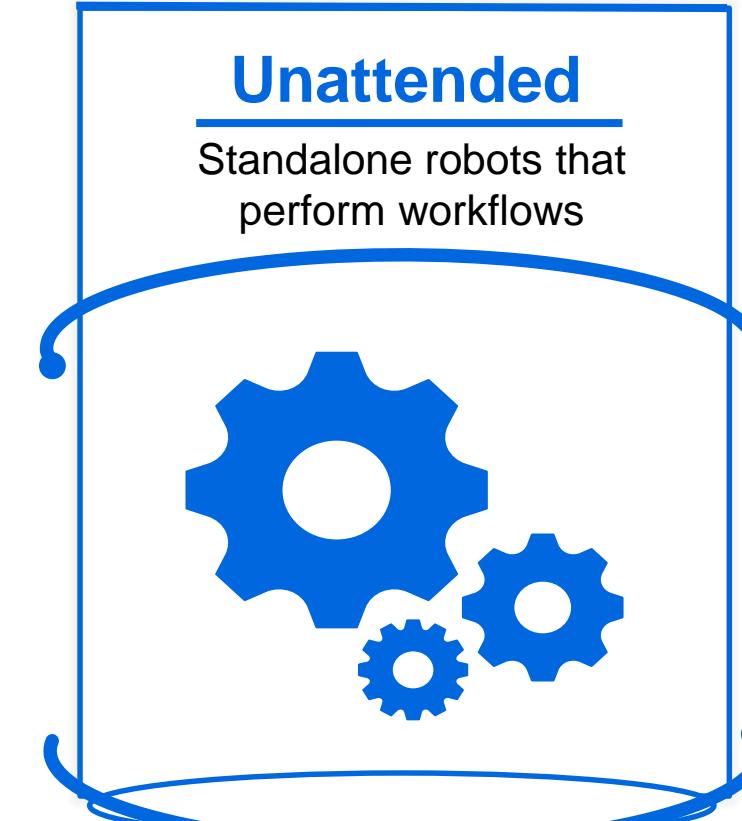
Processes that begin by receiving data through electronic files are a good choice

Robots

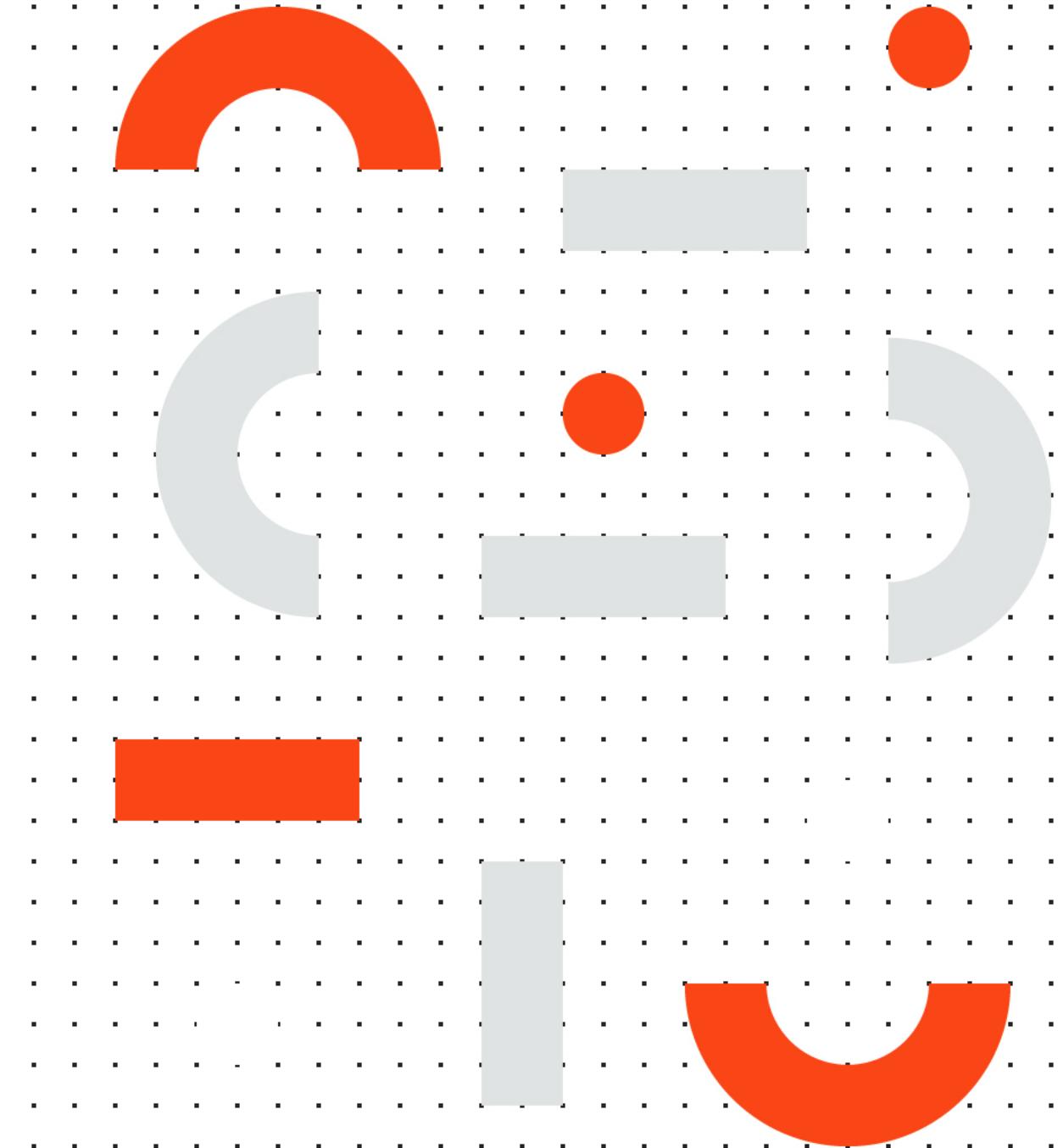


Types of RPA Robots

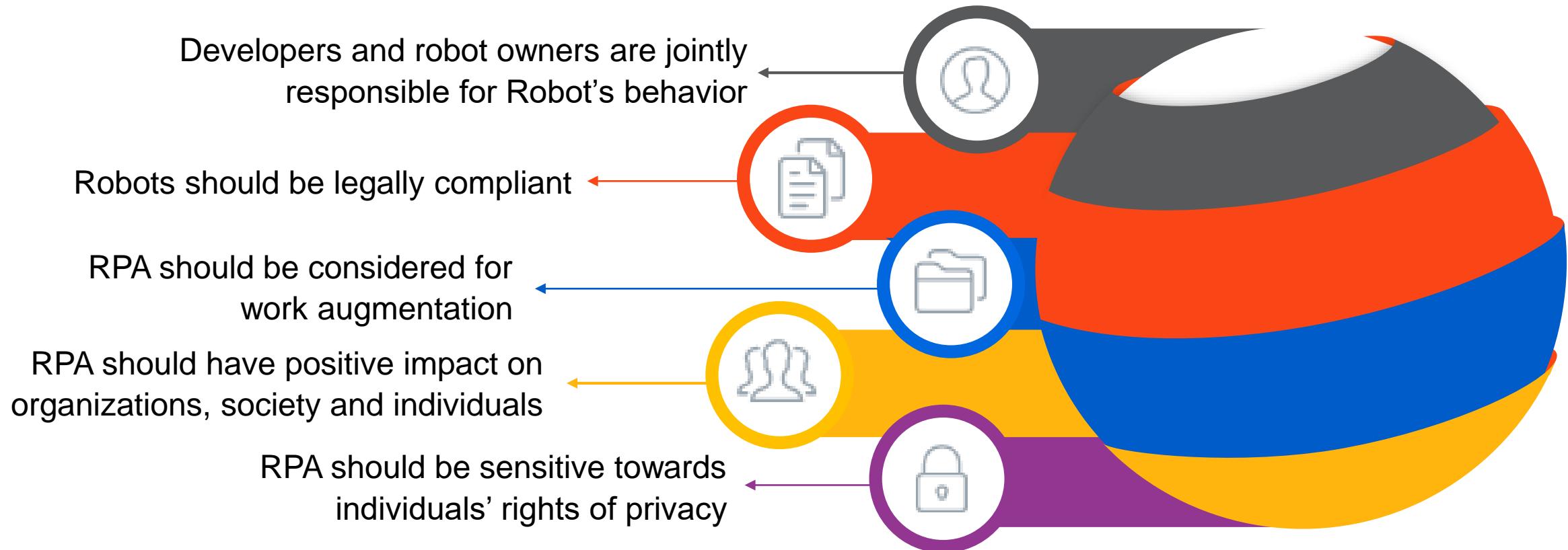
In RPA, robots are categorized on the basis of manual intervention required



Ethics and Best Practices for RPA



Ethical Considerations for Robotic Process Automation



Best Practices for Robotic Process Automation

01

Selecting the
right process

02

Consensus
within the
organization

03

Improving the
process

04

Documentation

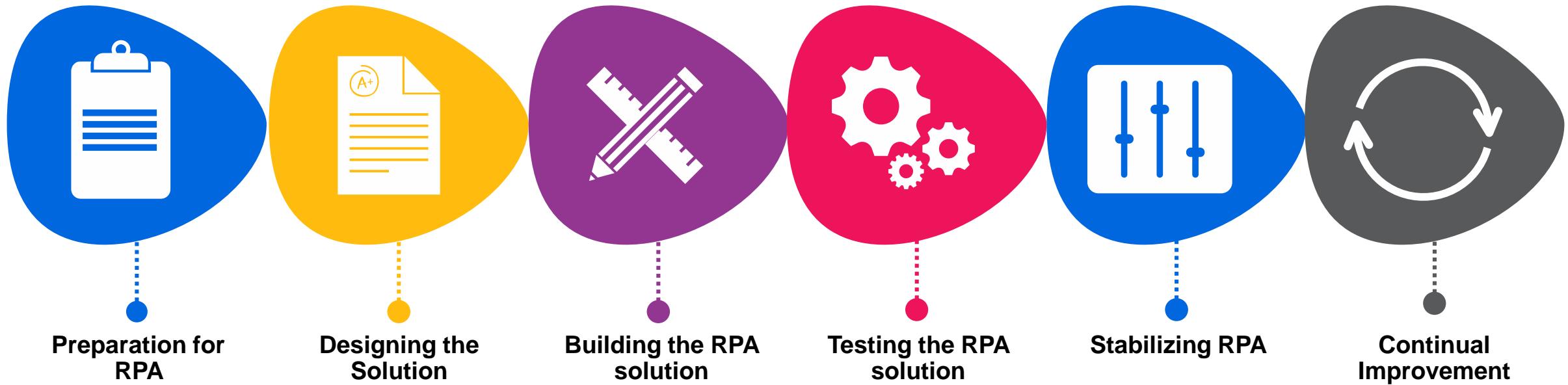


Understanding the Automation Cycle



Understanding the Automation Cycle

An Automation cycle comprises of 6 stages

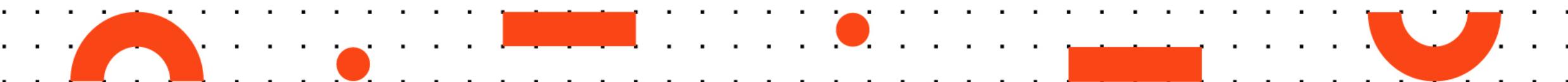


Automation Stages and Activity Owners



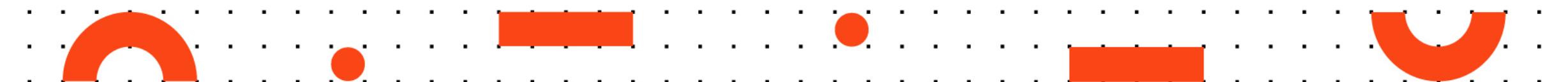
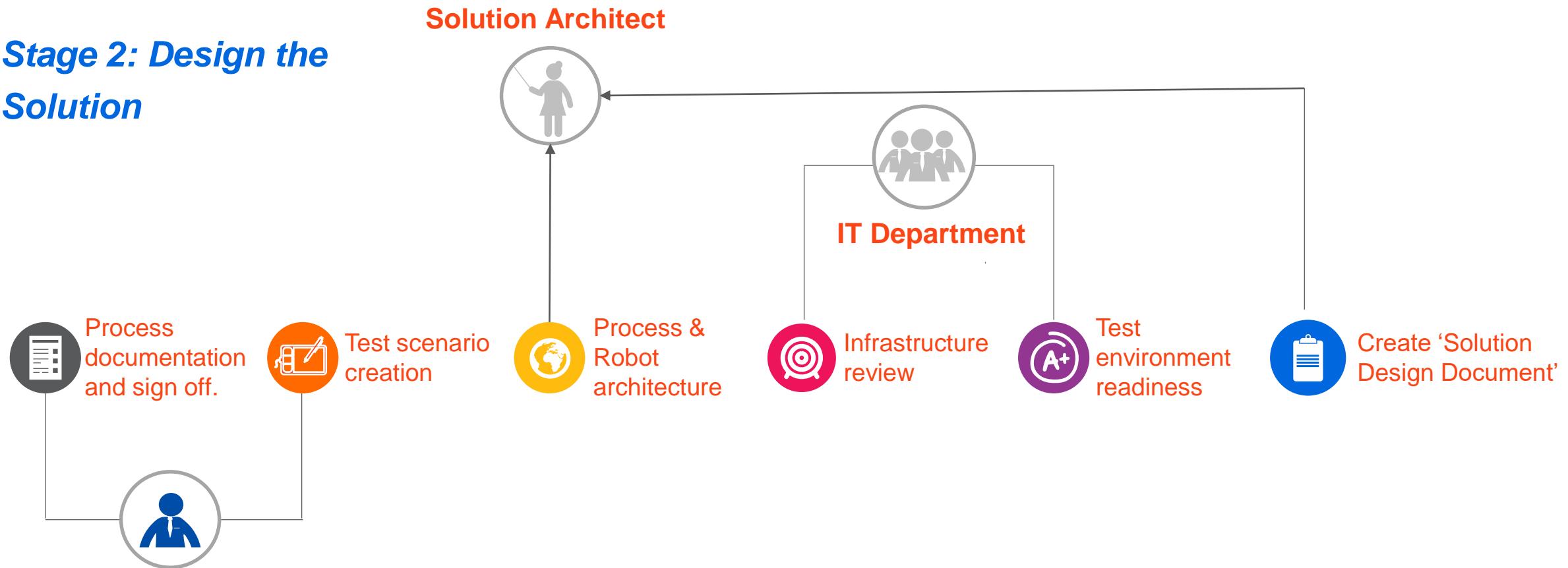
Description of Automation Stages and Activity Owners

Stage 1: Preparation for RPA



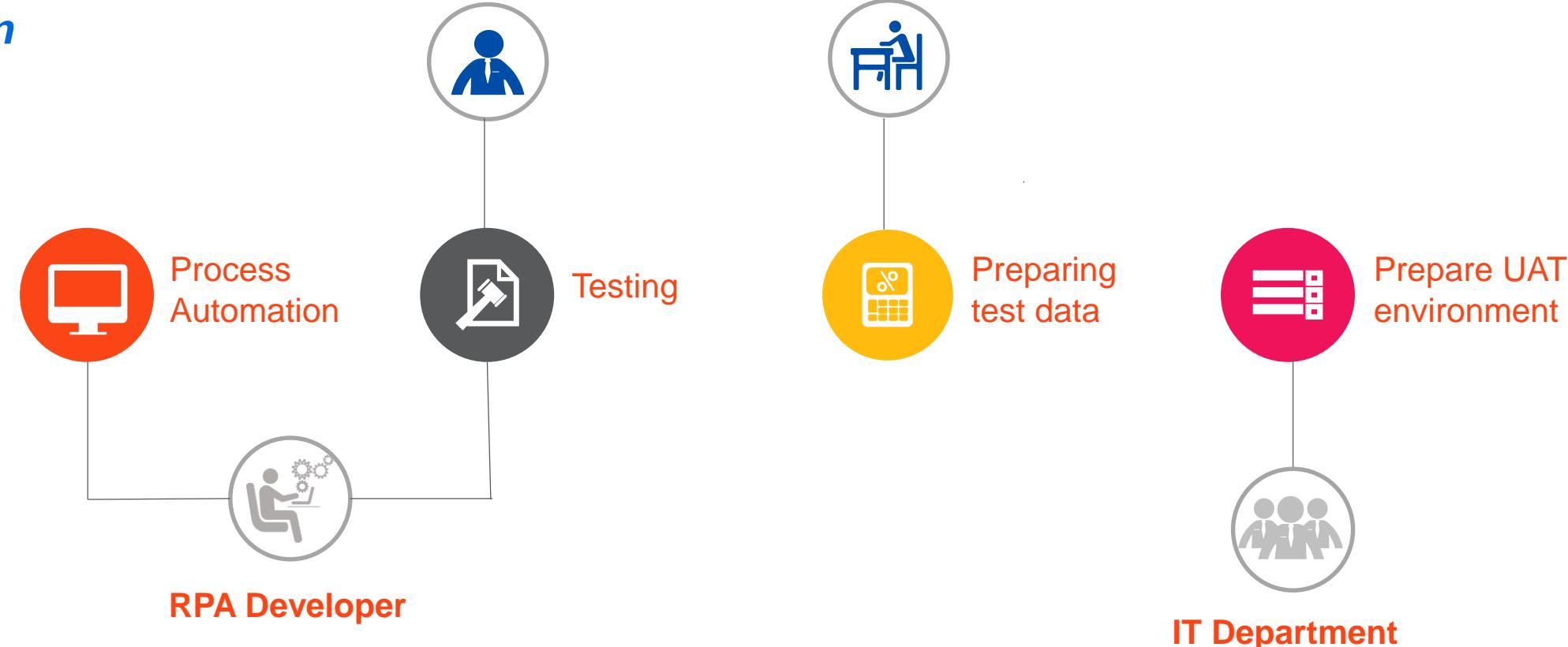
Description of Automation Stages and Activity Owners

Stage 2: Design the Solution

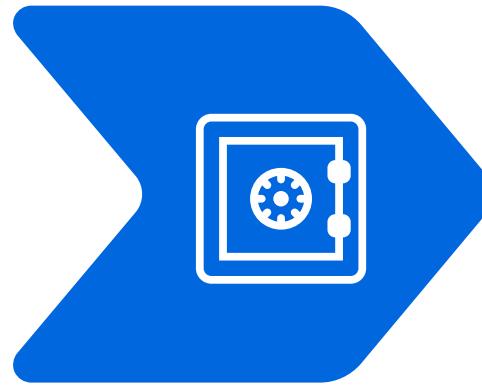


Description of Automation Stages and Activity Owners

Stage 3: Build the RPA Business Manager
Solution

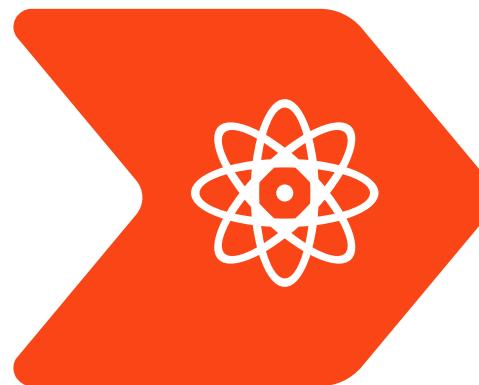


Activities in Preparation for RPA Stage



● Process Identification

- Devise strategy
- Identify processes to be automated
- Identify & calculate complexity factor
- Business benefit mapping
- Prioritize process
- Map process into quadrant



● Opportunity Assessment

- The Complexity of process
- FTE
- Automation Quadrant



● Planning and Communicating

- Project Plan
- Requirement gathering
- Prepare Process Map
- Review
- Document and sign off
- Communication emails

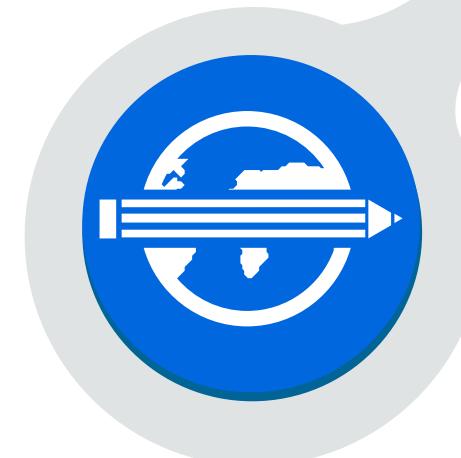


Activities in Design the Solution Stage



Create test scenarios

- Business manager creates multiple test scenarios
- ‘What-if analysis’ and ‘scenario planning’ frameworks

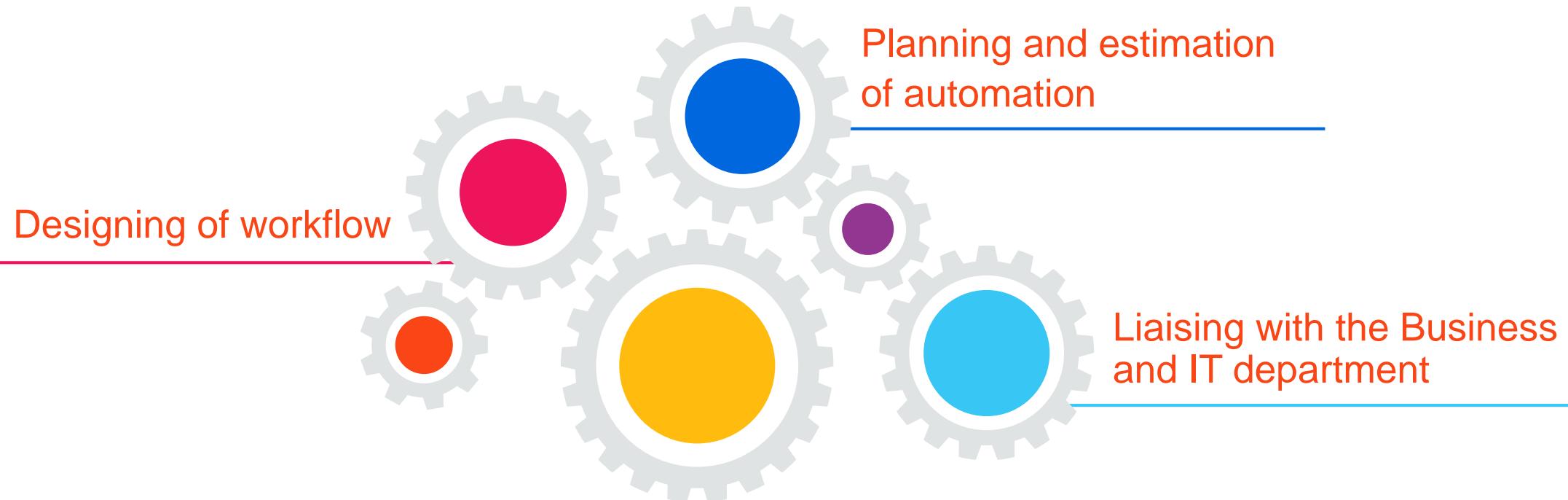


Process Design Document – Creation and Sign off

- Business manager creates complete documentation
 - Process Description
 - L4 Process/ Key-stroke level activities
 - ‘As-Is’ & ‘To-Be’ process map
- Sign off from Business team, IT team & RPA development team

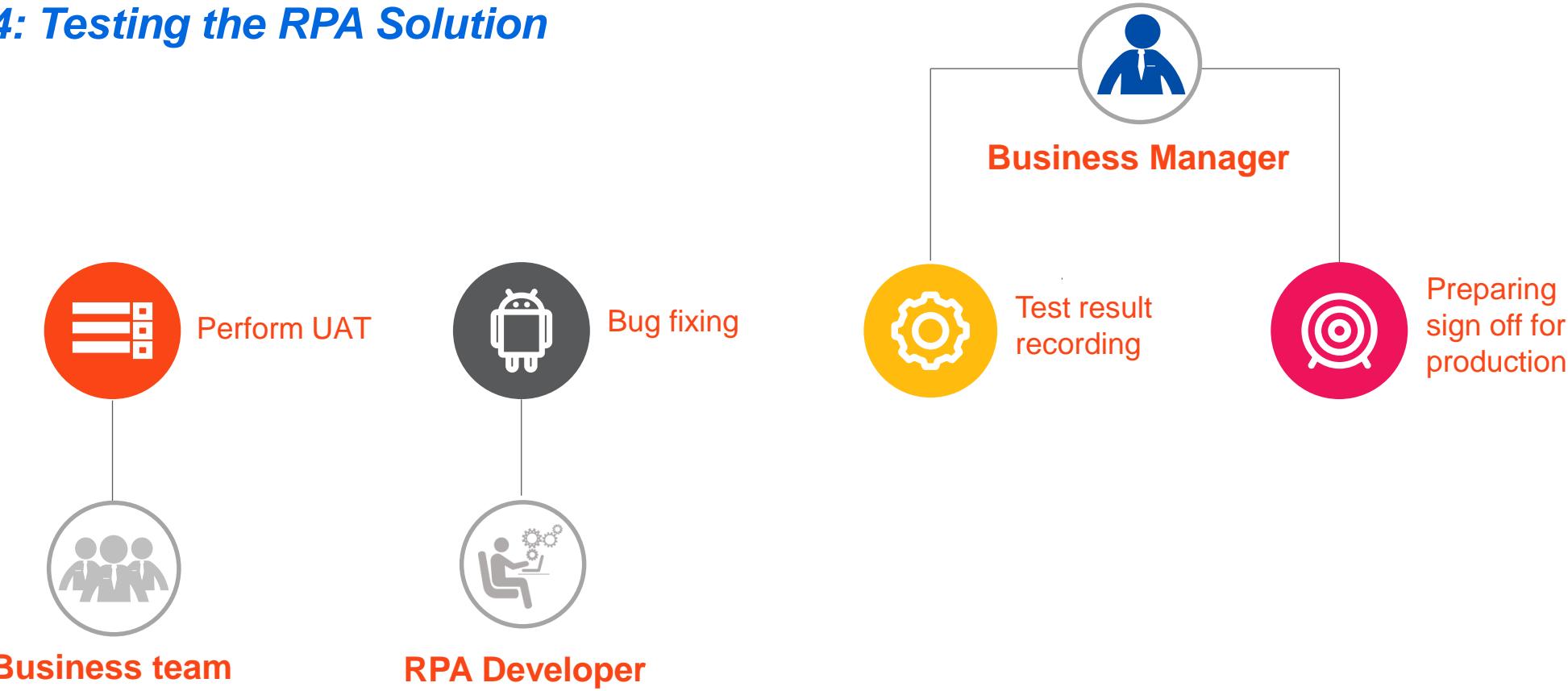


Activities in Building RPA Solution Stage



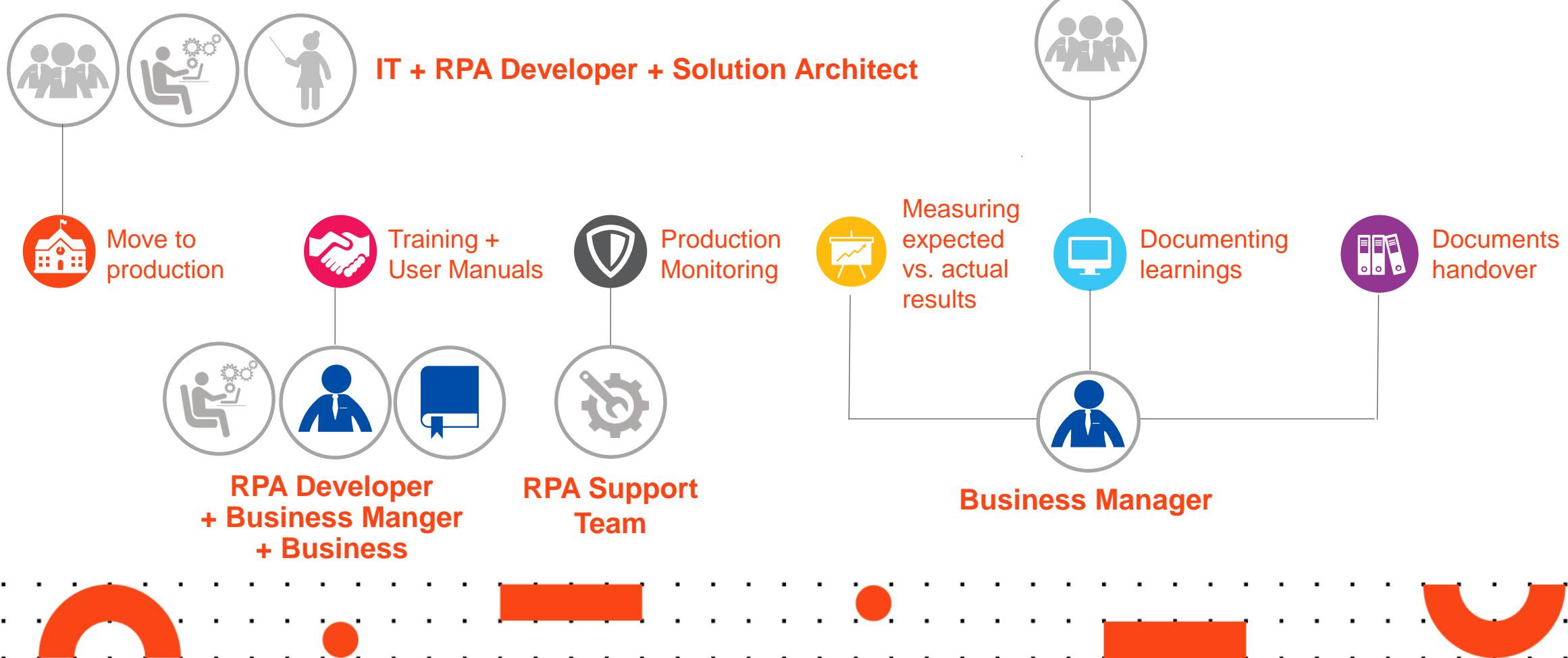
Activities in Testing RPA Solution Stage

