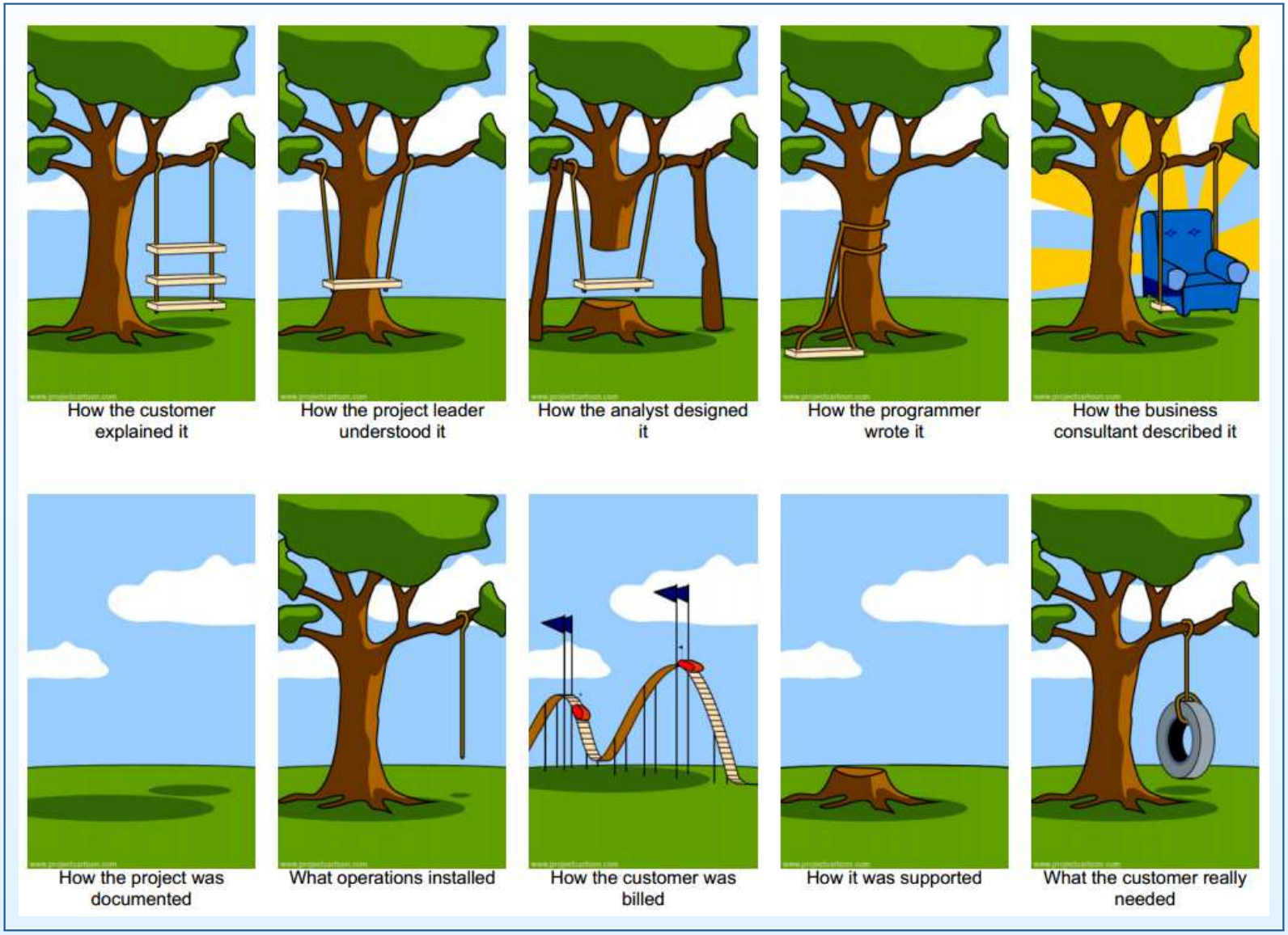
# *Curs 1*

## *Introducere în Ingineria Sistemelor Soft*

*Suport de curs bazat pe* ***B. Bruegge and A.H. Dutoit****,*
***"Object-Oriented Software Engineering using UML, Patterns, and Java"***