Stage 4: Testing the RPA Solution



Activities in stabilization of RPA Solution Stage

Stage 5: Stabilize RPA



Activities in Continual Improvement RPA Solution Stage

Stage 6: Continual Improvement



Performance assessment



Change Management



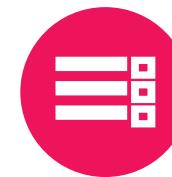
Business Manager



Business Team



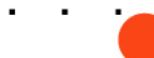
Benefit tracking



Prepare UAT environment



RPA Support Team



Activities in Testing RPA Solution Stage



Test Strategy

- What to test
- How to test
- Testing time & date
- Environment readiness



Test Design

- Business team & RPA developer discuss test cases & scenario



Test Execution

- Perform UAT with business
- Record results



Defect Management

- Log all defects
- Record defects
- Resolve defects



Test Result

- Share testing result with business team for validation



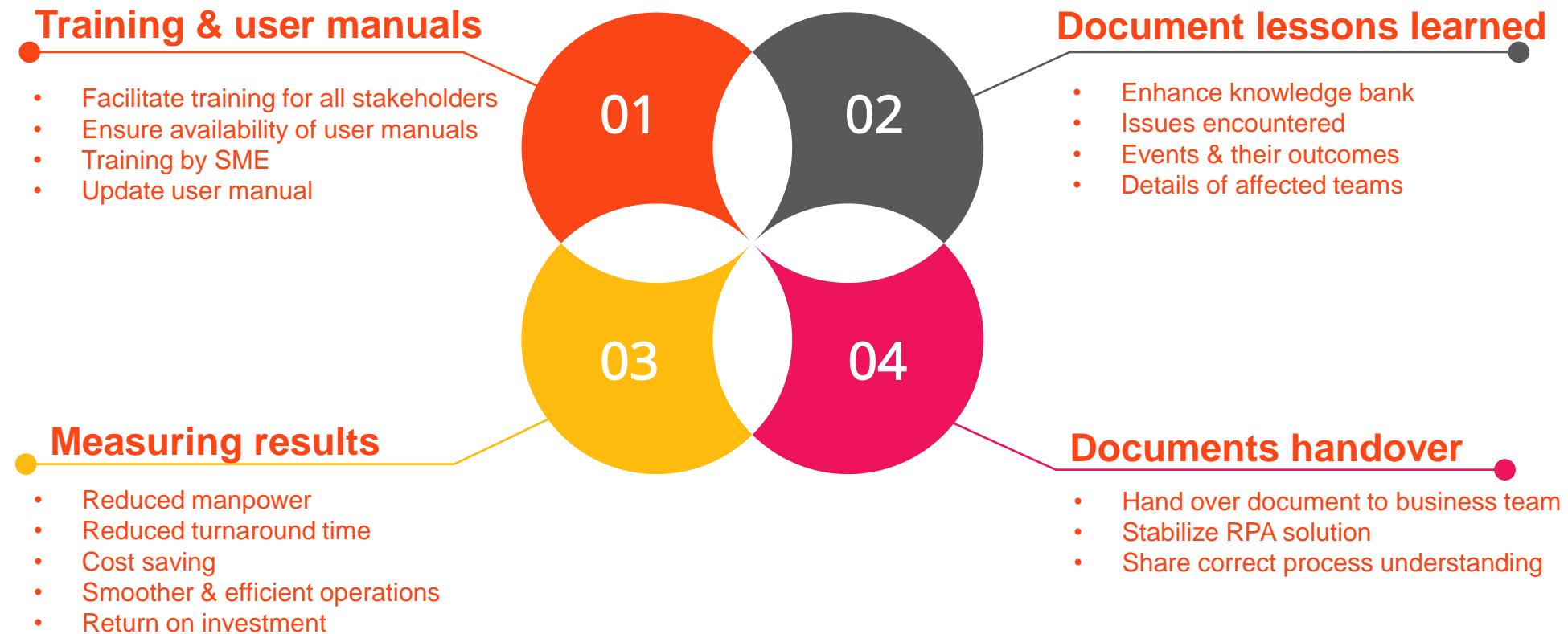
Final Sign off

- Sign off from business
- Move solution in production environment

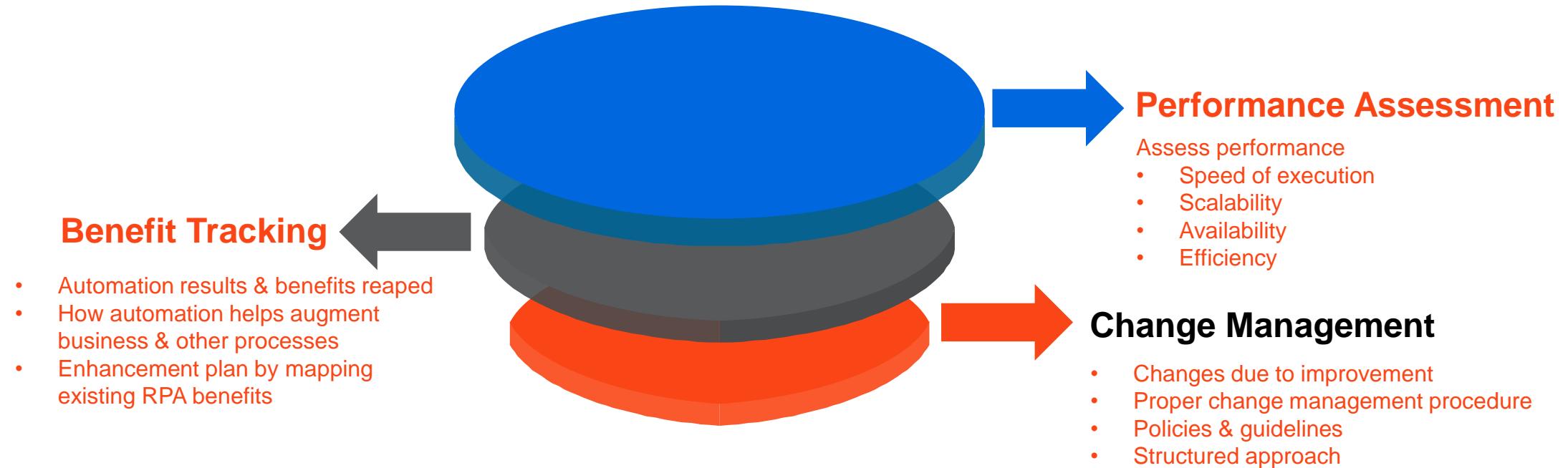


Activities in Stabilizing RPA Stage

The key activities performed by the Business Manager in this stage are:



Activities in Continual Improvement Stage



Industry Use-case: Invoice Processing

01

Information extraction & transfer

- Read out information from invoice & extract key information
- Open company's database
- Process invoices one by one

02

Email notification

Send posting notifications in the form of emails to responsible employee

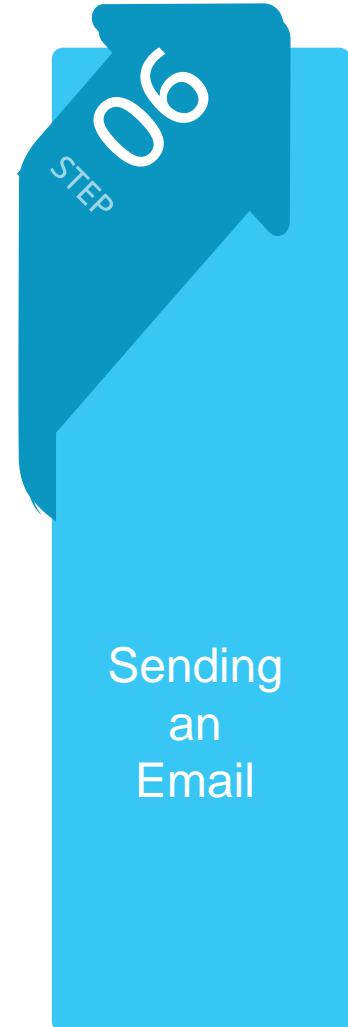
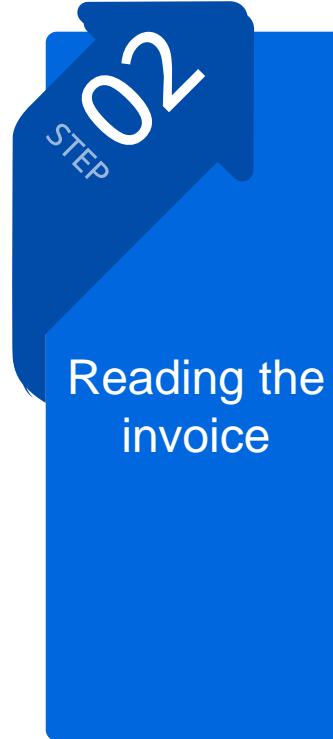
03

Other background activities

- Monitor dedicated invoice folder
- Perform basic checks on company's database
- Verify vendor information on invoice with database



Steps involved in the Invoice Processing



ACADEMIA DE STUDII ECONOMICE BUCUREŞTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

PROIECTAREA SISTEMELOR INFORMATICE

- CURS 9 -

BUCUREŞTI
2020-2021

Proiectarea sistemelor informatice

– partea I –

Cuprins

- ✓ Aspecte generale ale proiectării sistemelor informaticе
- ✓ Principiul proiectării eşalonate a sistemelor informaticе
- ✓ Proiectarea mediului de execuţie
- ✓ Proiectarea arhitecturii sistemului informatic
 - 1. Arhitectura client server
 - 2. Arhitectura orientată pe servicii
- ✓ Îmbunătăţirea modelului de analiză
- ✓ Proiectarea interfeţelor



Aspecte generale ale proiectării sistemelor informatice

- Proiectarea sistemului informatic constă în:
 - stabilirea soluțiilor logice
 - specificarea din punct de vedere fizic a componentelor noului sistem.
- Proiectarea se bazează în principal pe rezultatele obținute din cele două grupe de activități premergătoare:
 - definirea soluției de realizare a noului sistem
 - modelarea noului sistem.

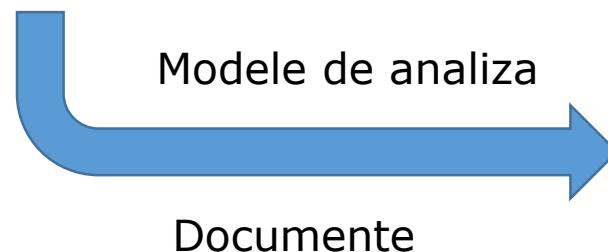


Obiectivele proiectării

Activități de analiză

Obiective: să înțeleaga:

- Evenimentele și procesele din companie
- Activitățile sistemului și cerințele de procesare
- Stocarea și necesarul de informații



Activități de proiectare

Obiective:

- Să definească, să organizeze, să structureze componentele sistemului soluție



Aspecte generale ale proiectării sistemelor informaticе

- Unele metodologii împart proiectarea sistemelor informaticе în:
 - A. **proiectare generală**/proiectare de ansamblu / conceperea sistemului informatic
 - B. **proiectarea de detaliu** – include proiectarea detaliilor specifice ale programelor:
 - i. Proiectarea fiecarui caz de utilizare
 - ii. Proiectarea bazei de date
 - iii. Proiectarea interfețelor cu utilizatorul și cu alte sisteme
 - iv. Proiectarea controlului și a securității
- În cadrul acestor etape, sistemul este proiectat din punct de vedere **logic** și din punct de vedere **fizic**, separat sau nu.



Principiul proiectării eşalonate a sistemelor informatic

- La stabilirea ordinii de prioritate în abordarea structurilor sistemului informatic pot fi avute în vedere următoarele criterii:
 - **prioritatea** obiectivelor componente;
 - asigurarea **legăturilor** între componente;
 - disponibilitatea resurselor (**limita fondurilor** ce pot fi alocate în timp pentru realizarea sistemului informatic);
 - nivelul de **dotare cu tehnică** de calcul existent în etapa de concepere și cel prevăzut a fi atins în timp;
 - **forțele de proiectare** pe care le va antrena proiectul;
 - **personalul de specialitate existent și în pregătire**, la unitatea beneficiară, necesar pentru implementarea și exploatarea curentă a sistemului informatic).



Întrebări cheie ale activității de proiectare

Activitate de proiectare	Intrebare
Proiectarea mediului de execuție	A fost specificat în detaliu mediul de execuție software cu toate opțiunile?
Proiectarea arhitecturii aplicației și software-lui	Au fost specificate în detaliu toate elementele software-lui și cum va fi realizat fiecare caz de utilizare?
Proiectarea interfețelor cu alte sisteme	A fost specificat în detaliu modul în care sistemul va comunica cu toate celelalte sisteme din interior/exteriorul organizației?
Proiectarea bazei de date	Au fost specificate în detaliu toate cerințele de stocare ale datelor, inclusiv toate elementele schemei?
Proiectarea controalelor și securitatii sistemului	Au fost specificate în detaliu toate elementele pentru a asigura siguranța sistemului, securitatea și protecția datelor?



I. Proiectarea mediului

- **MEDIUL** reprezintă toate tehnologiile necesare pentru a susține aplicația software
 - Calculatoare server sau desktop
 - Dispozitive mobile, sisteme de operare
 - Facilități de comunicație, de intrare și de ieșire
- Se mai numește și **Arhitectura tehnică**



II. Proiectarea arhitecturii aplicației și software-lui

- Împărțirea sistemului în subsisteme
- Definirea **arhitecturii software**
 - Pe trei niveluri sau model-view-controller
- Proiectarea detaliată a fiecarui **caz de utilizare**
 - diagrame de clase de proiectare
 - diagrame de secvențe
 - diagrame de stări



III. Proiectarea interfețelor cu utilizatorii

- Proiectarea dialogului **pleacă de la cerințe**
 - Fluxul de activități din cazurile de utilizare
 - Diagramele de secvență ale sistemului
- În prezent este necesară proiectarea de interfețe pentru diverse medii și dispozitive:
 - Telefoane smart
 - Tablete, iPad, etc



Proiectarea interfețelor cu alte sisteme

- Sistemul informatic interacționează cu alte sisteme interne sau externe, fiind necesară integrarea acestora
- Conectarea la alte sisteme se poate realiza în multiple feluri:
 - Se pot salva datele de alte sisteme
 - Se pot utiliza date salvate din alte sisteme
 - Se pot face solicitări de informații în timp real
 - Se poate apela la servicii software



Identificarea interfețelor sistemului

- Intrări și ieșiri cu intervenție umană minimă, cu nivel ridicat de automatizare
 - acestea sunt capturate de dispozitive (scanere, senzori etc.) sau generate de persoane care inițiază un proces care se desfășoară fără intervenții umane suplimentare
- Intrări și ieșiri de la/către alte sisteme:
 - acestea sunt interfețe directe cu alte sisteme informaticе, formatare, de obicei, ca mesaje de rețea
- Intrări și ieșiri către baze de date externe:
 - Acestea pot furniza intrări sau accepta ieșiri de la un sistem



IV. Proiectarea bazei de date

- Se pleaca de la **modelul claselor domeniului (ERD)**
- Se alege **structura bazei de date**
 - De obicei se lucreaza cu baze de date relaționale
 - pot există platforme care lucrează cu baze de date orientate obiect sau sisteme NoSQL
- Se proiectează **arhitectura BD** (distribuită, etc)
- Se proiectează **schema bazei de date**
 - Tabelele și coloanele în relațional
- Se proiectează **restricțiile de integritate** referențială
 - Referințe prin chei externe



V. Proiectarea securității și controalelor

- Scopul este de a proteja bunurile companiei
- Este un element crucial în Internet și rețele wireless
- Este nevoie să de proiecteze controale la nivel de:
 - Interfața cu utilizatorul
 - Aplicație
 - Bază de date
 - Rețea



I. Proiectarea arhitecturii tehnice (mediului)

1. Proiectarea unui sistem cu implementare internă

- a) Sisteme software stand alone - care rulează pe un singur dispozitiv, fără conectare la rețea
- b) Sisteme software bazate pe o rețea internă
 - Arhitectura client-server, rețea LAN
 - Aplicații desktop și aplicații bazate pe browser
- c) Arhitecturi client server pe trei niveluri
 - Nivelul datelor, nivelul aplicației și nivelul prezentare
 - Aplicații desktop și aplicații bazate pe browser



I. Proiectarea arhitecturii tehnice

2. Proiectare pentru implementare externă

- Configurare pentru **implementare pe Internet** – cu avantaje și riscuri
- Alternative de găzduire pentru **implementare pe Internet**
 - Colocație
 - Servicii de management
 - Servere virtuale
 - Cloud computing
- Diversitatea dispozitivelor client
 - Laptop, tablete si notebook, telefon smart etc



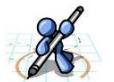
Configurație pentru implementare pe Internet

- **Avantaje**
 - **Accesibilitate** număr mare de utilizatori potențiali
 - **Costuri reduse** ale comunicației
 - **Standarde utilizate pe scară largă** – standarde Web arhicunoscute
- **Probleme potențiale:**
 - **Securitate** – breșe de Securitate pe serverele Web, atacuri hacker
 - https – Hypertext Transfer Protocol Secure
 - TLS – Transport Layer Security – ver. Avansata a protocolului de retea SSL
 - **Volumul datelor trasmise prin rețea** dacă traficul este încarcat, volumul datelor transmise și durata sunt ridicate
 - **Schimbarea standardelor**—Standardele Web se schimbă rapid (funcționalitate vs compatibilitate)



Atribute ale opțiunilor de găzduire

OPȚIUNI DE GĂZDUIRE				
Opțiuni serviciu	Colocare	Managed services	Servere virtuale	Cloud computing
Serviciul oferă găzduire și infrastructură	Da	Da	Da	Da
Clientul deține calculatorul	Da	Poate	Nu	Nu
Clientul gestionează configurația calculatorului	Da	Nu	Posibil	Nu
Scalabilitate	Clientul adaugă calculatoare	Clientul adaugă calculatoare	Clientul cumpără servere virtuale mai mari sau mai multe	Clientul cumpără capacitate suplimentară
Mentenanță	Oferită de client	Oferită de gazdă	Oferită de gazdă	Oferită de gazdă
Backup și recuperare	Oferită de client	Oferită de gazdă	Disponibilă	Disponibilă



II. Proiectarea arhitecturii sistemului informatic

- Cele mai cunoscute **tipuri** de rețele sunt:
 - Rețea punct la punct (bus);
 - Rețea inel (ring);
 - Rețea stea (star);
 - Rețea ierarhică.
- După alegerea tipului de rețea, echipa de analiză trebuie să identifice **protocolul de comunicație**. Cele mai cunoscute protocoale de comunicații sunt:
 - **TCP/IP**: Este un protocol indicat în cazul utilizării unei rețele Ethernet sau în cazul în care calculatoarele din rețea au arhitecturi diferite;
 - **SNA**: Este utilizat, în general pentru conectarea mainframe-urilor IBM.



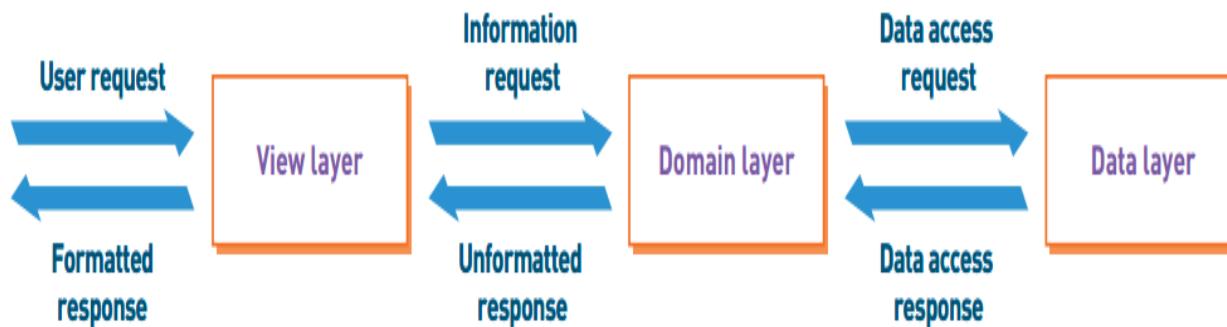
1. Arhitectura client/server

- Arhitectura client/server este un ansamblu de trei componente principale: server, client și o rețea care conectează calculatoarele client la servere pentru a colabora la îndeplinirea sarcinilor.
- Tipurile de aplicații client/server sunt:
 - Sisteme cu baze de date
 - Poșta electronică (E-mail)
 - Sisteme de tip „groupware“
 - Sisteme moștenite
- O arhitectură distribuită presupune existența unor baze de date multiple (care se găsesc pe calculatoare distincte) și a unor aplicații care manipulează datele de la diferite stații de lucru locale cu ajutorul unor sisteme de gestiune a bazelor de date (SGBD).



Proiectarea unei arhitecturi client-server triplu stratificată

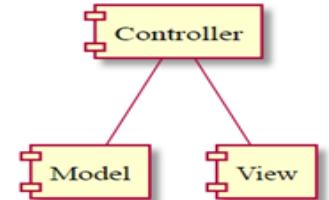
- Conține nivelul datelor, nivelul domeniului problemei, nivelul prezentării



- Potrivit pentru aplicații desktop și aplicații web
- Unul dintre avantajele arhitecturii client-server este acela că permite cu ușurință dezvoltarea aplicațiilor cu arhitectură triplu stratificată



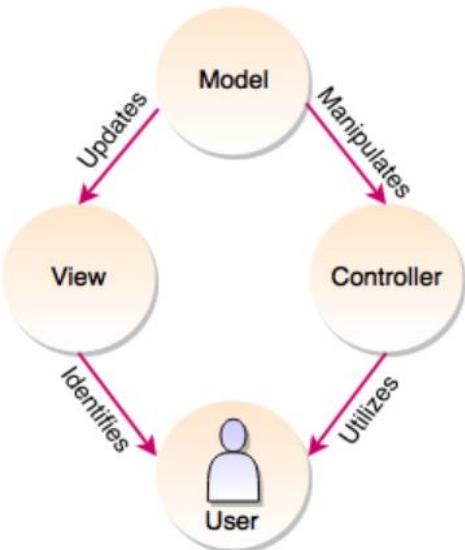
Model View Controller (MVC)



- Este un **model arhitectural** utilizat în ingineria software.
- Ajută la scrierea de cod mai bine organizat și mai ușor de gestionat.
- A fost **intens folosit și testat** pentru o varietate de limbi de programare (Java, PHP, ASP.NET etc.).
- Reprezintă un cadru de lucru important pentru **construirea aplicațiilor web**.
- Separarea oferită de MVC ajută la gestionarea **aplicațiilor complexe** și simplifică dezvoltarea aplicațiilor în echipă.



MVC Model-View-Controller



Model: componentă centrală a MVC, gestionează **datele, logica și constrângerile** unei aplicații. Acesta surprinde **comportamentul** unei aplicații.

View: responsabil pentru afișarea tuturor datelor sau a unei părți a acestora către utilizator. Formatează și prezintă datele preluate de la model către utilizator.

Controller: controlează interacțiunile dintre Model și View. Funcționează ca o interfață între modelele asociate, vizualizări și dispozitivele de intrare.



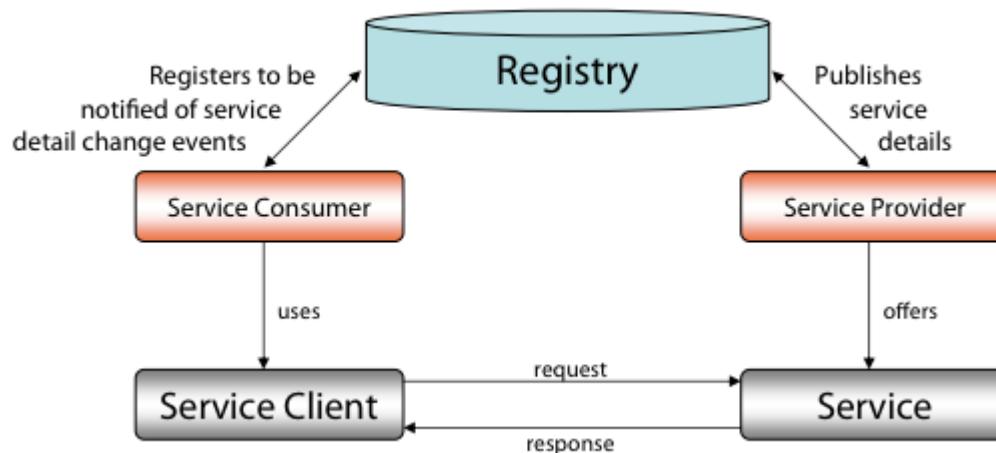
2. Arhitectura orientată pe servicii (SOA - Service Oriented Architecture)

- Aplicațiile moderne sunt construite din componente software bazate pe standarde de interacțiune:
 - Simple Object Access Protocol (SOAP) și
 - Java Platform Enterprise Edition (Java EE).
- SOA se bazează pe un **mecanism de cerere/răspuns** convențional: un consumator de serviciu invocă un furnizor de serviciu prin rețea și așteaptă până când se va realiza operația la furnizor.



SOA- Service Oriented Architecture

- SOA divide o aplicație în două părți:
 - **Un coordonator de serviciu**, care reprezintă funcționalitatea la utilizator și
 - **Furnizorii de servicii**, care implementează funcționalitatea.
- În timp ce coordonatorul trebuie să fie unic pentru o aplicație particulară, un serviciu poate fi reutilizat și partajat de multiple aplicații compozite.
- Coordonatorul serviciului, în mod explicit, specifică și invocă serviciile dorite.



Modele de analiza și modele de proiectare

ANALIZA

Diagrame de clase

Diagrame de cazuri de utilizare

Diagrame de secvențe

Descrieri ale cazurilor de utilizare

Diagrame de stări

Diagrame de activități



PROIECTARE

Diagrame de pachete

Diagrame de clase - proiectare

Diagrame de secvențe

Schema bazei de date

Interfața cu utilizatorul

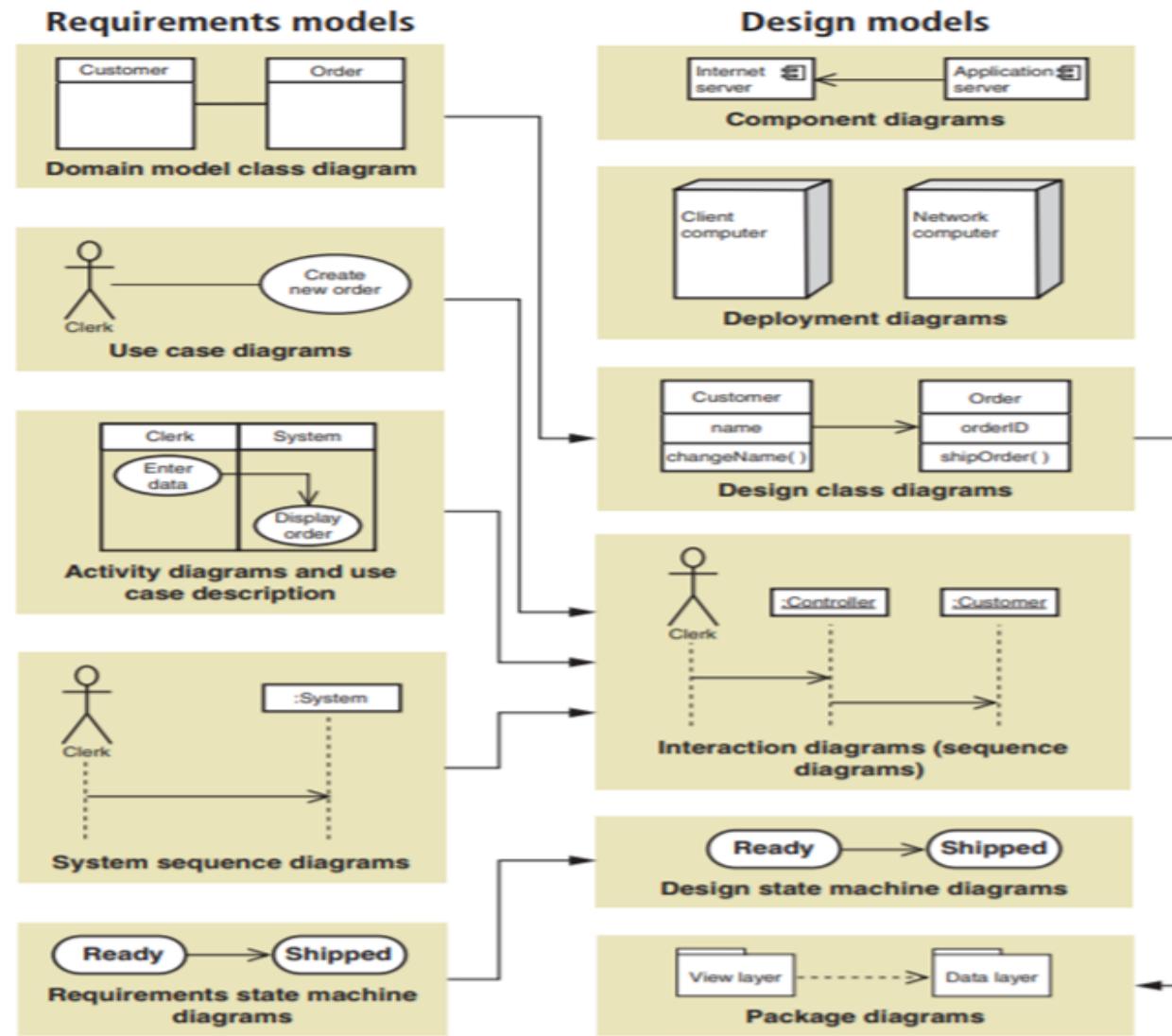
Diagrama de componente

Diagrama de desfășurare

Securitatea și controlul sistemului



Modele de analiza și modele de proiectare



Scopul modelării

- După analiza sistemului, trecem la proiectarea sistemului pentru a:
 - Facilitarea înțelegерii corecte a sistemului;
 - Eliminarea informațiilor despre redundanță operațională;
 - Reducerea timpului de execuție pentru diverse interogări și actualizări ale bazei de date;
 - Reducerea timpului de așteptare al clientului.
- Odată proiectată arhitectura sistemului se poate trece la reiterarea și îmbunătățirea modelului de analiză.



Realizarea cazurilor de utilizare

- **Proiectarea și implementarea cazurilor de utilizare**
 - **Diagrama de secvență** - este extinsă pornind de la diagrama de secvență a sistemului, adăugând un controller și clase de domeniu;
 - **Diagrama claselor de proiectare** - este extinsă pornind de la modelul claselor de domeniu și este actualizată pe baza diagramelor de secvență
 - Mesajele (apelurile) primite de un obiect ar trebui să devină metode ale clasei respective
 - **Definițiile clasei** - sunt scrise în limbajul de programare ales pentru implementarea claselor de proiectare și controller.
 - **Clase de interfață utilizator** - formulare sau pagini adăugate pentru a gestiona interfața dintre actori și clasele de tip controller.
 - **Clasele de acces BD** - acestea sunt adăugate pentru a gestiona cererile pentru interogarea sau salvarea datelor în DB
- **Proiectare detaliată a modelului static**
 - O versiune detaliată a diagramei de clase este proiectată, incluzând **vizibilitatea de navigare** pentru asociații.
 - Pe baza diagramelor de secvență, semnătura metodelor clasei este completată
 - Solutia este partitionată în pachete (elemente de grupare), după cum se consideră necesar



Proiectarea diagramei de clase

Stereotipurile sunt adăugate pentru clase

- **Clase persistente** - o clasă ale cărei obiecte trebuie să existe după oprirea sistemului
- **Clasa entitate** - un identificator pentru o clasă din domeniul problemei de business
- **Limită / clasa de vizualizare** - o clasă situată la limita automată a unui sistem, cum ar fi formular de intrare sau o pagină Web
- **Clasa de control** - o clasă care mediază între clase de graniță și entitate, acționând ca un panou de control între nivelul utilizatorului (vizualizare) și nivelul de afaceri
- **Clasa de acces la date** - o clasă care este utilizată pentru a primi sau trimite date dintr-o bază de date



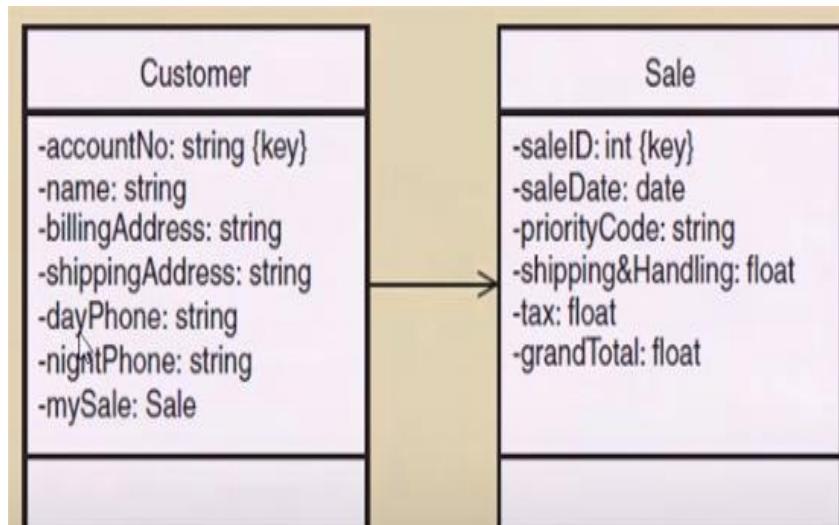
Diagrama detaliata a claselor

- Se parcurg pe rand cazurile de utilizare
- Se aleg clasele domeniului care sunt implicate in cazul de utilizare; se verifica preconditiile si postconditiile pentru completarea acestora
- Se adauga o clasa controller care sa fie responsabila pentru cazul de utilizare
- Se determina cerintele de vizibilitate a navigarii
- Se completeaza atributele fiecarei clase cu vizibilitate si tip
- Observatie: de multe ori asocierile si multiplicitatile sunt eliminate din diagrama, pentru a pune accentul pe navigare, dar ele se pot pastra



Vizibilitatea navigarii

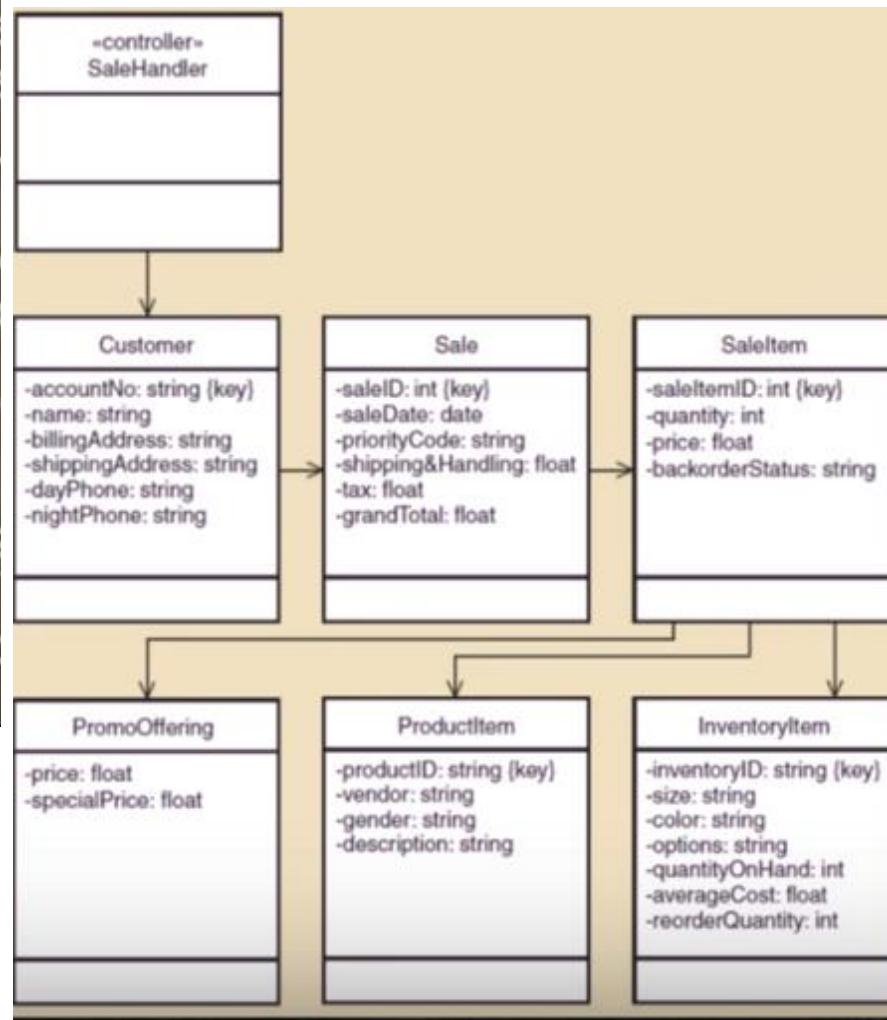
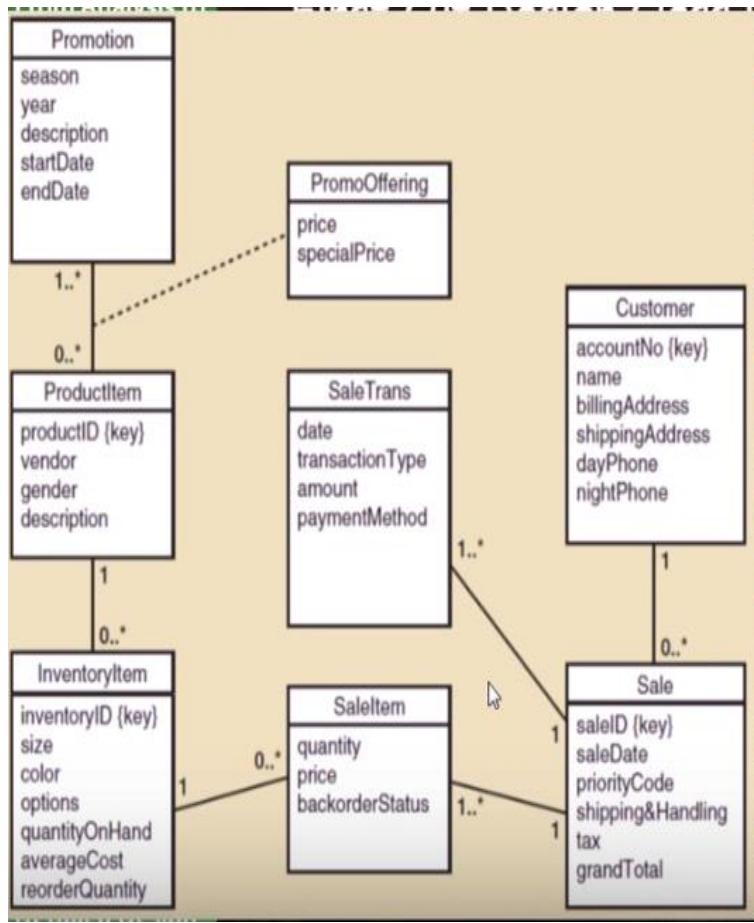
- Abilitatea unui obiect de a vedea și interacționa cu alt obiect
- Realizată prin adăugarea într-o clasă a unei variabile referință la obiect
- Apare ca o săgeată pe capatul asocierei – Clientul poate vedea și interacționa cu Vanzare



Reguli pentru navigabilitate

- Asocierile unu la multi care indica o relatie superior-subordonat asigura navigarea de la superior la subordonati
- Asocierile obligatorii in care obiectele dintr-o clasa nu pot sa existe fara obiectele din alta clasa asigura de obicei navigarea de la clasa mai independenta la cea mai dependenta
- Cand un obiect are nevoie de informatii de la un alt obiect, ar putea fi nevoie de o sageata de navigare



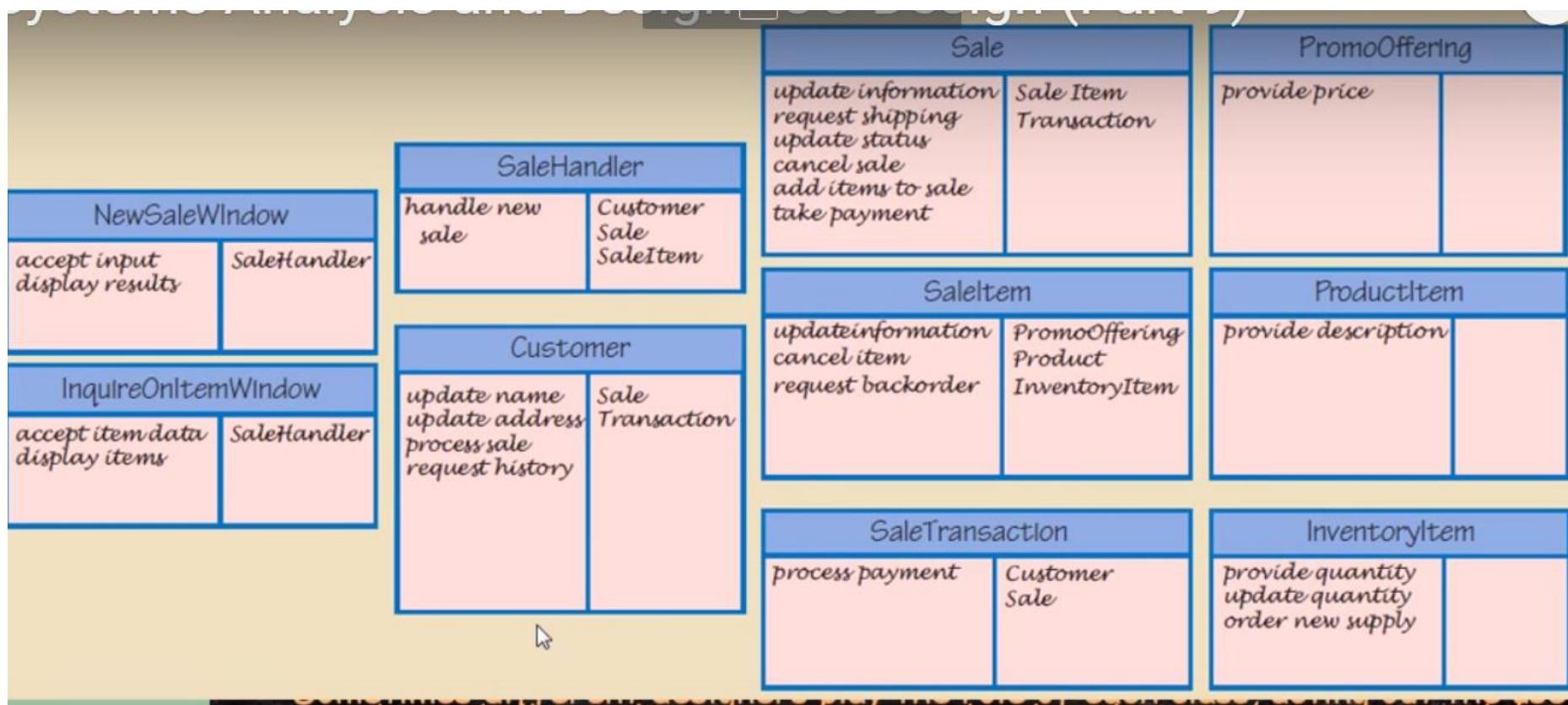


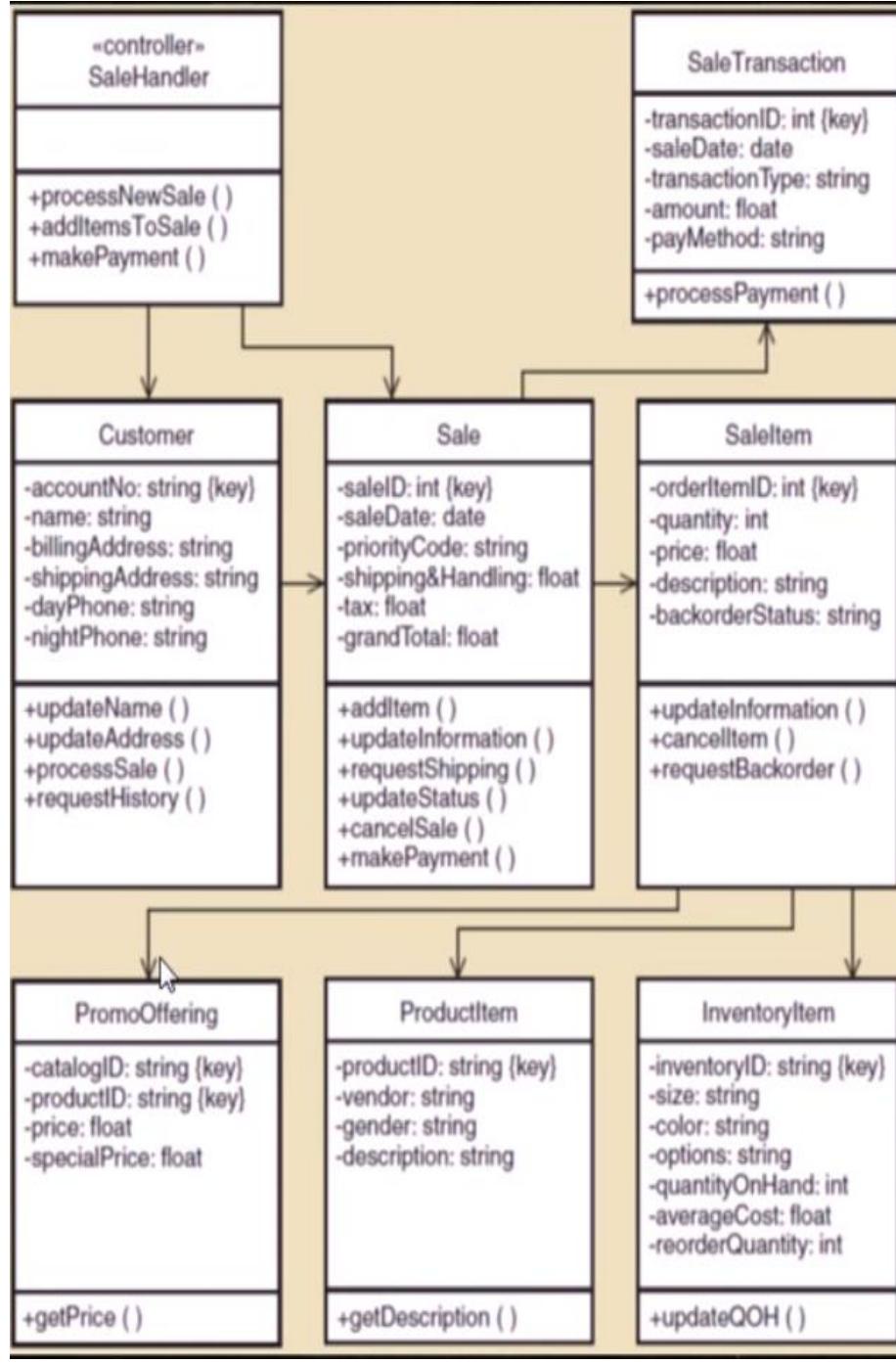
Metodele claselor

- Se poate folosi tehnica **CRC – Class, Responsibility, Collaboration cards**
 - Care sunt responsabilitatile unei clase si cum colaboreaza cu alte clase pentru a realiza cazul de utilizare
 - Se obtin prin brainstorming
- Se pot folosi **diagramele de secventa detaliate** – fiecare mesaj receptionat de un obiect al unei clase trebuie sa aiba in corespondenta o metoda in clasa respectiva



Exemplu de carduri CRC





Proiectarea interfețelor

- După completarea diagramelor de cazuri de utilizare și elaborarea primei versiuni stabile de diagrame de clasă și interacțiune, se recomandă implementarea unui **prototip al interfeței sistemului**.
- Acest prototip se numește **prototip de interfață**, deoarece are rolul de:
 - Rafina relațiile dintre actori și clase de interfață;
 - Obține feedback de la client (client) cu privire la aspectul vizual al aplicației.



Specificarea cerințelor interfeței

- Specificațiile interfeței includ detalii despre ceea ce este necesar pentru ca actorul să interacționeze cu sistemul: nume, titlu și câteva informații descriptive despre modul în care interfața este utilizată de actor.
- Astfel, un document cu specificații de interfață include:
 - Obiectivele actorului în interacțiunea cu sistemul (documentate și în cazurile de utilizare),
 - Lista interfețelor (bazate pe varietatea și numărul de interacțiuni pe care un actor le va avea cu sistemul) și
 - Navigare și dependențe (bazate pe fluxurile de proces documentate în cazuri de utilizare, dar acum revizuite pe baza interfețelor).
- Standardul UML joacă un rol minim în modelarea unei UI. Nu există o diagramă UML care să modeleze o interfață.
- UI-urile pot fi specificate folosind şabloane



Şablonul de specificaţii a interfeţei

- **Identifier interfaţă**
< Aceasta reprezintă numărul și denumirea interfeței>
- **Actori**
< O listă a actorilor care interacţionează cu sistemul prin această interfaţă>
- **Cazuri de utilizare**
< Lista cazurilor de utilizare în care apare această interfaţă>
- **Scurtă descriere**
< Descrierea în cîteva rânduri a interfeței>

User Interface Identifier: UI10-PatientRegistrationForm

Actors: A10-Patient, A80-Administrator

Use Cases: UC10-RegistersPatient; UC12-MaintainsPatientDetails

Short Description: This UI enables the creation of registration details for a first-time patient at the hospital. This same UI also enables maintenance of a patient's registration details. Specific registration details of the patient for this interface are available in the respective use cases.



Proiectarea interfețelor

- Primul pas - investigarea așteptărilor actorilor referitor la interfață prin completarea **chestionarelor** specifice constând din următoarele întrebări:
 - Ce nivel de pregătire (informatică) trebuie să aibă actorul pentru a realiza o anumită funcționalitate?
 - Are actorul experiență de lucru în medii bazate pe ferestre?
 - Are actorul experiență în utilizarea altor sisteme automate de modelare a proceselor?
 - Este necesar să consulte documente / cataloage în paralel cu utilizarea aplicației?
 - Actorul dorește să implementeze facilități de „salvare / restaurare”?



Proiectarea interfețelor

- Obiectivele prototipului sunt:
 - Stabilirea cerințelor interfeței pentru funcționalitățile cheie ale aplicației;
 - Aceasta demonstrează clientului (într-o formă vizuală) că cerințele proiectului au fost bine înțelese și pot fi realizate;
 - Începerea fazei de dezvoltare a elementelor standard ale interfeței.



Detalii pentru proiectarea interfeței

- Fiecare interfață are datele sale descrise ca atribute în cadrul claselor / programelor care manipulează și afișează aceste date. Aceste interfețe sunt proiectate și dezvoltate iterativ. De exemplu, o schiță inițială în prima iteratăie, nu oferă detalii specifice de proiectare, cum ar fi culoarea, dimensiunea căsuțelor de text și aşa mai departe.

The diagram shows a wireframe of a 'PATIENT REGISTRATION FORM' with various input fields and sections. Dashed arrows point from specific form elements to corresponding design and validation requirements listed on the left and right sides.

PATIENT REGISTRATION FORM

Annotations:

- Name and optional form/screen identifier
- Registration No. is assigned by the system
- Address validation from an external service
- Separate section for another category of details
- Drop-down menus (or Radials) where possible to reduce data entry errors.
- Logo of organization/system
- Provide date selection through calendar to reduce format and accuracy errors
- Basic name validation (e.g., no numbers)
- In addition to previously mentioned date validation, ensure birthdate is reasonable (e.g., Age > 15)
- Phone number validation; country code? (based on use cases)
- Standardize buttons and their locations for the system



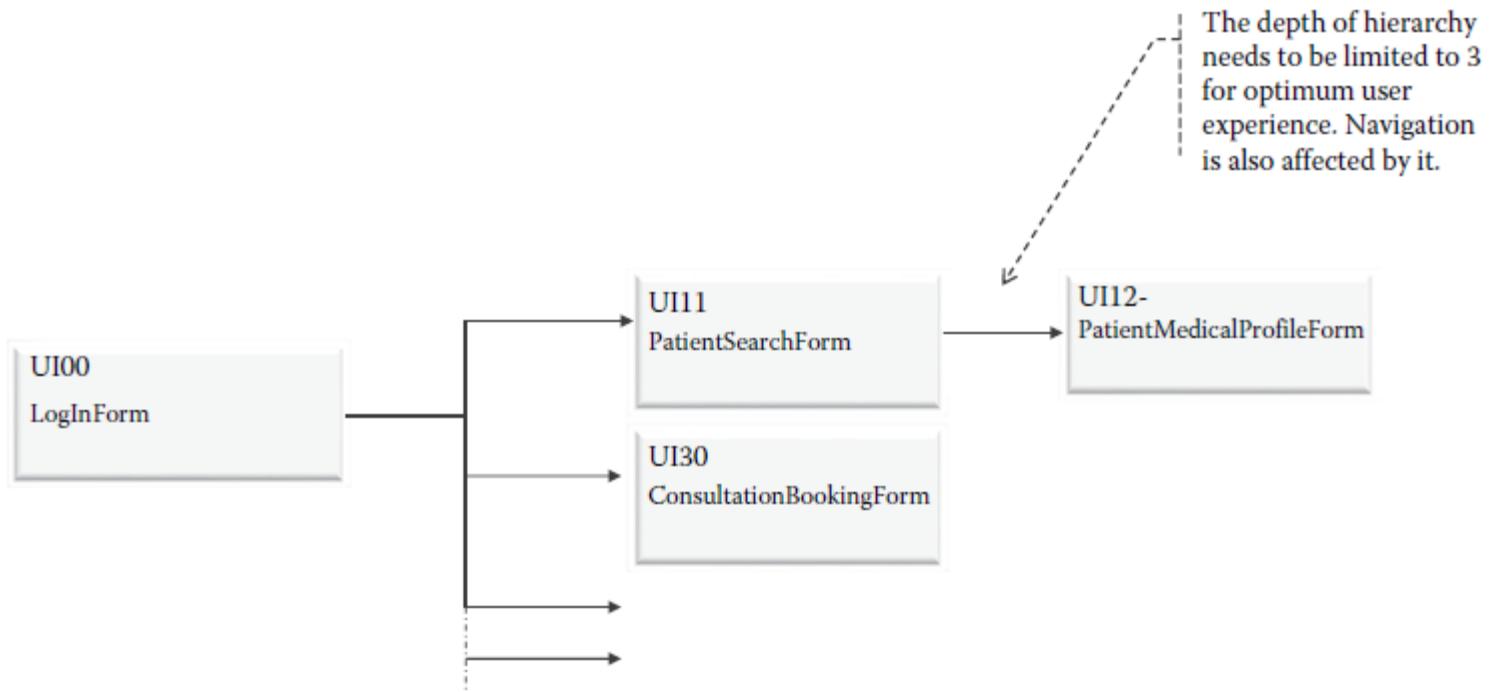
Proiectarea interfețelor

- Hărți cu structura ecranului (diagrame) sunt utilizate pentru a descrie fluxul aplicației urmând principalele moduri de utilizare.
- Reprezentare:
 - Forme pătrate pentru reprezentarea ferestrelor modale (necesită un răspuns al utilizatorului pentru a continua o activitate).
 - Forme pătrate cu colțuri rotunjite pentru reprezentarea ferestrelor nemodale
- Direcția de trecere arată calea de navigare a ferestrei.

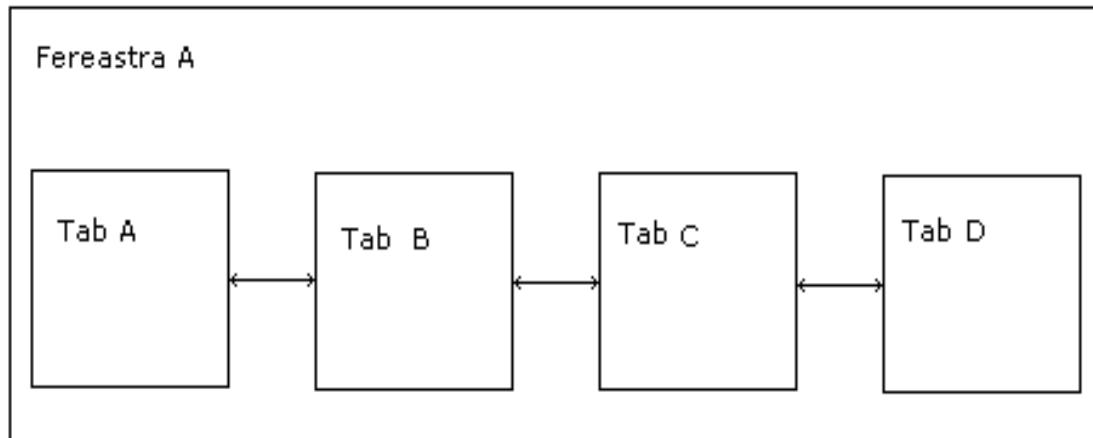


Specificarea fluxului de interfețe utilizator - harta de navigare pentru un doctor

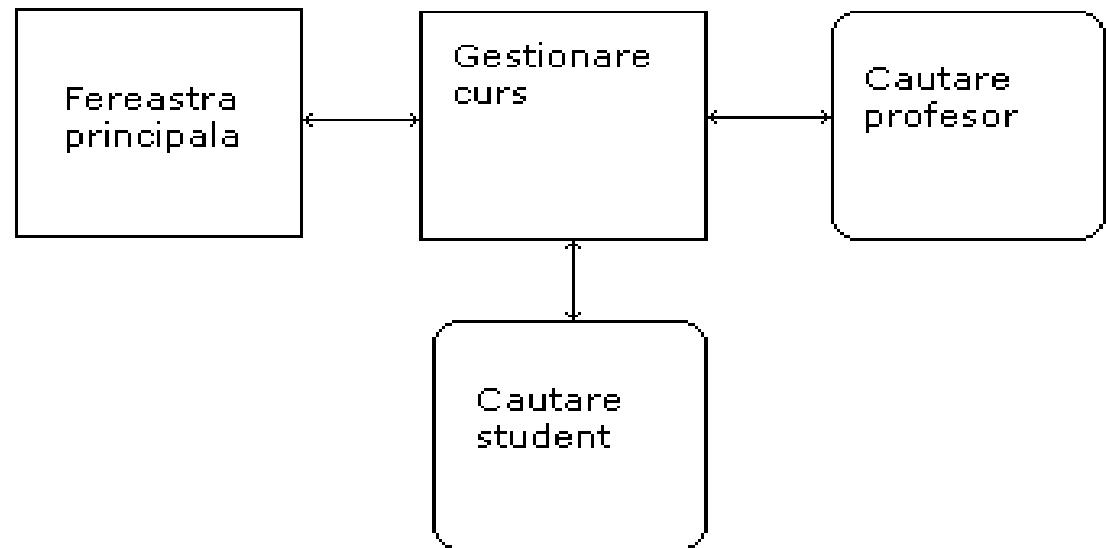
- Diagramele de flux UI, numite ocazional și storyboard-uri, diagrame de navigare a ecranului sau hărți de navigare, modelează relațiile și dependențele dintre interfețele utilizator. Deoarece UML oferă o diagramă de activități ca diagramă de flux, putem crea această hartă de navigare. Ecranele sunt reprezentate de obiecte în această diagramă de activitate.



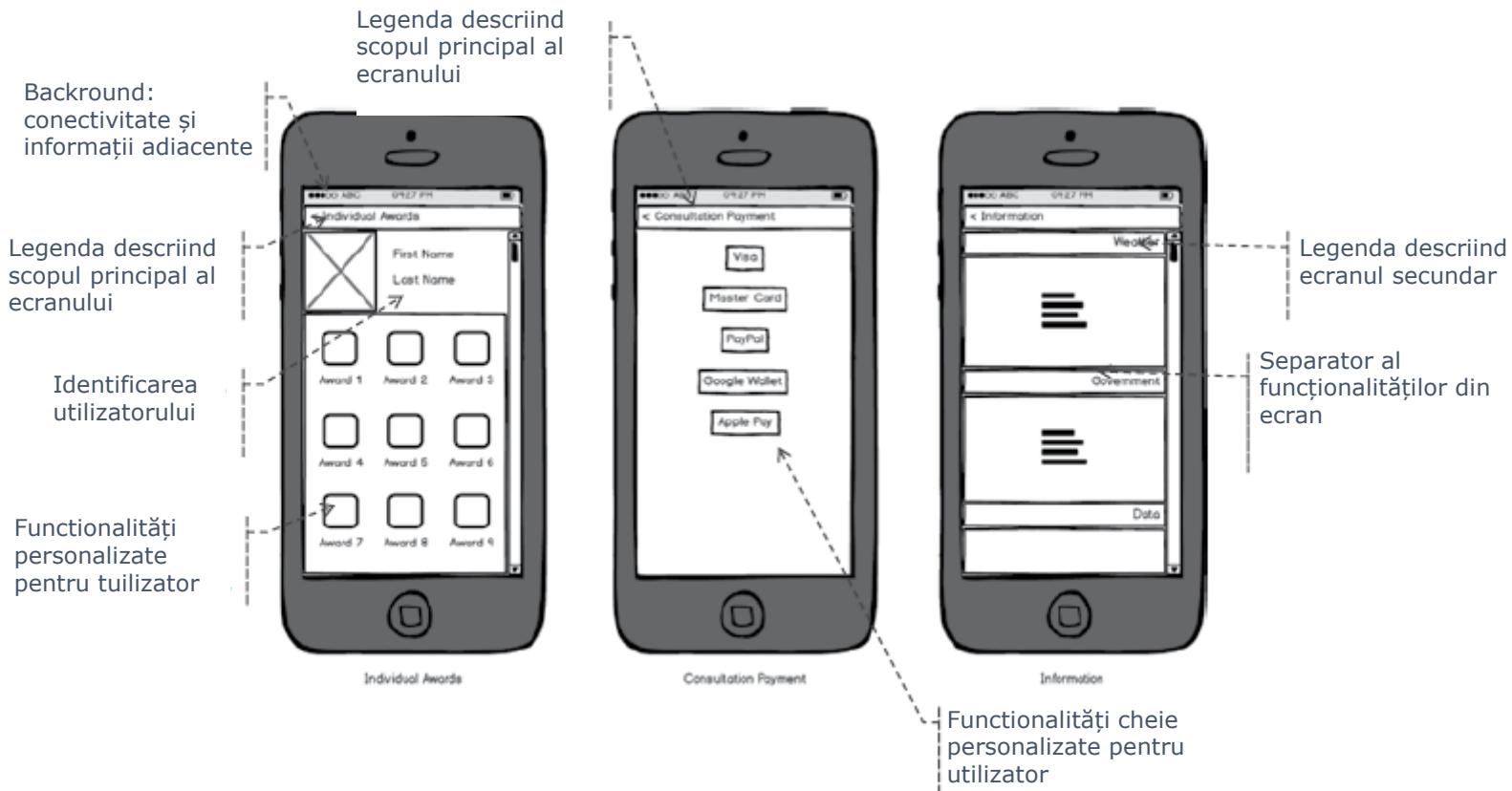
Ferestre care conțin tab-uri



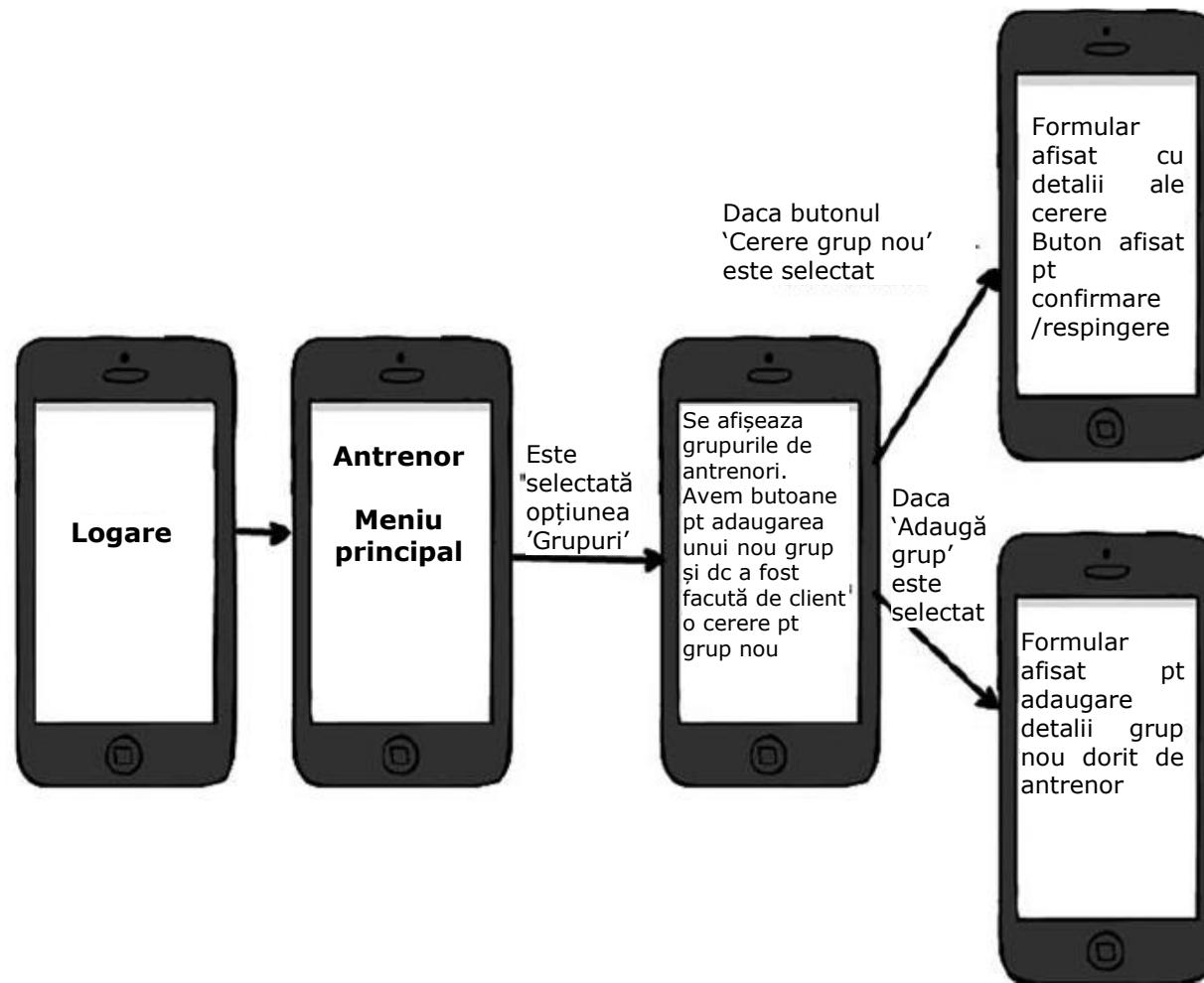
Diagramă de structură a ecranului



Proiectarea interfețelor pentru aplicații mobile cu machete



Proiectarea interfețelor pentru aplicații mobile cu storyboard-uri



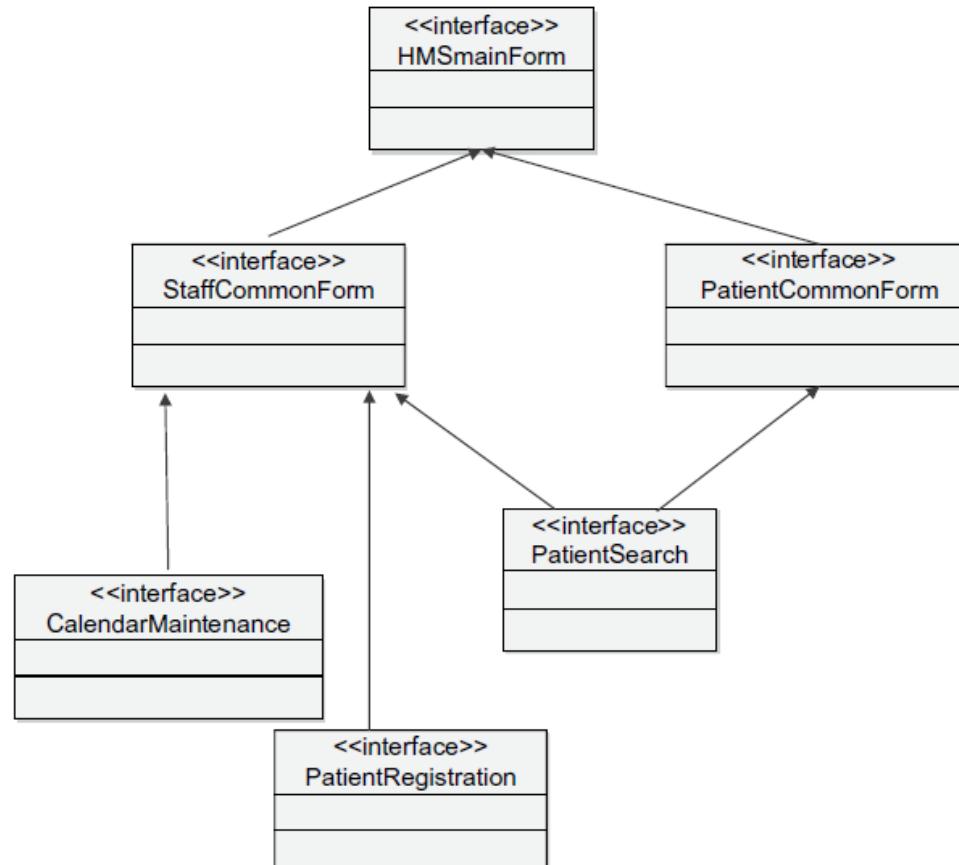
Considerații privind designul interfeței cu utilizatorul

- Analistul de afaceri este inițial responsabil pentru specificațiile UI. Aceste specificații sunt analizate în timpul proiectării UI pentru a crea clase de interfață detaliate și implementabile.
- GUI-urile rezultate pot fi apoi organizate în funcție de actorii (utilizatorii) care le vor folosi.
- GUI-urile sunt de obicei derive din suportul grafic existent oferit de mediul de dezvoltare: ferestre, bare de defilare și butoane radio etc. Un obiect de tip FORM este de obicei disponibil și oferă cele mai multe funcționalități GUI.
- Interfețele pot fi, de asemenea, create și reutilizate de către designeri: o clasă de interfață comună este creată care reprezintă fereastra principală pentru interacțiunea actorilor. Apoi, restul interfețelor necesare aceluiași actor pot moșteni funcționalitatea comună oferită de clasa de interfață comună



Derivarea modelării interfețelor: ierarhia de moșteniri

- Toate clasele de graniță sunt derivate dintr-o clasă principală **HMSmainForm** (bazată pe clase furnizate de mediul de dezvoltare). Clasele specifice actorului care furnizează interfețele comune pentru actor sunt **StaffCommonForm** și **PatientCommonForm**.



Treisprezece principii pentru proiectarea modului de afişare

- Christopher Wickens și colab. au definit 13 principii ale designului în cartea lor *An Introduction to Human Factors Engineering* (2004)
- pot fi utilizate pentru proiectarea eficientă a modului de afişare.

BENEFICII POTENȚIALE

- o reducere a erorilor și a timpului de pregătire necesar,
- o creștere a eficienței,
- o creștere a satisfacției utilizatorilor

Sunt grupate în 4 categorii



a. Principii bazate pe percepție

1. *Faceți afișajele lizibile (sau audibile).* Legibilitatea unui afișaj este critică și necesară pentru proiectarea unui afișaj utilizabil.
2. *Evitați limitele absolute ale judecății.* Nu cereți utilizatorului să determine nivelul unei variabile pe baza unei singure variabile senzoriale (de exemplu, culoare, dimensiune, sunet).
3. *Procesare de sus în jos (top-down).* Semnalele sunt cel mai probabil percepute și interpretate în conformitate cu ceea ce se așteaptă pe baza experienței utilizatorului. Dacă un semnal este prezentat contrar așteptării utilizatorului, poate fi necesară prezentarea mai multor dovezi fizice ale semnalului pentru a fi înțeles corect.
4. *Cresterea redundanței.* Dacă un semnal este prezentat de mai multe ori, este mai probabil să fie înțeles corect. Acest lucru se poate realiza prin prezentarea semnalului în forme fizice alternative (de exemplu, culoare și formă, voce și imprimare etc.), deoarece redundanța nu implică repetarea..
5. *Asemănarea provoacă confuzie: Utilizați elemente distințe.* Semnalele care par a fi similare vor fi probabil confundate. Raportul de caracteristici similare la caracteristici diferite face ca semnalele să fie similare. De exemplu, A423B9 este mai asemănător cu A423B8 decât 92 este cu 93.



b. Principiile modelului mental

6. *Principiul realismului pictural.* Un afișaj ar trebui să arate ca variabila pe care o reprezintă (de exemplu, temperatura ridicată pe un termometru arătat ca un nivel vertical mai mare). Dacă există mai multe elemente, ele pot fi configurate într-o manieră care ar arăta la fel ca în mediul reprezentat.

7. *Principiul piesei în mișcare.* Elementele în mișcare ar trebui să se deplaseze într-un model și o direcție compatibile cu modelul mental al utilizatorului cu privire la modul în care se mișcă efectiv în sistem. De exemplu, elementul care se mișcă pe un altimetru ar trebui să se deplaseze în sus, odată cu creșterea altitudinii.



c. Principii bazate pe atenție

8. *Minimizarea costului accesului la informații sau al interacțiunii.* Atunci când atenția utilizatorului este redirecționată de la o locație la alta pentru a accesa informațiile necesare, există un cost asociat în timp sau efort. O interfață ar trebui să reducă la minimum acest cost, permitând localizarea surselor frecvent accesate în cea mai apropiată poziție posibilă. Cu toate acestea, nu trebuie sacrificată lizibilitatea adecvată pentru a reduce acest cost.

9. *Principiul compatibilității proximității.* Atenție împărțită între două surse de informații poate fi necesară pentru finalizarea unei sarcini. Aceste surse trebuie să fie integrate mental și sunt definite pentru a avea o proximitate mentală strânsă. Costurile de acces la informații ar trebui să fie scăzute, ceea ce poate fi obținut în mai multe moduri (de exemplu, apropierea, creare unei legături prin culori, modele, forme etc.). Cu toate acestea, apropierea excesivă a elementelor de afișat poate fi dăunătoare provocând prea multă aglomerație.

10. *Principiul resurselor multiple.* Un utilizator poate prelucra mai ușor informațiile prezentate prin diferite resurse. De exemplu, informațiile vizuale și auditive pot fi prezentate simultan, în loc să fie prezinte exclusiv vizual sau auditiv.



d. Principiile bazate pe memorie

11. *Înlocuiți memoria cu informații vizuale: cunoștințe din lume.* Un utilizator nu trebuie să rețină informații importante doar în memoria de lucru sau să le recupereze din memoria pe termen lung. Un meniu, o listă de verificare sau un alt afișaj poate ajuta utilizatorul ușurând utilizarea memoriei sale. Utilizarea cunoștințelor memorate de un utilizator și cunoștințele din lume trebuie să fie echilibrate pentru un design eficient.

12. *Principiul ajutorului predictiv.* Acțiunile proactive sunt de obicei mai eficiente decât acțiunile reactive. Un afișaj ar trebui să încerce să eliminate sarcinile cognitive care necesită resurse și să le înlocuiască cu sarcini perceptive mai simple pentru a reduce utilizarea resurselor mentale ale utilizatorului. Acest lucru va permite utilizatorului să se concentreze asupra condițiilor actuale și să ia în considerare posibile condiții viitoare. Un exemplu de ajutor predictiv este un indicativ rutier care afișează distanța până la o anumită destinație.

13. *Principiul consecvenței.* Obiceiurile vechi de la alte afișaje se transferă cu ușurință pentru a sprijini procesarea de afișaje noi dacă sunt proiectate în mod consistent. Memoria unui utilizator pe termen lung va declanșa acțiuni care se așteaptă să fie adecvate. Un design trebuie să accepte acest fapt și să utilizeze consecvență între diferite afișaje



ACADEMIA DE STUDII ECONOMICE BUCUREŞTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

PROIECTAREA SISTEMELOR INFORMATICE

- CURS 11 -

BUCUREŞTI
2020-2021

Proiectarea sistemelor informatice

– partea II –

Cuprins

- ✓ Proiectarea interfețelor
- ✓ Proiectarea bazei de date
- ✓ Proiectarea diagramelor de implementare



Proiectarea interfețelor

- După completarea diagramelor de cazuri de utilizare și elaborarea primei versiuni stabile de diagrame de clasă și interacțiune, se recomandă implementarea unui **prototip al interfeței sistemului**.
- Acest prototip se numește **prototip de interfață**, deoarece are rolul de:
 - Rafina relațiile dintre actori și clase de interfață;
 - Obține feedback de la client (client) cu privire la aspectul vizual al aplicației.



Specificarea cerințelor interfeței

- Specificațiile interfeței includ detalii despre ceea ce este necesar pentru ca actorul să interacționeze cu sistemul: nume, titlu și câteva informații descriptive despre modul în care interfața este utilizată de actor.
- Astfel, un document cu specificații de interfață include:
 - Obiectivele actorului în interacțiunea cu sistemul (documentate și în cazurile de utilizare),
 - Lista interfețelor (bazate pe varietatea și numărul de interacțiuni pe care un actor le va avea cu sistemul) și
 - Navigare și dependențe (bazate pe fluxurile de proces documentate în cazuri de utilizare, dar acum revizuite pe baza interfețelor).
- Standardul UML joacă un rol minim în modelarea unei UI. Nu există o diagramă UML care să modeleze o interfață.
- UI-urile pot fi specificate folosind şabloane



Şablonul de specificaţii a interfeţei

- **Identifier interfaţă**
< Aceasta reprezintă numărul și denumirea interfeței>
- **Actori**
< O listă a actorilor care interacționează cu sistemul prin această interfaţă>
- **Cazuri de utilizare**
< Lista cazurilor de utilizare în care apare această interfaţă>
- **Scurtă descriere**
< Descrierea în cîteva rânduri a interfeței>

User Interface Identifier: UI10-PatientRegistrationForm

Actors: A10-Patient, A80-Administrator

Use Cases: UC10-RegistersPatient; UC12-MaintainsPatientDetails

Short Description: This UI enables the creation of registration details for a first-time patient at the hospital. This same UI also enables maintenance of a patient's registration details. Specific registration details of the patient for this interface are available in the respective use cases.



Proiectarea interfețelor

- Primul pas - investigarea așteptărilor actorilor referitor la interfață prin completarea **chestionarelor** specifice constând din următoarele întrebări:
 - Ce nivel de pregătire (informatică) trebuie să aibă actorul pentru a realiza o anumită funcționalitate?
 - Are actorul experiență de lucru în medii bazate pe ferestre?
 - Are actorul experiență în utilizarea altor sisteme automate de modelare a proceselor?
 - Este necesar să consulte documente / cataloage în paralel cu utilizarea aplicației?
 - Actorul dorește să implementeze facilități de „salvare / restaurare”?



Proiectarea interfețelor

- Obiectivele prototipului sunt:
 - Stabilirea cerințelor interfeței pentru funcționalitățile cheie ale aplicației;
 - Aceasta demonstrează clientului (într-o formă vizuală) că cerințele proiectului au fost bine înțelese și pot fi realizate;
 - Începerea fazei de dezvoltare a elementelor standard ale interfeței.



Detalii pentru proiectarea interfeței

- Fiecare interfață are datele sale descrise ca atribute în cadrul claselor / programelor care manipulează și afișează aceste date. Aceste interfețe sunt proiectate și dezvoltate iterativ. De exemplu, o schiță inițială în prima iteratăie, nu oferă detalii specifice de proiectare, cum ar fi culoarea, dimensiunea căsuțelor de text și aşa mai departe.

The diagram shows a wireframe of a 'PATIENT REGISTRATION FORM' with various input fields and sections. Dashed arrows point from specific form elements to corresponding design and validation requirements listed on the left and right sides.

PATIENT REGISTRATION FORM

Annotations:

- Name and optional form/screen identifier
- Registration No. is assigned by the system
- Address validation from an external service
- Separate section for another category of details
- Drop-down menus (or Radials) where possible to reduce data entry errors.
- Logo of organization/system
- Provide date selection through calendar to reduce format and accuracy errors
- Basic name validation (e.g., no numbers)
- In addition to previously mentioned date validation, ensure birthdate is reasonable (e.g., Age > 15)
- Phone number validation; country code? (based on use cases)
- Standardize buttons and their locations for the system



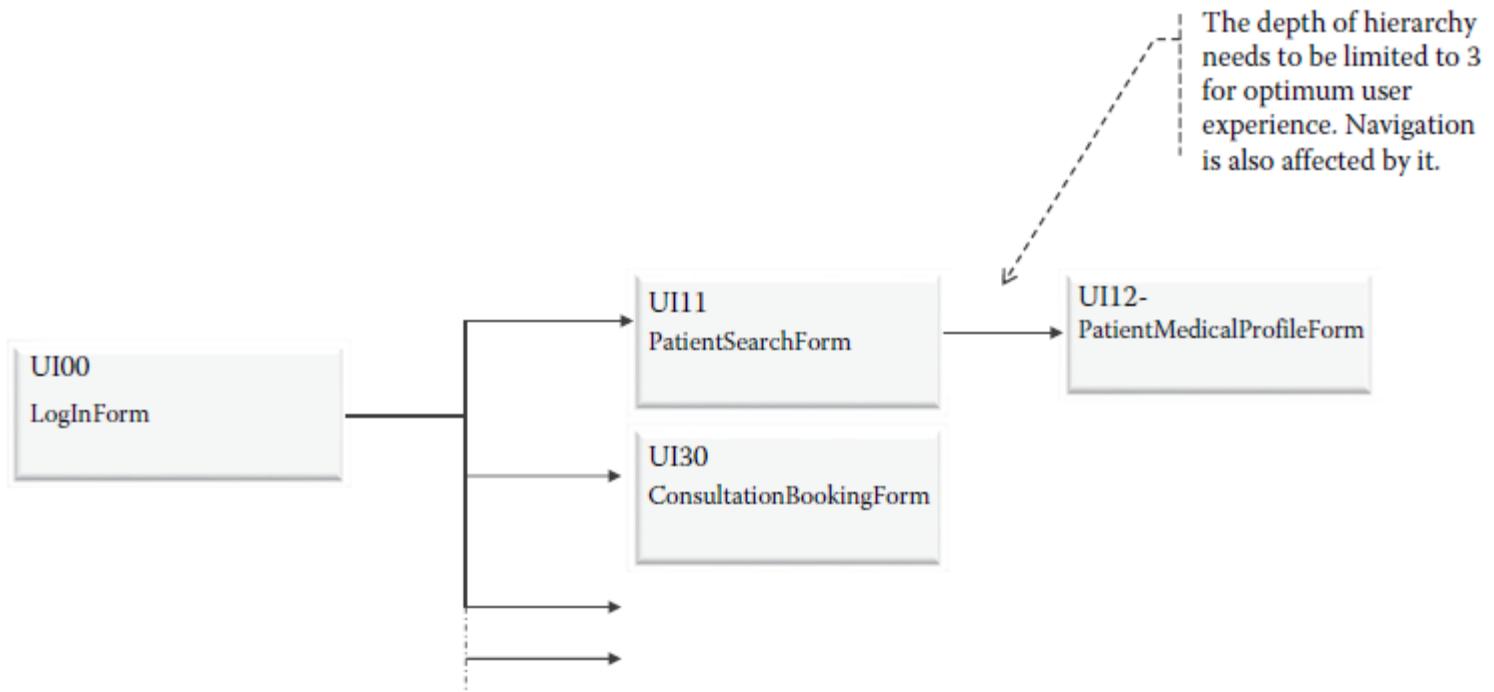
Proiectarea interfețelor

- Hărți cu structura ecranului (diagrame) sunt utilizate pentru a descrie fluxul aplicației urmând principalele moduri de utilizare.
- Reprezentare:
 - Forme pătrate pentru reprezentarea ferestrelor modale (necesită un răspuns al utilizatorului pentru a continua o activitate).
 - Forme pătrate cu colțuri rotunjite pentru reprezentarea ferestrelor nemodale
- Direcția de trecere arată calea de navigare a ferestrei.

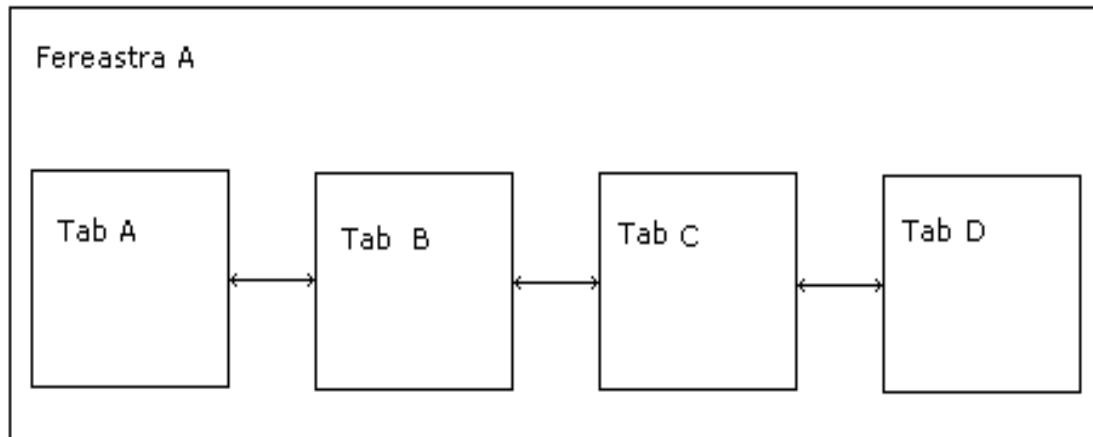


Specificarea fluxului de interfețe utilizator - harta de navigare pentru un doctor

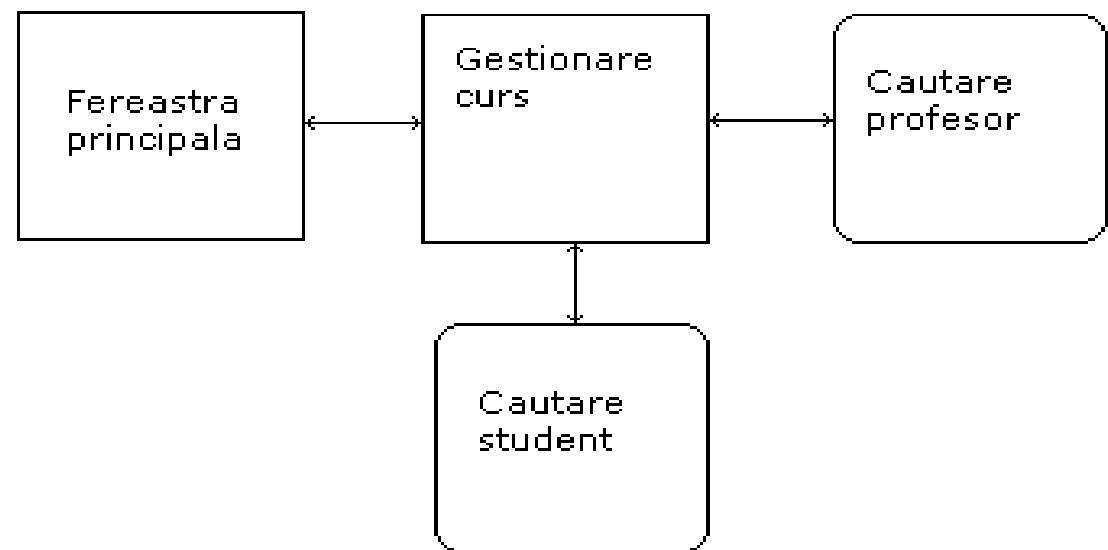
- Diagramele de flux UI, numite ocazional și storyboard-uri, diagrame de navigare a ecranului sau hărți de navigare, modelează relațiile și dependențele dintre interfețele utilizator. Deoarece UML oferă o diagramă de activități ca diagramă de flux, putem crea această hartă de navigare. Ecranele sunt reprezentate de obiecte în această diagramă de activitate.



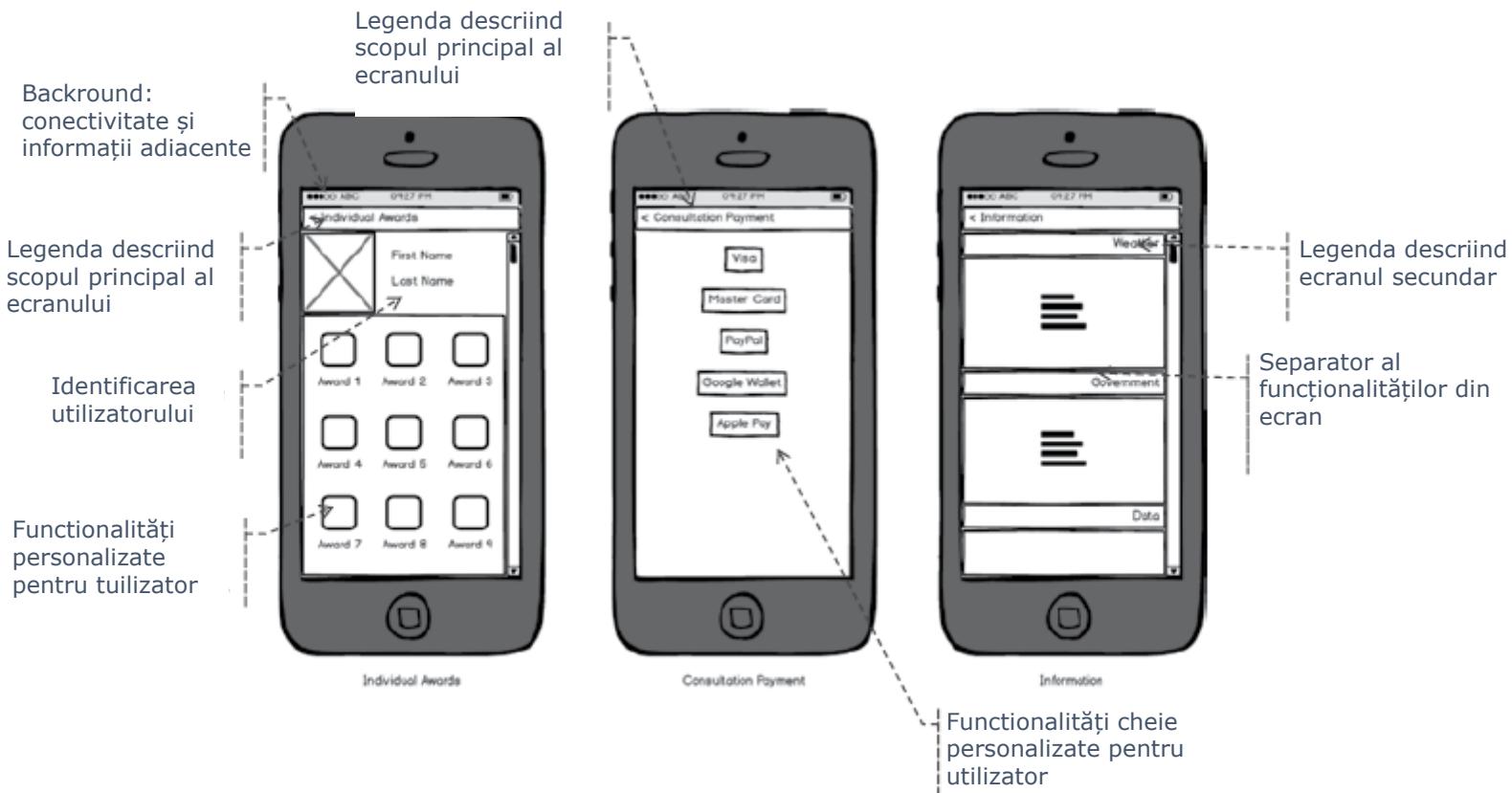
Ferestre care conțin tab-uri



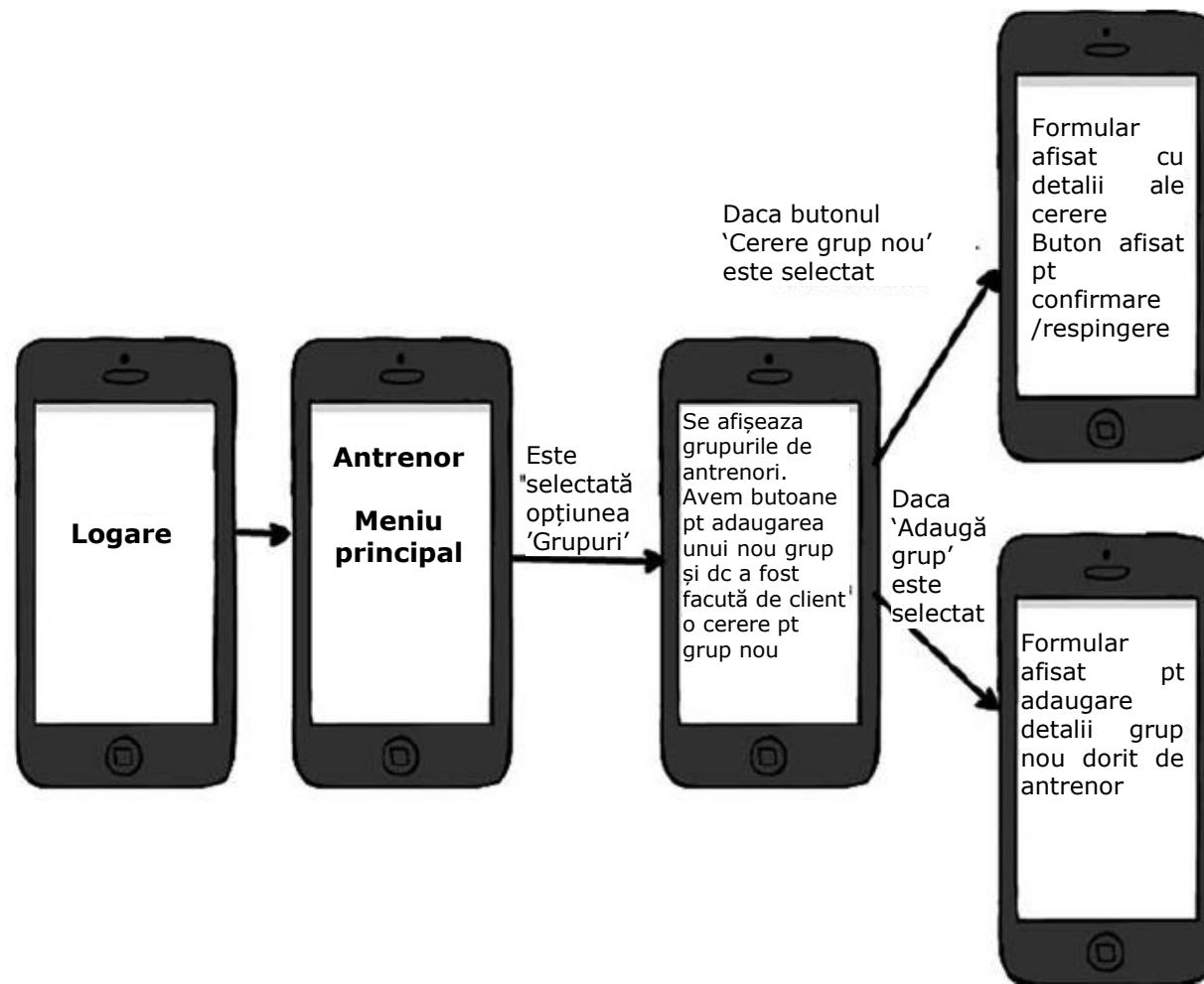
Diagramă de structură a ecranului



Proiectarea interfețelor pentru aplicații mobile cu machete



Proiectarea interfețelor pentru aplicații mobile cu storyboard-uri



Considerații privind designul interfeței cu utilizatorul

- Analistul de afaceri este inițial responsabil pentru specificațiile UI. Aceste specificații sunt analizate în timpul proiectării UI pentru a crea clase de interfață detaliate și implementabile.
- GUI-urile rezultate pot fi apoi organizate în funcție de actorii (utilizatorii) care le vor folosi.
- GUI-urile sunt de obicei derive din suportul grafic existent oferit de mediul de dezvoltare: ferestre, bare de defilare și butoane radio etc. Un obiect de tip FORM este de obicei disponibil și oferă cele mai multe funcționalități GUI.
- Interfețele pot fi, de asemenea, create și reutilizate de către designeri: o clasă de interfață comună este creată care reprezintă fereastra principală pentru interacțiunea actorilor. Apoi, restul interfețelor necesare aceluiași actor pot moșteni funcționalitatea comună oferită de clasa de interfață comună



Treisprezece principii pentru proiectarea modului de afişare

- Christopher Wickens și colab. au definit 13 principii ale designului în cartea lor *An Introduction to Human Factors Engineering* (2004)
- pot fi utilizate pentru proiectarea eficientă a modului de afişare.

BENEFICII POTENȚIALE

- o reducere a erorilor și a timpului de pregătire necesar,
- o creștere a eficienței,
- o creștere a satisfacției utilizatorilor

Sunt grupate în 4 categorii



a. Principii bazate pe percepție

1. *Faceți afișajele lizibile (sau audibile).* Legibilitatea unui afișaj este critică și necesară pentru proiectarea unui afișaj utilizabil.
2. *Evitați limitele absolute ale judecății.* Nu cereți utilizatorului să determine nivelul unei variabile pe baza unei singure variabile senzoriale (de exemplu, culoare, dimensiune, sunet).
3. *Procesare de sus în jos (top-down).* Semnalele sunt cel mai probabil percepute și interpretate în conformitate cu ceea ce se așteaptă pe baza experienței utilizatorului. Dacă un semnal este prezentat contrar așteptării utilizatorului, poate fi necesară prezentarea mai multor dovezi fizice ale semnalului pentru a fi înțeles corect.
4. *Cresterea redundanței.* Dacă un semnal este prezentat de mai multe ori, este mai probabil să fie înțeles corect. Acest lucru se poate realiza prin prezentarea semnalului în forme fizice alternative (de exemplu, culoare și formă, voce și imprimare etc.), deoarece redundanța nu implică repetarea..
5. *Asemănarea provoacă confuzie: Utilizați elemente distințe.* Semnalele care par a fi similare vor fi probabil confundate. Raportul de caracteristici similare la caracteristici diferite face ca semnalele să fie similare. De exemplu, A423B9 este mai asemănător cu A423B8 decât 92 este cu 93.



b. Principiile modelului mental

6. *Principiul realismului pictural.* Un afișaj ar trebui să arate ca variabila pe care o reprezintă (de exemplu, temperatura ridicată pe un termometru arătat ca un nivel vertical mai mare). Dacă există mai multe elemente, ele pot fi configurate într-o manieră care ar arăta la fel ca în mediul reprezentat.

7. *Principiul piesei în mișcare.* Elementele în mișcare ar trebui să se deplaseze într-un model și o direcție compatibile cu modelul mental al utilizatorului cu privire la modul în care se mișcă efectiv în sistem. De exemplu, elementul care se mișcă pe un altimetru ar trebui să se deplaseze în sus, odată cu creșterea altitudinii.



c. Principii bazate pe atenție

8. Minimizarea costului accesului la informații sau al interacțiunii. Atunci când atenția utilizatorului este redirecționată de la o locație la alta pentru a accesa informațiile necesare, există un cost asociat în timp sau efort. O interfață ar trebui să reducă la minimum acest cost, permitând localizarea surselor frecvent accesate în cea mai apropiată poziție posibilă. Cu toate acestea, nu trebuie sacrificată lizibilitatea adecvată pentru a reduce acest cost.

9. Prințipiu compatibilității proximității. Atenție împărțită între două surse de informații poate fi necesară pentru finalizarea unei sarcini. Aceste surse trebuie să fie integrate mental și sunt definite pentru a avea o proximitate mentală strânsă. Costurile de acces la informații ar trebui să fie scăzute, ceea ce poate fi obținut în mai multe moduri (de exemplu, apropierea, creare unei legături prin culori, modele, forme etc.). Cu toate acestea, apropierea excesivă a elementelor de afișat poate fi dăunătoare provocând prea multă aglomerație.

10. Prințipiu resurselor multiple. Un utilizator poate prelucra mai ușor informațiile prezentate prin diferite resurse. De exemplu, informațiile vizuale și auditive pot fi prezentate simultan, în loc să fie prezinte exclusiv vizual sau auditiv.



d. Principiile bazate pe memorie

11. *Înlocuiți memoria cu informații vizuale: cunoștințe din lume.* Un utilizator nu trebuie să rețină informații importante doar în memoria de lucru sau să le recupereze din memoria pe termen lung. Un meniu, o listă de verificare sau un alt afișaj poate ajuta utilizatorul ușurând utilizarea memoriei sale. Utilizarea cunoștințelor memorate de un utilizator și cunoștințele din lume trebuie să fie echilibrate pentru un design eficient.

12. *Principiul ajutorului predictiv.* Acțiunile proactive sunt de obicei mai eficiente decât acțiunile reactive. Un afișaj ar trebui să încerce să eliminate sarcinile cognitive care necesită resurse și să le înlocuiască cu sarcini perceptive mai simple pentru a reduce utilizarea resurselor mentale ale utilizatorului. Acest lucru va permite utilizatorului să se concentreze asupra condițiilor actuale și să ia în considerare posibile condiții viitoare. Un exemplu de ajutor predictiv este un indicativ rutier care afișează distanța până la o anumită destinație.

13. *Principiul consecvenței.* Obiceiurile vechi de la alte afișaje se transferă cu ușurință pentru a sprijini procesarea de afișaje noi dacă sunt proiectate în mod consistent. Memoria unui utilizator pe termen lung va declanșa acțiuni care se așteaptă să fie adecvate. Un design trebuie să accepte acest fapt și să utilizeze consecvență între diferite afișaje



Proiectarea bazei de date

- Se pleacă de la **modelul claselor domeniului**
- Se alege **structura bazei de date**
 - De obicei se lucreaza cu baze de date relaționale, dar pot există platforme care lucrează cu baze de date orientate obiect sau alte sisteme NoSQL
- Se proiectează **arhitectura BD** (distribuită, etc)
- Se proiectează **schema bazei de date**
 - Tabelele și coloanele în relațional
- Se proiectează **restricțiile de integritate** referențială
 - Referințe prin chei externe



Stereotipuri pentru persistență

- Majoritatea obiectelor stereotype *<entity>* trebuie să existe chiar și după închiderea sistemului
- În timp ce un obiect persistent există dincolo de execuția sistemului, un obiect tranzitoriu dispare atunci când sistemul este oprit și este recreat atunci când sistemul este executat din nou. Majoritatea obiectelor *<control>* și *<boundary>* sunt tranzitorii și nu este necesară salvarea acestor tipuri de obiecte.



Mecanismul persistenței - baze de date

- Persistența poate lua diverse forme. Mai jos prezentăm câteva tehnici pentru a face obiectele persistente:
 - Stocarea într-un **fișier flat**. O astfel de stocare poate conține datele utilizate de sistem, dar nu este inteligentă și nu oferă o modalitate de a căuta în date.
 - Stocarea într-un **fișier cu mecanism de acces secvențial indexat** sau într-o bază de date care organizează datele prin indexuri care pot fi utilizate pentru căutarea înregistrărilor de date specifice și actualizarea acestora.
 - Stocarea într-o **bază de date relațională**, care este cea mai potrivită pentru datele de afaceri care pot fi ușor formatare în rânduri și coloane, fiind optimizate astfel pentru căutare.
 - Stocarea într-o **bază de date orientată obiect**, care este mai potrivite pentru informații științifice sau neformatate.
 - Stocarea într-o **bază de date NoSQL**, care poate gestiona documente mari, nestructurate și date care pot fi căutate în mod optional.



Baze de date orientate obiect

- BDOO sunt capabile să stocheze obiecte împreună cu valorile atributelor acestora, operațiile și relațiile lor.
- Structurile obiectului din memorie în timpul execuției pot fi stocate direct „as is” în baza de date. Obiecte binare de dimensiuni mari (BLOB) și datele complexe nestructurate (de exemplu, video și audio) pot fi de asemenea stocate în aceste baze de date fără a fi nevoie de conversie
- De asemenea, stochează relații precum moștenirea, asocierea și agregarea direct în baza de date.



Baze de date NoSQL

- Nu respectă formatul rând-coloană tradițional al unei baze de date cu un limbaj de interogare structurat (SQL), (baza de date relațională), dar este capabil să **gestioneze date nestructurate**.
- Tehnologia pe care se bazează permite **gestionarea volumelor mari de date**.
- Bazele de date NoSQL gestionează o structură de baze de date complicată și federalizată, care se regăsește de obicei în **Cloud**. Structura bazei de date federalizată a bazelor de date NoSQL este de asemenea înțeleasă ca o arhitectură de baze de date distribuite
- De exemplu, o relație **Client-Cont** nu este doar o asociere; **Cont** este o colecție de conturi care aparțin clientului și vor fi încorporate în fiecare **Client**



Bazele de date relaționale

- Majoritatea aplicațiilor comerciale utilizează în continuare **baze de date relaționale**. Acestea oferă mecanisme ideale și mature pentru stocarea, preluarea și gestionarea datelor structurate
- Tabelele sunt **structural** diferite de obiecte și necesită „**traducerea**” claselor în tabele
- Cele mai multe instrumente de modelare bazate pe UML permit arhitectului să **marcheze clasele ca persistente**, ele pot fi apoi folosite de instrumente pentru a crea o schemă inițială a bazei de date relaționale pe baza diagramelor de clase.



Maparea obiectelor domeniului pentru SGBDR

1. Maparea tuturor claselor concrete ale domeniului în tabele. De asemenea dacă o clasă abstractă a domeniului are moștenitori direcți multiplii, mapăm clasa într-o tabelă.
2. Mapăm atributele cu valoare unică în coloane ale tabelei
3. Mapăm metodele în proceduri stocate sau module de program
4. Mapăm agregările și relațiile de asociere cu multiplicitate unu-la-unu într-o coloană care poate stoca cheia tabelei asociate ex: adăugăm o cheie externă tabelei. Facem acest lucru pentru ambele capete ale relației.
5. Mapăm atributele multi-valoare și grupurile repetitive în tabele noi și creăm asocierei 1-n de la tabela originală către cele noi.
6. Mapăm agregările și relațiile de asociere cu multiplicitate mulți-la-mulți într-o nouă tabelă de legătură între cele 2 tabele originale. Copiem cheile primare din ambele tabele în noua tabelă.
7. Pentru relații de agregare și asociere de tip mixt, copiem cheia primară din partea 1 a relației (1..1 sau 0..1) într-o nouă coloană în tabela aferentă laturii mulți (1..* sau 0..*) a relației care poate stoca cheia tabelei de legătură ex: adăugăm p cheie externă tabelei de pe latura multi a relației.
- 8a. Ne asigurăm că cheia primară a subclasei este aceeași ca și cheia primară a superclasei. Multiplicitatea acestei noi asocieri de la subclasă la superclasă ar trebui să fie 1..1. Dacă super clasele pot fi instanțiate, atunci multiplicitatea dinspre superclasă către subclasă este 0..1, altfel este 1..1. Mai mult, o restricție exclusivă (XOR) trebuie să fie adăugată între asocieri. Facem acest lucru pentru fiecare superclasă

SAU

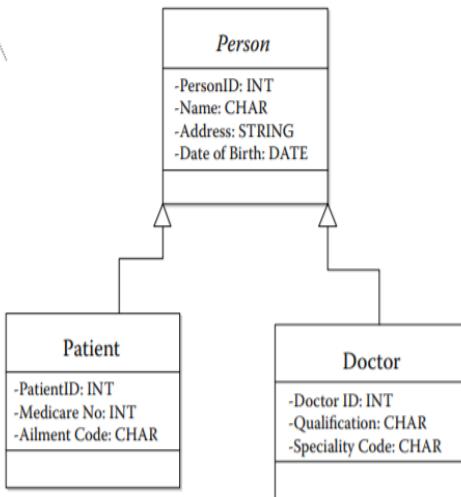
- 8b. Aplatizăm ierarhia de moștenire prin copierea atributelor superclasei în toate subclasele și eliminarea superclasei din model.



Relațiile de moștenire și tabelele relaționale

Care sunt modurile posibile de a mapa Pacient și Doctor?

- O singură tabelă
- 2 tabele, una pt Pacient, alta pt Doctor
- 3 tabele?



a

PERSON TABLE

PersonType	PersonID	Name	Address	MedicareNo	Ailmentcode	Qualification
P	P001	Mark	Parramatta	13254678	PC2003	
D	D001	Bala	Strathfield			M.B.B.S
P	P020	Mariana	Campbelltown	15487962	AS5006	
D	D023	Fiona	Redfern			F.R.C.S

b

PATIENT TABLE

PatientID	Name	Address	MedicareNo	Ailmentcode
P001	Mark	Parramatta	13254678	PC2003
P020	Mariana	Campbelltown	15487962	AS5006

DOCTOR TABLE

DoctorID	Name	Address	Qualification
D001	Bala	Strathfield	M.B.B.S
D023	Fiona	Redfern	F.R.C.S

c

PERSON TABLE

PersonID	Name	Address
P001	Mark	Parramatta
D001	Bala	Strathfield
P020	Mariana	Campbelltown
D023	Fiona	Redfern

PATIENT TABLE

PersonID	PatientID	MedicareNo	Ailmentcode
P001	IP2001	13254678	PC2003
P020	OP2003	15487962	AS5006

DOCTOR TABLE

PersonID	DoctorID	Qualification
D001	DP3005	M.B.B.S
D023	DS2334	F.R.C.S



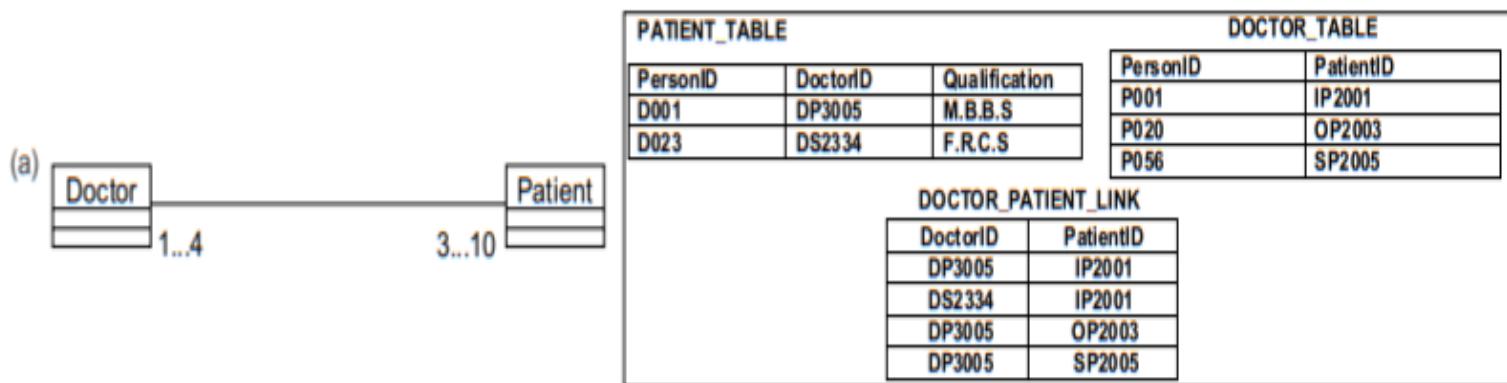
Relațiile de moștenire

- a) Cel mai simplu mod este să mapăm toate atributele din clasa părinte, precum și subclase, pe coloanele unui singur tabel. Spațiu irosit: când un obiect Pacient este stocat în acest tabel anume, acesta ar lăsa coloanele specifice Doctorului necompletate.
- b) Creați tabele pentru toate clasele pentru copii și adăugați-i atributele clasei părinte. Această abordare devine mai dificilă pentru mai multe niveluri de moștenire.
- c) Creați tabele separate atât pentru clasa părinte, cât și pentru copii. Aceste tabele sunt apoi legate cu ajutorul cheii primare a tabelei care reprezintă clasa părinte (PersonID)



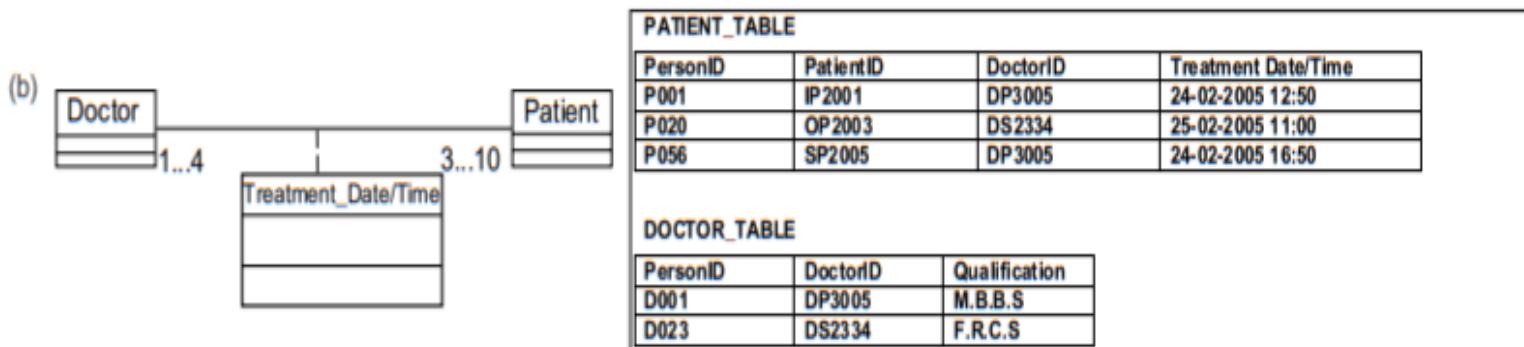
Multiplicități și clase de asociere

- Fiecare clasă este transformată inițial într-o tabelă în baza de date relațională și o nouă tabelă este creată pentru a mapa asocierea; multiplicități mulți-la-mulți nu pot fi implementate direct.



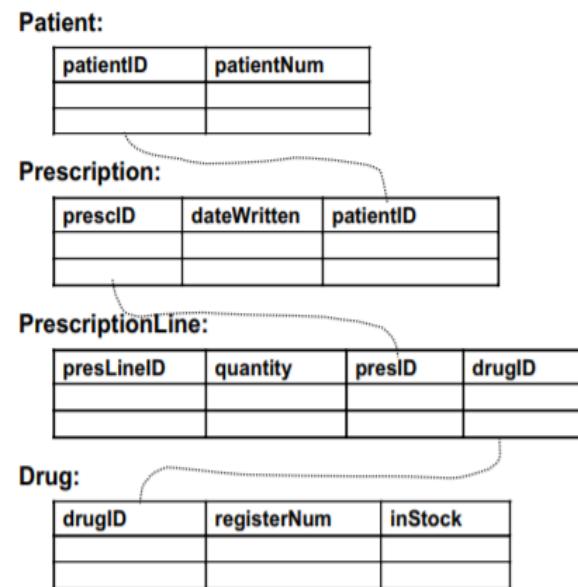
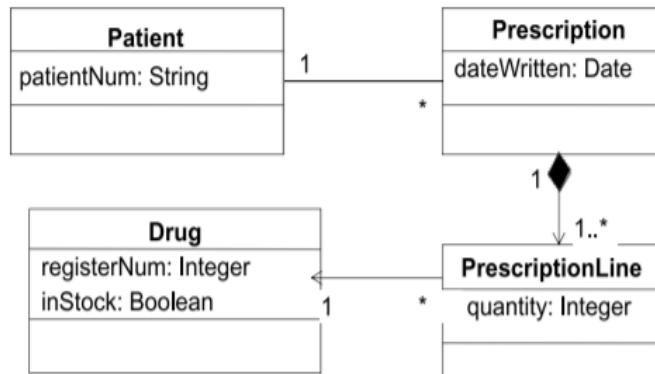
Multiplicități și clase de asociere

- Maparea unei clase de asociere variază în diferite cazuri și depinde de multiplicități.
- Maparea va rezulta în două tabele, cu cheia pentru Doctor adăugată tabelei de stocare a pacienților. Cu toate acestea, clasa de asociere nu este mapată într-o tabelă. În schimb, atributele data / ora sunt anexate tabelei Pacient



Maparea agregărilor

- Ambele clase vor fi mapate în tabele individuale. Cheia clasei de asociere va fi anexată tabeliei corespunzătoare: **PrescriptionLine** va conține **PresID**, cheia tabelii **Prescription**.
- Ori de câte ori un obiect **Prescription** este distrus, trebuie să aveți grijă să eliminați toate obiectele **PrescriptionLine** înrudite.



Diagramele de implementare

- *Diagrama de componentă*
- *Diagrama de desfășurare*

Diagrama de componentă

- O **diagramă de componentă** prezintă dependențele existente între diverse componente software ce compun un sistem informatic.
- Aceste **dependențe** sunt:
 - **statice** - au loc în etapele de compilare sau link-editare
 - **dinamice** - au loc în timpul execuției
- Modeleză **arhitectura de ansamblu** și componentele locale din interiorul acesteia.
 - **Componente** ale sistemului, logice și reutilizabile, care definesc arhitectura sistemului.
 - Interfețe bine definite sau metode publice, care pot fi accesate de alte programe sau dispozitive externe.



Componentă

- O componentă este un modul sau program executabil (cod sursă, cod binar, dll, executabil, script etc) și constă din toate clasele care sunt compilate într-o singură entitate.
- Fiecare componentă este responsabilă pentru un obiectiv clar în cadrul întregului sistem și interacționează numai cu alte elemente esențiale, în funcție de necesitate.
- La un nivel înalt de abstractizare, componente sunt considerate unități autonome încapsulate într-un sistem sau subsistem care furnizează una sau mai multe interfețe. Prin clasificarea unui grup de clase ca și componentă, întregul sistem devine mai modular, deoarece componente pot fi schimbată și reutilizate.
- O componentă este trăsătură ca un dreptunghi cu compartimente optionale organizate vertical.



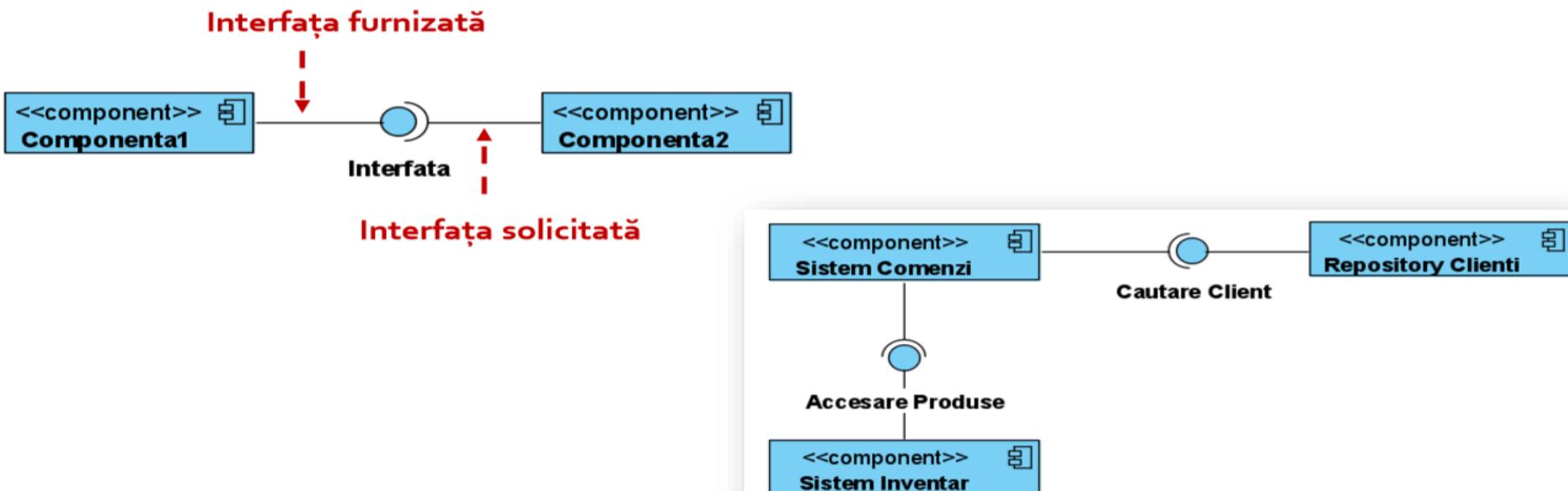
Stereotipuri pentru componente

- Exemple de stereotipuri predefinite pentru componente:
 - <<buildComponent>>
 - <<document>>
 - <<executable>>
 - <<file>>
 - <<library>>
 - <<entity>>
 - <<service>>



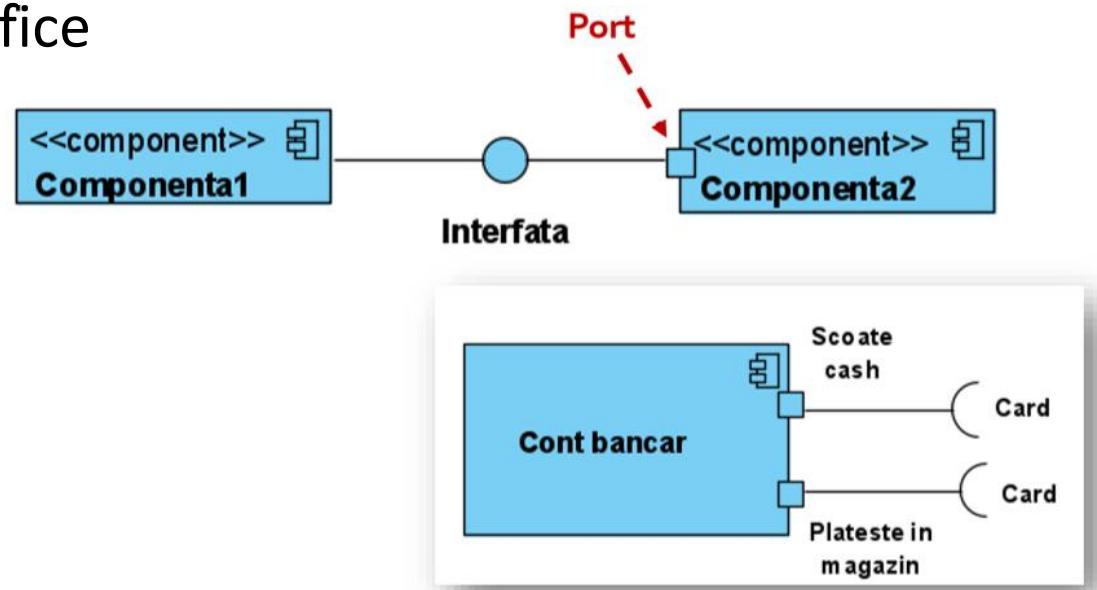
Interfață

- Interfață specifică un contract constând dintr-un set de atribută și operații publice pentru o clasă.
- Există două tipuri de interfețe ale componentelor:
 - Furnizată - oferite de către componentă, se reprezintă cu simbolul unui cerc.
 - Solicitată – necesare interfeței, se reprezintă cu un semicerc.



Portul

- Porturile sunt reprezentate folosind un **pătrat** de-a lungul marginii unei componente.
- Un port este adesea folosit pentru a ajuta la **expunerea interfețelor** necesare și furnizate ale unei componente.
- Este o **fereastră explicită** într-o componentă încapsulată. Toate interacțiunile în și din astfel de componente trec prin porturi.
- Fiecare port **furnizează sau necesită** una sau mai multe interfețe specifice



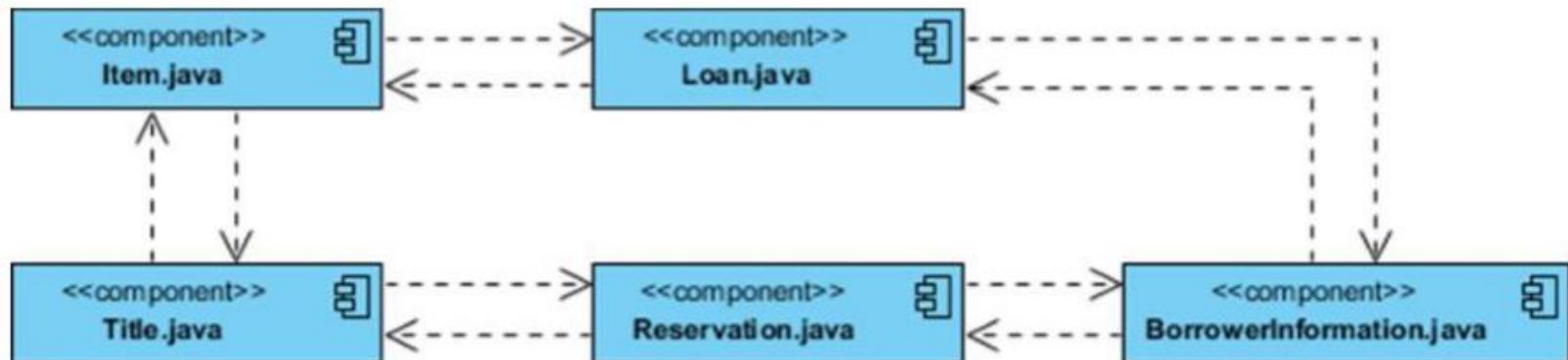
Relații frecvent utilizate în diagrama de componente

- **Dependență**
 - linie îintreruptă îndreptată spre furnizorul componentei
 - clasele incluse în componenta client pot **moșteni, instanția sau utiliza** clasele incluse în componenta furnizorului
 - pot fi, de asemenea, relații de dependență între **componente și interfețe ale altor componente**
- **Relația de compunere** (componente incluse fizic în alte componente).



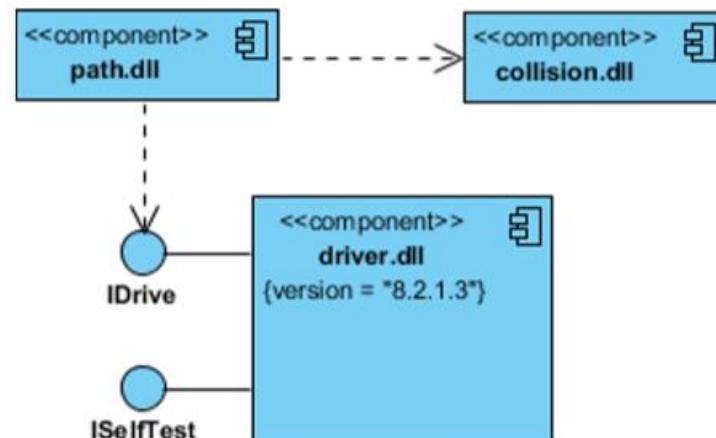
Modelarea codului sursă

- Prin inginerie directă sau inversă, identificați setul de fișiere de cod sursă necesare și modelați-le ca și componente cu sterotipul <<file>>.
- Pentru sisteme mai mari, utilizați pachete pentru a afișa grupuri de fișiere de cod sursă.
- Luați în considerare descrierea unei valori etichetate care să indice informații precum numărul versiunii fișierului cod sursă, autorul acestuia și data ultimei modificări.
- Modelați dependențele de compilare dintre aceste fișiere folosind dependențe.

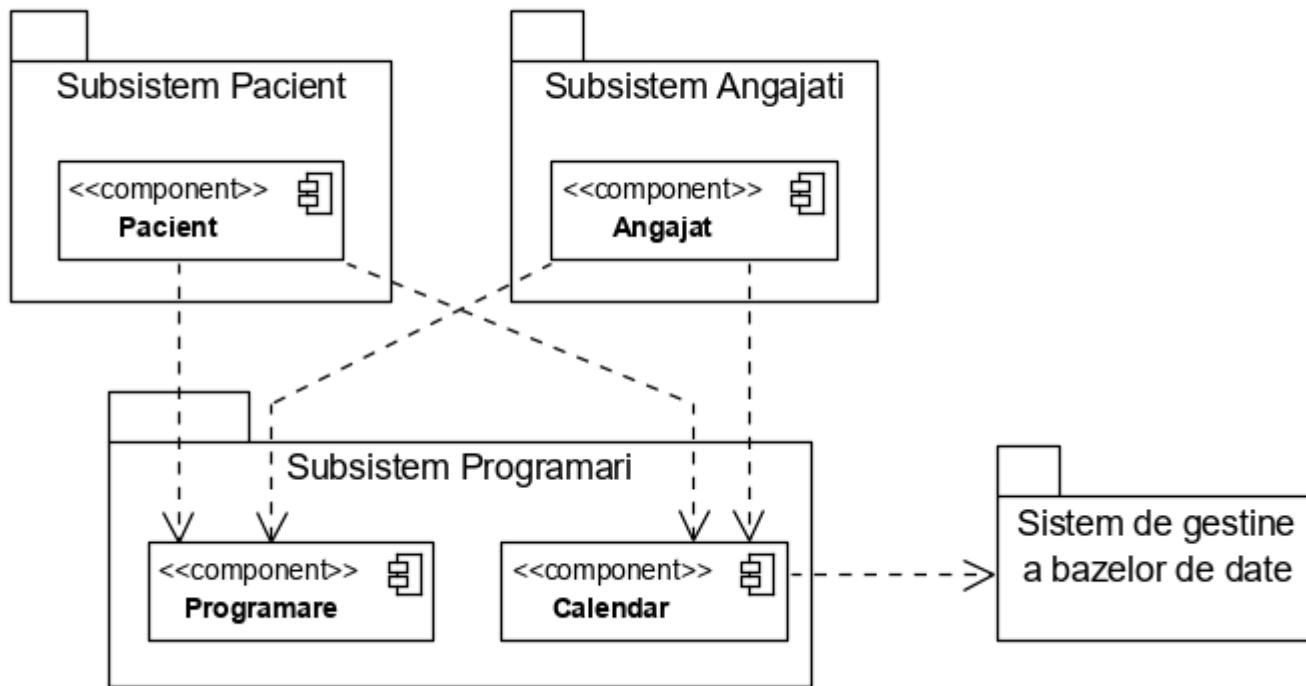


Modelarea unei distribuții executabile

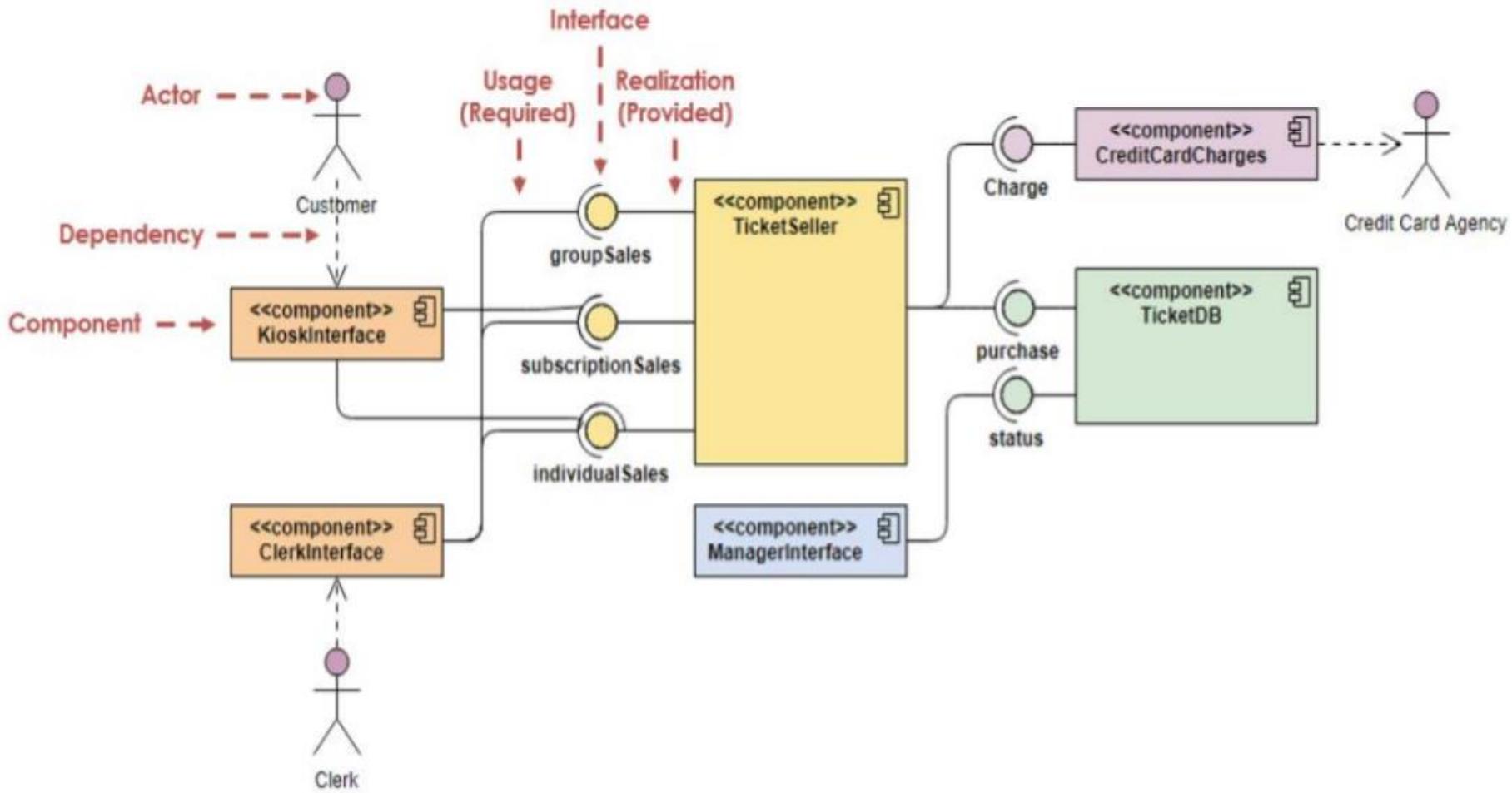
- Identificați setul de componente pe care doriți să le modelați. Se pot identifica toate componentele prezente într-un singur nod sau distribuirea acestor seturi de componente pe toate nodurile din sistem.
- Specificați stereotipul fiecărei componente din acest set. Pentru majoritatea sistemelor, va fi identificat un număr redus de tipuri de componente (cum ar fi executable, biblioteci, tabele, fișiere și documente).
- Pentru fiecare componentă din acest set, luați în considerare relația sa cu vecinii săi. Cel mai adesea, aceasta va implica interfețe care sunt exportate (realizate) de anumite componente și apoi importate (utilizate) de către altele. Dacă se dorește un model cu un nivel mai ridicat de abstractizare, se pot elimina interfețele, arătând numai dependențe dintre componente.



Interdependențe și pachete



Exemplu 1



Exemplu 2

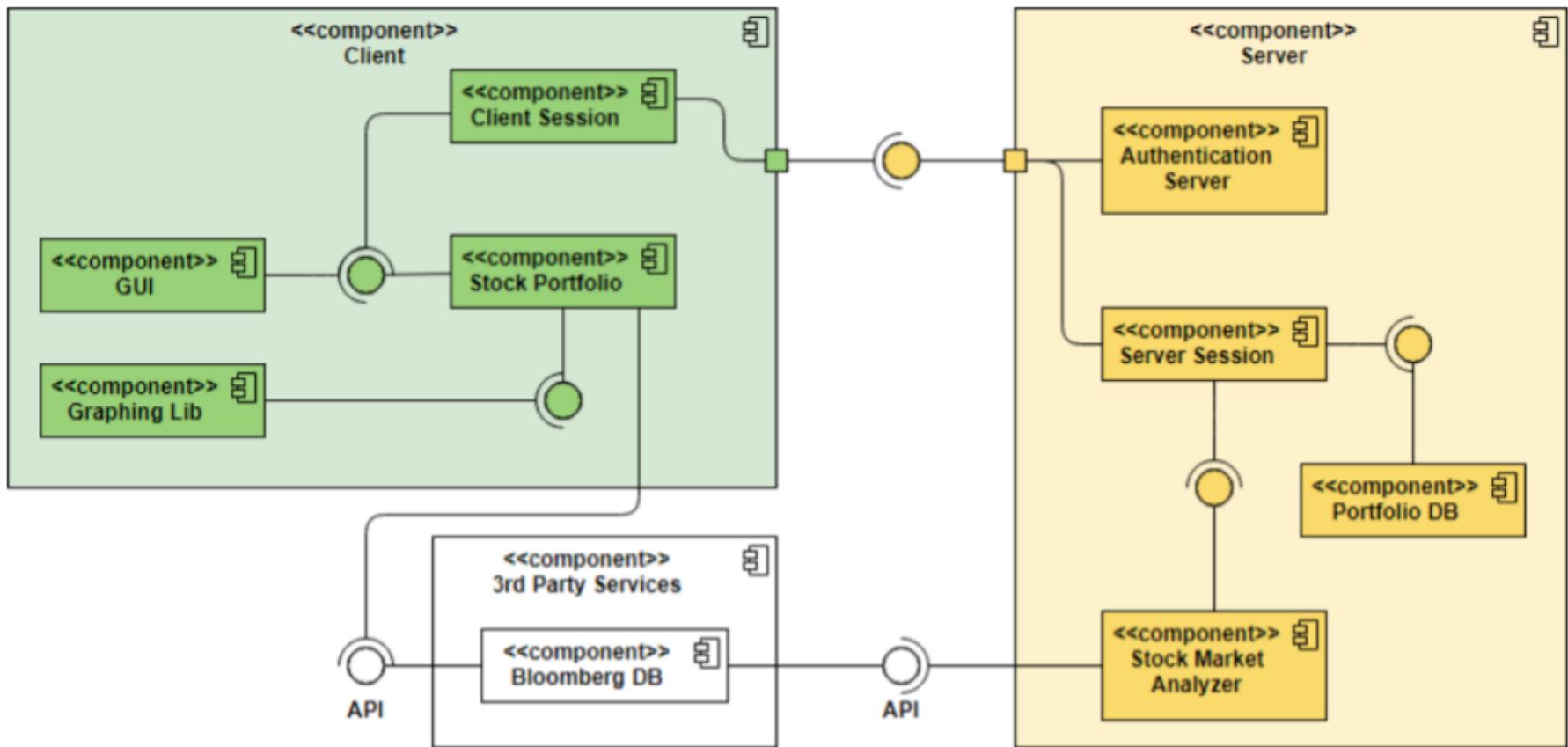


Diagrama de desfășurare

- Diagramele de desfășurare sunt utilizate pentru a reprezenta **relațiile dintre componente hardware** utilizate în infrastructura fizică a unui sistem informatic.
- De exemplu, atunci când este proiectat un sistem informatic distribuit care va utiliza o rețea pe o zonă extinsă, o diagramă de desfășurare poate să fie folosită pentru a arăta **relațiile de comunicare** dintre diferitele noduri din rețea.
- De asemenea, acestea pot fi folosite pentru **a reprezenta componente software** și modul în care acestea sunt alocate peste arhitectură fizică sau infrastructura unui sistem informatic. În acest caz, o diagramă de desfășurare reprezintă mediul necesare pentru execuția componentelor software.



Diagrama de desfășurare

- Elementele de bază ale unei diagrame de desfășurare sunt **nodurile, artefactele și căile de comunicare**.
- **Un nod** reprezintă orice element hardware care trebuie inclus în modelul de proiectare a unei arhitecturi fizică. De exemplu, nodurile pot include computere client, servere, rețele separate sau dispozitive de rețea individuale.
 - În mod obișnuit, un nod este etichetat cu ajutorul lui numele și, eventual, cu un stereotip. Stereotipul este modelat ca element de text înconjurat de simbolurile "<>". Stereotipul reprezintă tipul de nod reprezentat în diagramă.
 - Exemple tipice de dispozitive: dispozitivul mobil, serverul de baze de date, serverul Web și serverul de aplicații.



Diagrama de desfășurare

- **Un artefact** reprezintă o piesă a sistemului informatic care urmează să fie implementată pe arhitectura fizică. În mod obișnuit, un artefact reprezintă o componentă software, un subsistem, o tabelă dintr-o baze de date, o întreagă bază de date sau un nivel al aplicației (gestionarea datelor sau interacțiunea om-calculator). Artefactele pot fi etichetate atât cu un nume, cât și cu un stereotip.
- **O cale de comunicare** reprezintă o legătură între nodurile arhitecturii fizice. Căile de comunicare sunt stereotipizate pe baza tipului de legături pe care le reprezintă (de exemplu, LAN, Internet, serial, paralel sau USB) sau un protocol (de exemplu, TCP / IP).



Diagrama de desfășurare – notații

Element	Reprezentare
Nodul: <ul style="list-style-type: none">▪ Este o resursă de calcul, de exemplu un computer client, un server, o rețea separată sau un dispozitiv de rețea individuală.▪ Este etichetat cu numele său.▪ Poate conține un stereotip pentru a eticheta în mod specific tipul de nod reprezentat, de exemplu, dispozitiv, stație de lucru client, server de aplicații, dispozitiv mobil etc.	
Artefactul: <ul style="list-style-type: none">▪ Este o specificare a unei componente software.▪ Este etichetat cu numele său.▪ Poate conține un stereotip pentru a marca în mod specific tipul de artefact (fișierul sursă, tabelă de baze de date, fișier executabil).	
Calea de comunicare: <ul style="list-style-type: none">▪ Reprezintă o asociere între două noduri.▪ Permite nodurilor să schimbe mesaje.▪ Poate conține un stereotip pentru a eticheta în mod specific tipul de cale de comunicare reprezentat (Internet, serial, paralel) sau poate fi doar denumită sau poate fi calificată (agregare, compunere, dependență, generalizare etc.)	

Diagrama de desfășurare

- Diagramele de desfășurare conțin două tipuri de noduri: **dispozitive și mediile de execuție**.
 - **Dispozitivele (Device)** sunt resurse de calcul cu capacitați de procesare și capacitatea de a executa programe (calculatoare, laptopuri și telefoane mobile).
 - **Mediile de execuție (EEN – Execution Environment Node)** sunt noduri care conțin medii software capabile să execută alte entități software precum sisteme de operare, servere web (Apache sau Microsoft's Internet Information Server (IIS) sau Java Runtime Environment (JRE)).

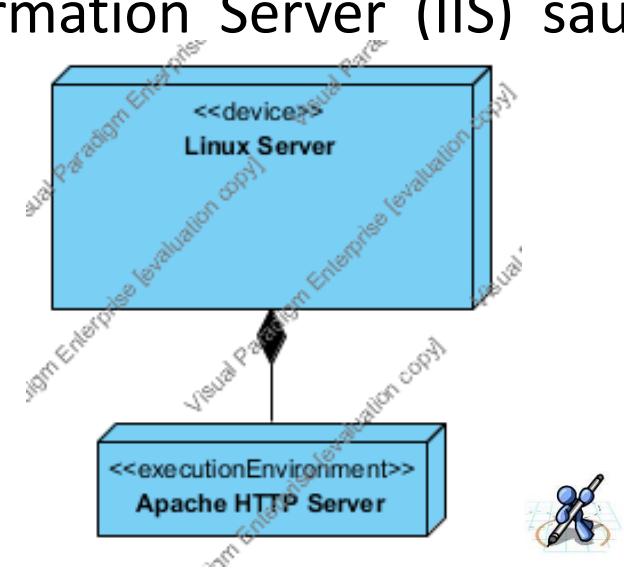
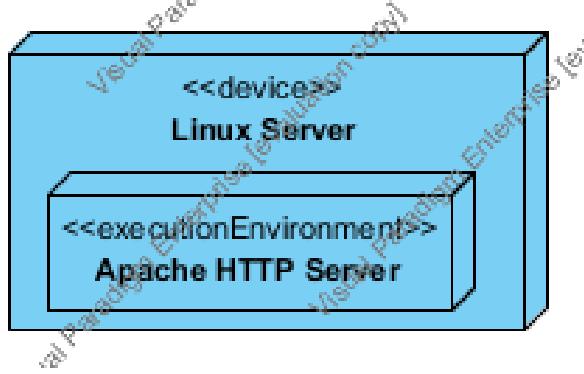


Diagrama de desfășurare

- Diagramele de desfășurare pot fi utilizate pentru reprezentarea componentelor ce pot apartine anumitor noduri prin imbricarea grafică a simbolului componentei în cadrul simbolului ce reprezintă nodul.

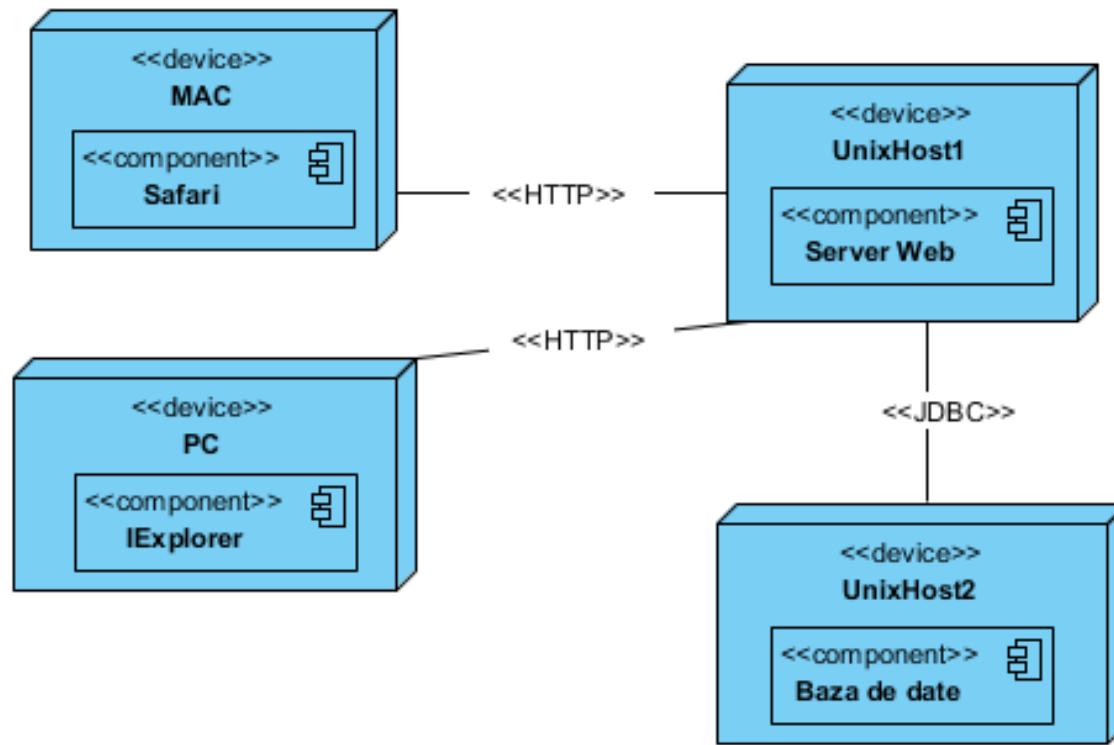
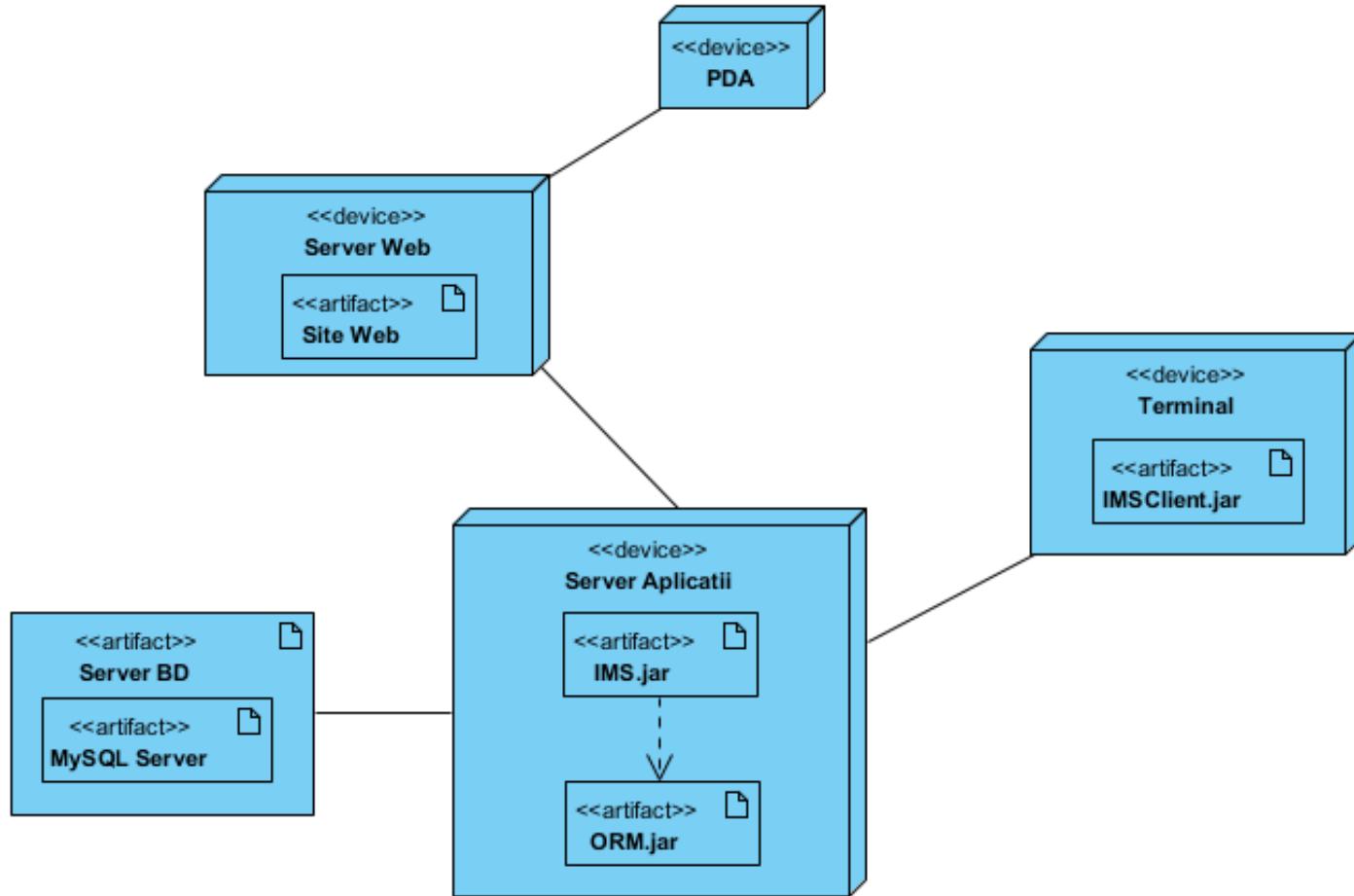


Diagrama de desfășurare - exemplu



ACADEMIA DE STUDII ECONOMICE BUCUREŞTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

PROIECTAREA SISTEMELOR INFORMATICE

- CURS 10 -

BUCUREŞTI
2020-2021

Metodologii de realizare a sistemelor informatice

METODOLOGII DE REALIZARE A SISTEMELOR INFORMATICE

- O metodologie reprezintă o **abordare formalizată** a implementării ciclului de viață. Scopul acesteia este să ofere indicații privind modul de dezvoltare a sistemului, furnizând, o **listă de pași** care trebuie urmați **și de livrabile** corespunzătoare acestora.
- Există o **mare varietate** de metodologii de dezvoltare a sistemelor și fiecare este unică, în funcție de ordonarea etapelor de dezvoltare și de importanța pe care o acordă fiecăreia dintre ele.
- Unele metodologii sunt **standarde** formale utilizate de agențiile guvernamentale, în timp ce altele au fost **dezvoltate de companii** de profil pentru a le oferi clientilor.
- Multe organizații au **metodologii interne** care au fost perfecționate de-a lungul anilor.
- Metodologiile pot fi **clasificate** după diferite criterii.

TIPOLOGIA METODOLOGIILOR DE REALIZARE A SISTEMELOR INFORMATICE

A. Clasificare după gradul de generalitate:

- *Metodologiile generale* au un grad înalt de generalitate, pot fi folosite pentru realizarea sistemelor informatiche din domenii diferite. Exemple: SSADM (Structured System Analysis and Design Methodology), MERISE (Méthode d'Etude et de Realization, Informatique pour les Systèmes d'Entreprise), OMT (Object Modeling Technique), RUP (Rational Unified Process).
- *Metodologii specialize* sunt cele dezvoltate și utilizate pentru implementare a unui singur produs software. Exemplu: ASAP (pentru SAP).

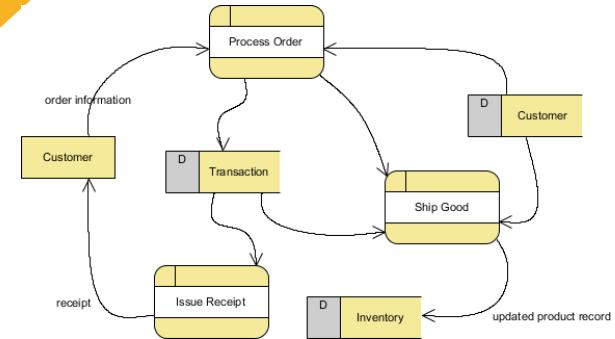
TIPOLOGIA METODOLOGIILOR DE REALIZARE A SISTEMELOR INFORMATICE

B. Clasificare din punct de vedere al modului de abordare al sistemelor:

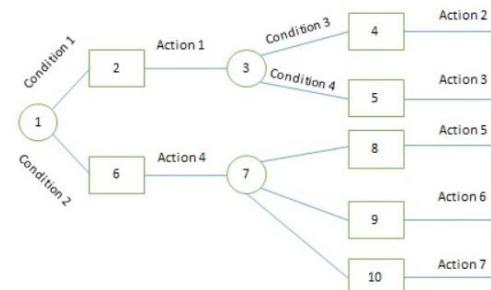
- *Metodologiile cu abordare structurată* au ca principiu de lucru împărțirea sistemului în subsisteme pe baza funcțiilor sistemului (*abordarea funcțională*) sau în funcție de date (*abordarea bazată pe date*). Propun modelarea datelor separat de modelarea procedurilor. Modelarea procedurilor se face plecând de la ideea că funcțiile sunt active, având un comportament, iar datele sunt afectate de aceste funcții.
- *Metodologiile cu abordare orientată obiect* permit construirea sistemelor informaticе folosind *conceptele tehnologiei orientate obiect*. Tehnologia orientată obiect a apărut odată cu apariția limbajelor de programare orientate obiect (SIMULA (1960), SMALLTALK (1970), EIFFEL, C++, Object Pascal (1980)).

METODOLOGII CU ABORDARE STRUCTURATĂ

- Utilizează reprezentării grafice, la îndemâna atât a analiștilor, cât și a beneficiarilor.
- Permite lanificarea eficace a proiectului prin divizarea în subsisteme.
- Oferă posibilitatea reducerii timpului și a costului de dezvoltare a sistemului prin luarea în considerare de la început a detaliilor, prin interacțiunea continuă cu beneficiarul.
- Folosesc diferite tehnici pentru reprezentarea sistemului:
 - Diagrame de flux de date
 - Diagrame entitate-relație
 - Diagrame de tranziție a stărilor
 - Dicționar de date
 - Arbori decizionali
 - Tabele de decizie
 - Engleza structurată
 - Pseudocod



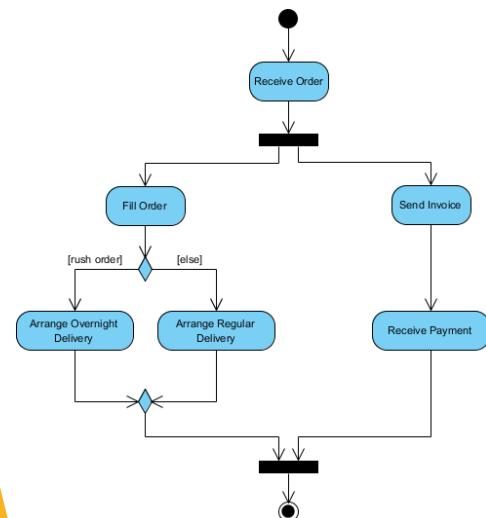
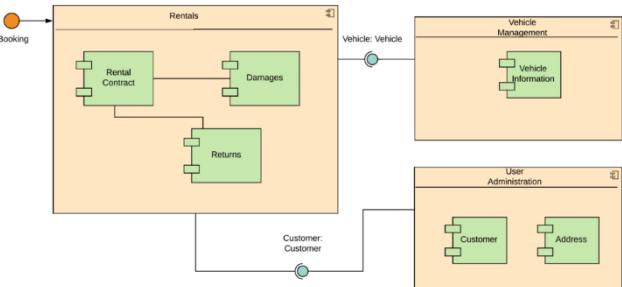
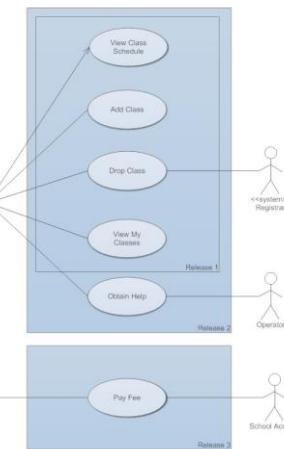
Sr.No.	Data Name	Description	No. of Characters
1	ISBN	ISBN Number	10
2	TITLE	title	60
3	SUB	Book Subjects	80
4	ANAME	Author Name	15



METODOLOGII CU ABORDARE ORIENTATĂ OBIECT

- Datele și prelucrările nu sunt reprezentate distinct, ca în cazul abordării structurate, ci încapsulat în clase de obiecte.
- Modelul realizat pentru un sistem poate fi modificat în scurt timp pentru a fi utilizat pentru modelarea sistemelor din aceeași sferă de activitate.
- Oferă consistență crescută între activitățile de analiză, proiectare și programare.
- Sunt orientate pe structura și comportamentul obiectelor care compun sistemul.
- Procesul de dezvoltare este iterativ și incremental, permitând rafinarea și extinderea modelelor.
- Permit modelarea arhitecturii logice și fizice a sistemului.
- Folosesc o multitudine de diagrame pentru reprezentarea grafică a aspectelor statice, dinamice și funcționale ale sistemului (spre exemplu, limbajul UML oferă 14 tipuri de diagrame).

Use Case Diagram: Class Registration



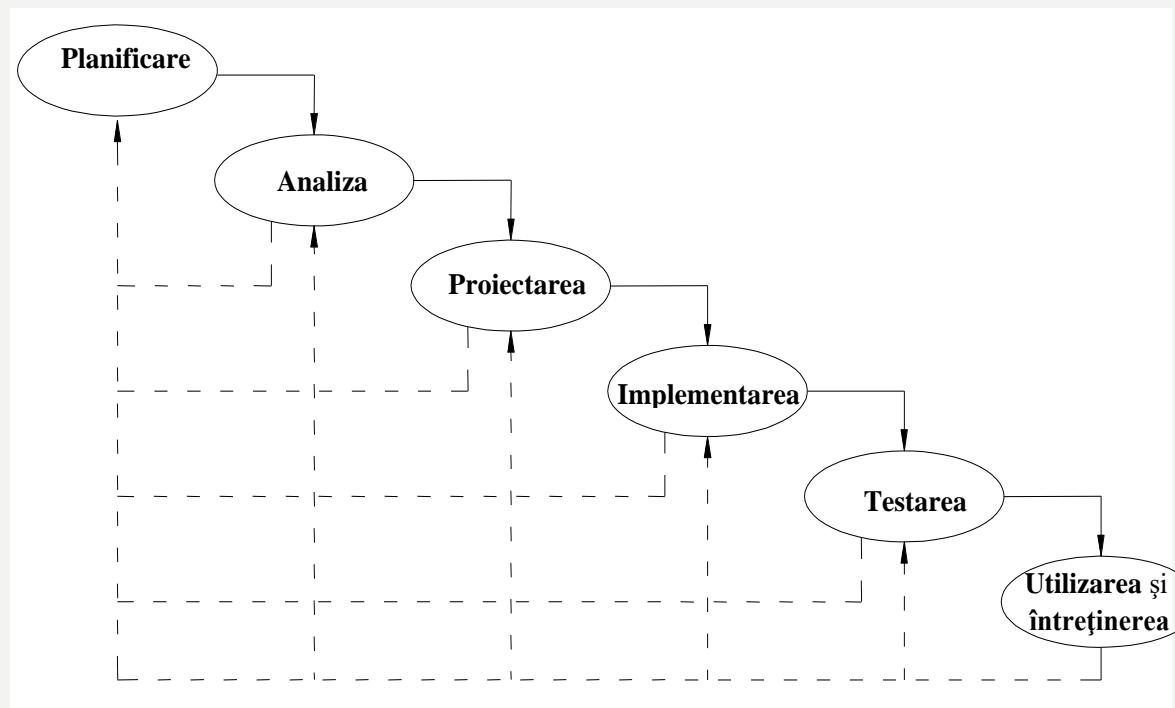
TIPOLOGIA METODOLOGIILOR DE REALIZARE A SISTEMELOR INFORMATICE

C. Clasificare după modelul de parcurgere a etapelor ciclului de viață:

- *Modelul de parcurgere în cascadă*
- *Modelul de parcurgere în spirală*
- *Modelul de parcurgere cu extensii*
- *Modelul de parcurgere evolutiv*
- *Bazate de dezvoltarea rapidă (RAD)*
- *Bazate pe dezvoltarea agilă*

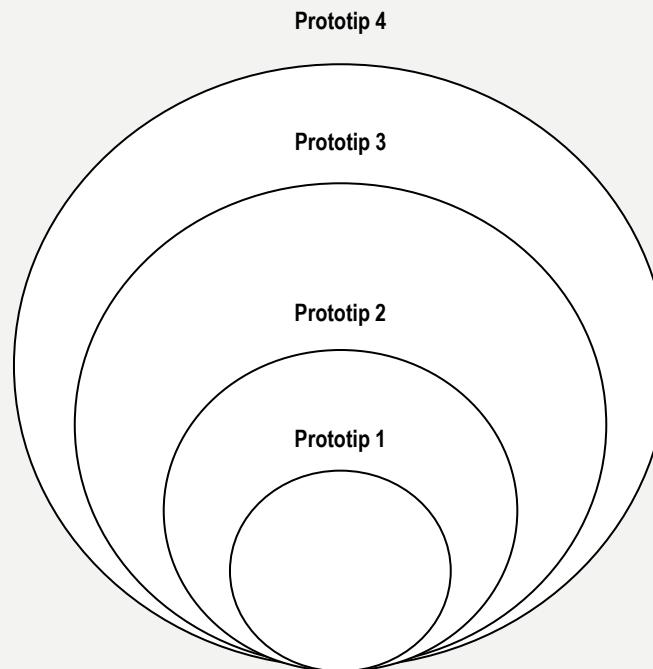
MODELUL DE PARCURGERE ÎN CASCADĂ

- ✓ Parcurgerea secvențială a etapelor, cu eventuale reveniri la etapa precedentă.
- ✓ Utilizat pentru sisteme informaticice de mică complexitate.
- ✓ Modelul în cascadă sau liniar este teoretic, deoarece în realitate, parcurgerea etapelor este un proces iterativ, desfășurându-se adesea în paralel mai multe activități.



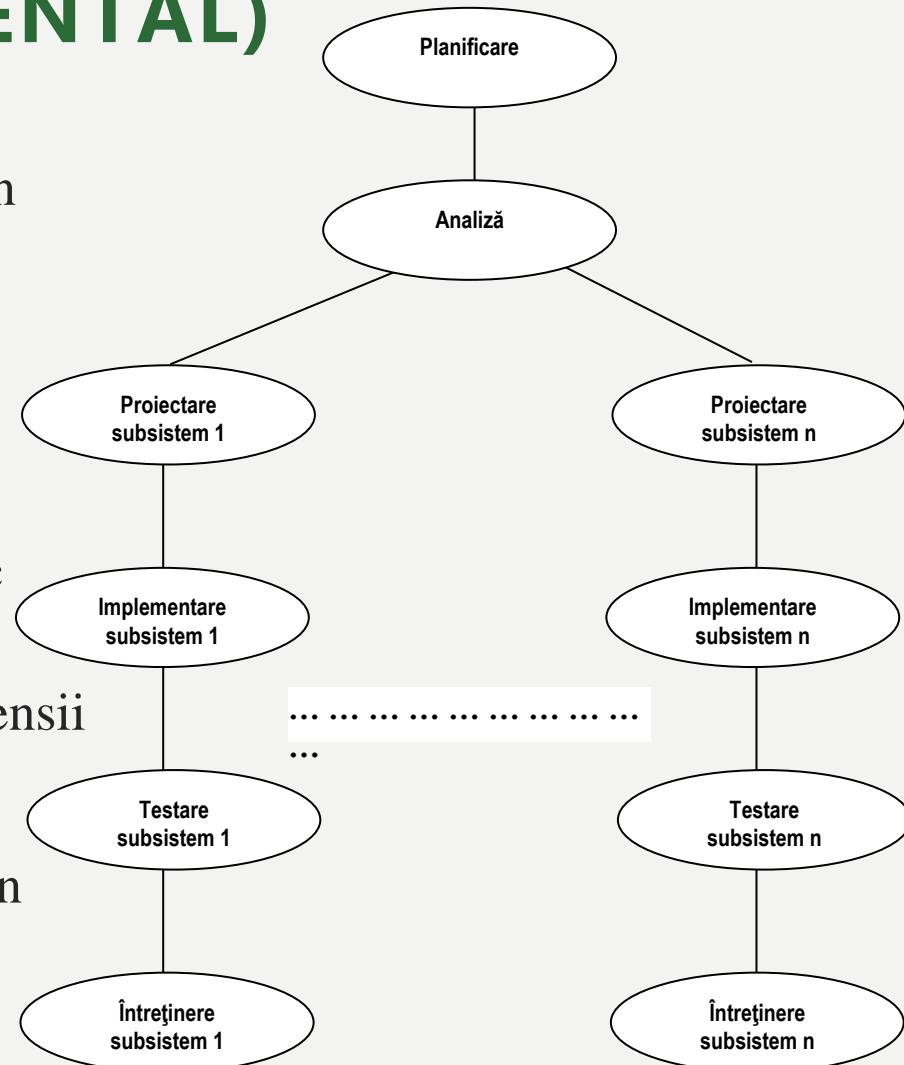
MODELUL DE PARCURGERE ÎN SPIRALĂ (MODELUL CU PROTOTIP)

- ✓ Presupune elaborarea completă, rapidă și la costuri scăzute a unei versiuni inițiale, simplificate, cu caracter de prototip, pe baza căreia se stabilesc noi specificații de definire a sistemului informatic și se desfășoară activitatea de realizare a unei noi versiuni de sistem informatic.
- ✓ Elaborarea noii versiuni presupune parcurgerea integrală sau parțială a etapelor, modificându-se numai anumite părți din prototip.



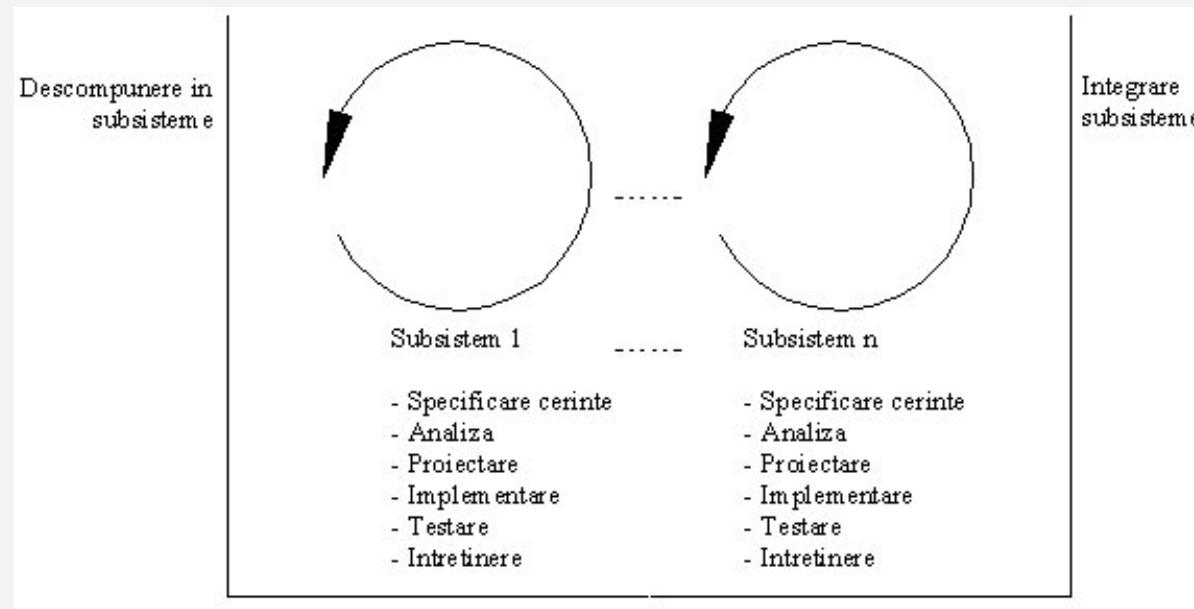
MODELUL DE PARCURGERE CU EXTENSII (INCREMENTAL)

- ✓ Se utilizează atunci când sistemele informatic se pot realiza și pune în funcțiune parțial pe subsisteme, aplicații, module.
- ✓ Realizarea lor se poate face deci în maniera extensibilă, astfel încât la început se analizează și se definesc cerințele, iar apoi subsistemele se realizează și se integrează prin extensiile succesive sau simultane.
- ✓ De obicei, extensiile se ramifică din etapa de proiectare a sistemului informatic.



MODELUL DE PARCURGERE EVOLUTIV

- ✓ Se utilizează în cazul sistemelor complexe, care se descompun în subsisteme. Ele sunt realizate și livrate în mod iterativ și contribuie la sporirea treptată a performanțelor sistemului.
- ✓ Pornește de la un studiu inițial privind obiectivele sistemului informatic.
- ✓ Ulterior, oricare dintre subsisteme trece prin toate fazele de dezvoltare a sistemelor: definirea cerințelor, analiză, proiectare, implementare, testare, întreținere, pentru ca în final acestea să fie integrate.
- ✓ Este rezultatul unei concepții bazate pe arhitecturi deschise și flexibile.



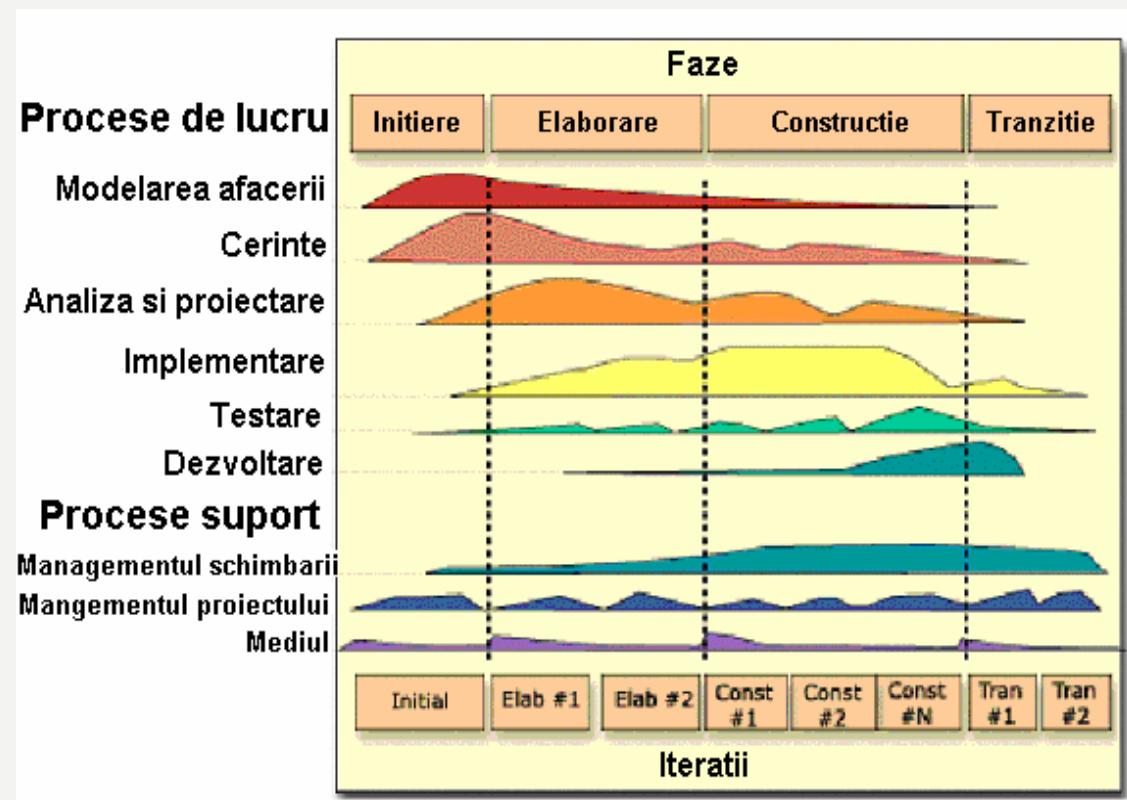
METODOLOGIA UNIFICATA DE REALIZARE A SISTEMELOR INFORMATICE (RUP)

- Rational Unified Process (RUP) este un **proces general** pentru dezvoltarea orientată obiect de produse informaticе.
- Este un ghid care arată **cum se poate utiliza practic UML** (Unified Modeling Language) pentru a dezvolta un sistem informatic.
- RUP a fost realizat de compania Rational și dezvoltat acum de către IMB.
- Nucleul îl reprezintă metodologia propusă de Jacobson, Booch și Rumbaugh (Unified Process) care este mai mult decât un simplu proces, este un cadru general care a permis dezvoltarea de metodologii specializate (pe diverse tipuri de sisteme informaticе în funcție de aria de aplicare, diverse tipuri de organizații, niveluri de competență sau dimensiuni diferite ale proiectelor)

FAZELE RUP

Axa orizontală: reprezintă **timpul** și evidențiază aspectele dinamice ale procesului. Pe axa orizontală procesul se exprimă în termeni de: cicluri, faze, iterații și jaloane.

Axa verticală: reprezintă **aspectele statice** ale procesului și se exprimă în termeni de: activități, produse, execuționi și fluxuri.



METODOLOGII BAZATE PE DEZVOLTAREA RAPIDĂ RAD (RAPID APPLICATION DEVELOPMENT)

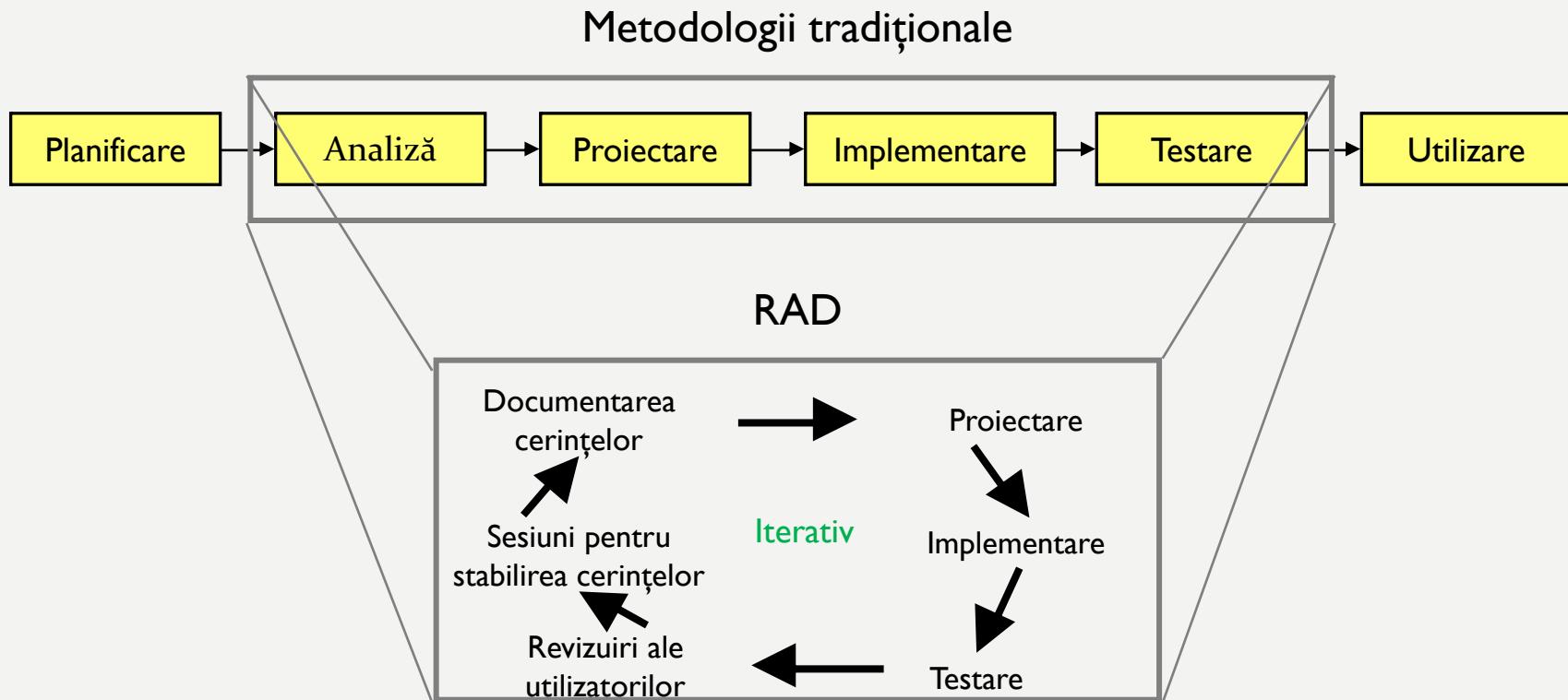


- Se referă la o serie de principii care urmăresc ajustarea etapelor de realizare a sistemelor informaticice, astfel încât o parte a sistemului să fie dezvoltată și să ajungă la utilizatori **rapid**.
- Astfel, utilizatorii pot **înțelege mai bine** sistemul și pot sugera **revizuirile** care aduc sistemul mai aproape de cerințele acestuia.
- Se recomandă ca analiștii să folosească tehnici speciale și instrumente informaticice pentru **a accelera** etapele de analiză, proiectare și implementare, cum ar fi:
 - instrumente CASE;
 - sesiuni comune pentru stabilirea cerințelor Join Requirement Planning (JRP);
 - limbaje de programare de generația a patra;
 - limbaje de programare vizuale ;
 - generatoare de cod;
 - reutilizarea componentelor software;
- Astăzi, aproape toate mediile de dezvoltare integrate au facilități specifice RAD.

METODOLOGII BAZATE PE DEZVOLTAREA RAPIDĂ RAD (RAPID APPLICATION DEVELOPMENT)



- RAD comprimă pașii metodologiilor tradiționale într-un proces iterativ.
 - Se bazează pe prototipizare și pe revizuiri ale utilizatorilor înainte de a trece la parcurgerea unei noi iterări.



METODOLOGII BAZATE PE DEZVOLTAREA AGILĂ



- Aceste metodologii sunt orientate pe programare și au puține reguli și practici.
- Toate metodologiile de dezvoltare agile sunt bazate pe **manifestul agil** și pe un set de **douăsprezece principii**:
 - Este prioritară **satisfacția clientului** prin livrarea rapidă și continuă de software calitativ.
 - Schimbarea cerințelor este binevenită chiar și într-o fază avansată a dezvoltării. Procesele agile **valorifică schimbarea în avantajul competitiv al clientului**.
 - **Livrarea de software funcțional se face frecvent**, de preferință la intervale de timp cât mai mici, de la câteva săptămâni la câteva luni.
 - Oamenii de afaceri și dezvoltatorii trebuie să **colaboreze zilnic** pe parcursul proiectului.
 - Proiecte se construiesc în jurul **oamenilor motivați**. Oferindu-le mediul propice și suportul necesar este foarte probabil că obiectivele vor fi atinse.

METODOLOGII BAZATE PE DEZVOLTAREA AGILĂ

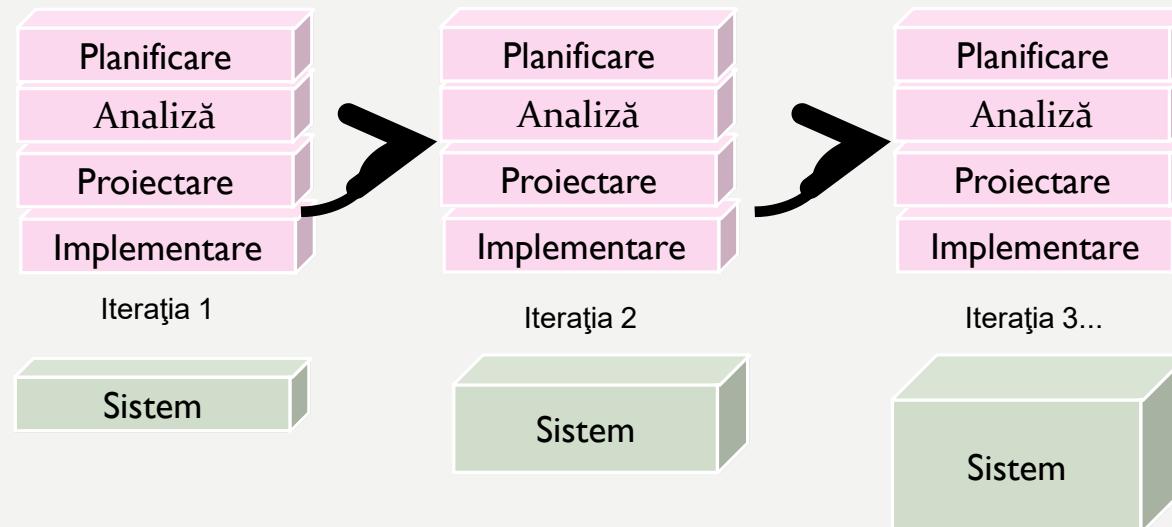


- Principii ale manifestului agil- continuare:
 - Cea mai eficientă metodă de a transmite informații înspre și în interiorul echipei de dezvoltare este **comunicarea față în față**.
 - Software-ul funcțional este principala **măsură a progresului**.
 - Procesele agile promovează dezvoltarea durabilă. Beneficiarii, dezvoltatorii și utilizatorii trebuie să poată menține un **ritm de lucru constant** pe termen lung.
 - Atenția continuă pentru **excellență tehnică și design bun** îmbunătățește agilitatea.
 - **Simplitatea**-arta de a maximiza cantitatea de muncă nerealizată- este esențială.
 - Cele mai bune arhitecturi, cerințe și design sunt create de **echipe care se auto-organizează**.
 - La intervale regulate, echipa **reflectă la cum să devină mai eficientă**, apoi își adaptează și ajustează comportamentul în consecință.

METODOLOGII BAZATE PE DEZVOLTAREA AGILĂ



- Pe baza acestor principii, metodologiile agile se concentrează pe optimizarea procesului de dezvoltare a sistemelor prin **eliminarea unei părți semnificative din modelare și documentare**.
- Este susținută realizarea simplă, **iterativă** a sistemelor. Practic toate metodologiile agile sunt folosite **în combinație cu tehnologiile orientate-obiect**.
- Toate metodologiile bazate pe dezvoltarea agilă urmează un ciclu de dezvoltare simplu prin **parcursarea etapelor tradiționale** ale procesului de dezvoltare a sistemelor.
- Două dintre cele mai populare exemple de metodologii de dezvoltare agile sunt **Extreme Programming (XP)** și **SCRUM**.



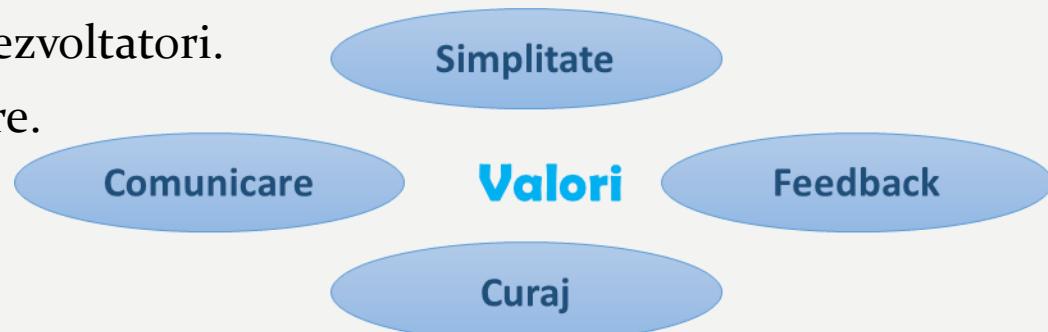
METODOLOGII BAZATE PE DEZVOLTAREA AGILĂ



Avantaje	Dezavantaje
Abordare realistă în realizarea sistemelor informaticice.	Nu sunt potrivite pentru a gestiona dependențe complexe.
Promovează lucrul în echipă și învățarea.	Risc crescut de sustenabilitate, mentenabilitate și extensibilitate.
Funcționalitățile pot fi implementate rapid și verificate.	Fără o documentație suficientă, nici sistemul și nici procesul de dezvoltare al sistemelor nu pot fi auditate.
Model potrivit pentru mediile care se schimbă în mod continuu.	Depinde foarte mult de interacțiunea cu beneficiarul.
Reguli minime, documentație ușor de realizat.	Transferul de informații către membrii noi ai echipei este îngreunat de lipsa documentației.
Ușor de gestionat.	Lipsa regulilor poate duce la apariția unui mediu de lucru haotic.
Oferă flexibilitate.	Dependența de membrii echipei care cunosc cerințele sistemului.

EXTREME PROGRAMMING - XP

- Pune accentul pe **codificare (standarde, principii)** - utilizează un set comun de nume, descrieri și practici de codificare.
- Susține ca programatorii să **lucreze câte doi** ("pair programming"), iar programatorii au o responsabilitate comună pentru fiecare componentă software elaborată.
- Numeroase **sesiuni de discuții** pe parcursul dezvoltării.
- **Feedback rapid** al utilizatorilor finali în mod continuu.
- Dezvoltatorii trebuie să aibă o mentalitate orientată către **calitate**.
- Se bazează foarte mult pe **refactoring**, care este un mod disciplinat de restructurare a codului pentru a-l păstra simplu.
- Sistemul este dezvoltat într-un mod **evolutiv și incremental**.
- Fiecare iteratie (**1-4 săptămâni**) are un rezultat funcțional.
- **Suport redus** pentru modelare.
- **Relație strânsă** între clienți și dezvoltatori.
- **Lipsa documentației** de realizare.



EXTREME PROGRAMMING - XP



Când se recomandă:

- Pentru proiectele mici cu echipe extrem de motivate, unite, stabile, și cu experiență, XP ar trebui să funcționeze foarte bine.
- XP este recomandat numai pentru grupuri mici de dezvoltatori, nu mai mult de zece persoane.
- Pentru cicluri scurte de dezvoltare și atunci când sunt facilitate dicuțiile frecvente cu utilizatorii finali.

Când NU se recomandă:

- În cazul în care proiectul nu este mic sau echipele nu sunt unite, succesul unui efort de dezvoltare XP este îndoiosnic.
- Există dubii asupra beneficiilor introducerii unor contractori externi în cadrul unei echipe existente, când se lucrează conform XP.
- XP necesită un grad ridicat de disciplină; în caz contrar proiectele vor deveni nefocalizate și haotice.
- Nu se recomandă pentru aplicații mari. Din cauza lipsei de analiză și documentației de proiectare, există doar documentație cod asociat cu XP, deci menținerea sistemelor de mari dimensiuni construite cu XP poate fi imposibilă.

SCRUM



- Numele metodologiei este preluat din jocul de rugby, și desemnează o **grămadă ordonată** folosită pentru a reporni un joc
- Creatorii metodei Scrum cred că indiferent cât bine este realizată planificarea dezvoltării unui sistem, de îndată ce software-ul începe să fie dezvoltat va izbucni haosul și **planurile nu vor mai avea utilitate.**

Principii de organizare

- Echipele sunt **auto-organizate** și **auto-dirijate**.
- Spre deosebire de alte abordări, echipele Scrum **nu au un lider de echipă** desemnat.
- Echipele se organizează într-o **manieră simbiotică** și își stabilesc propriile obiective pentru fiecare sprint (iterație).



Principii de funcționare

- Odată ce un sprint a început, echipele Scrum **nu mai iau în considerare** nici o cerință suplimentară.
- Orice cerințe noi care sunt descoperite sunt plasate într-o **listă de cerințe care urmează să fie abordate**.
- La începutul fiecărui zile de lucru, are loc o reuniune unde **toți membrii echipei stau într-un cerc** și raportează realizările zilei precedente, stabilesc ce au de gând să facă astăzi și descriu tot ceea ce a blocat progresul în ziua precedentă.
- Pentru a asigura un progres continuu, orice blocaj identificat **este abordat** în următoarea oră.
- La sfârșitul fiecărui sprint, **echipa prezintă software-ul** clientului.
- Pe baza rezultatelor iterătiei încheiate, este început **un nou plan** pentru următoarea iterătie.