# **Communication** in SE - a Tree Swing story!



# Obiective curs

- Înţelegerea conceptelor legate de modelarea softului
- Cunoaşterea şi aplicarea tehnicilor de dezvoltare a softului pe baza modelelor
- Familiarizarea cu limbajul UML
- Abilitatea de a utiliza instrumente CASE
- Cunoaşterea etapelor ciclului de viaţă al softului şi a modelelor de procese soft
- Familiarizarea cu unele dintre metodologiile de dezvoltare, tradiţionale sau agile
- Însuşirea unor aspecte de bază legate de gestiunea softului

# Utile

- Evaluare - colocviu

| Modalitate de evaluare | Procent din nota finală |
|---|---|
| Proiect de laborator | 40% |
| Examen scris | 60% |
| Activitate seminar | bonus (în limita unui punct) |

- Resurse (notiţe curs, materiale seminar, cerinţe laborator)

Cod team curs: h0kwscn

Cod team seminar: m4kvdhx

# Bibliografie


- **[1] Bruegge, B., Dutoit, A.H., *Object-Oriented Software Engineering using UML, Patterns, and Java* (3rd edition), Prentice Hall, 2010.**
- [2] Larman, C., *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd edition), Addison-Wesley, 2004.
- [3] Sommerville, I., *Software Engineering* (10th edition), Pearson, 2015.
- [4] Pressman, R.S., *Software Engineering - A Practitioners Approach* (8th edition), McGraw-Hill, 2014.
- [5] Schach, S.R., *Object-Oriented and Classical Software Engineering*, (8th edition), McGraw-Hill, 2010.
- [6] IEEE Computer Society, *SWEBOK v3.0: Guide to the Software Engineering Body of Knowledge*, 2014.
- [7] Pârv, B., *Analiza si proiectarea sistemelor*, Univ. Babeş-Bolyai, CFCID, Facultatea de Matematică şi Informatică, Cluj-Napoca, 2004.
- [8] Seidl, M., Scholz, M., Huemer, C., Kappel, G., *UML@Classroom - An Introduction to Object-Oriented Modeling*, Springer, 2015.

# Bibliografie (cont.)


- [9] Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3nd edition), Addison Wesley, 2003.
- [10] Rumbaugh, J., Jacobson, I., Booch, G., *The Unified Modeling Language Reference Manual* (2nd edition), Addison Wesley, 2010.
- [11] Booch, G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide, V.2.0*, Addison Wesley, 2005.
- [12] Object Management Group, *UML 2.4.1 Infrastructure & Superstructure Specification - 2012 ISO standard*, `https://www.omg.org/spec/UML/ISO/19505-1/PDF`. `https://www.omg.org/spec/UML/ISO/19505-2/PDF`.
- [13] Object Management Group, *OCL 2.3.1 Specification - 2012 ISO standard* , `https://www.omg.org/spec/OCL/ISO/19507/PDF`.
- [14] Object Management Group, *UML 2.5.1 Specification*, 2017, `https://www.omg.org/spec/UML/2.5.1/PDF`.
- [15] Object Management Group, *OCL 2.4 Specification*, 2014, `https://www.omg.org/spec/OCL/2.4/PDF`.

# Bibliografie (cont.)


- [16] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1996.
- [17] Fowler, M. et al., *Refactoring - Improving the Design of Existing Code*, Addison-Wesley, 1999.
- [18] Eckel, B., *Thinking in Java* (4th edition), Prentice Hall, 2006.
- [19] Beck, K., *Test Driven Development: By Example*, Addison-Wesley, 2002.
- [20] Rubin, K.S., *Essential Scrum - A Practical Guide to the Most Popular Agile Process*, Addison-Wesley, 2012.
- [21] Brambilla, M., Cabot, J., Wimmer, M., *Model-Driven Software Engineering in Practice (2nd edition)*, Morgan and Claypool Publishers, 2017.

# Structura curs

- **Curs 1: *Introducere în ingineria sistemelor soft***
- Curs 2: *Specificarea modelelor folosind UML*
- Curs 3: *Colectarea cerinţelor*
- Curs 4: *Analiza cerinţelor*
- Curs 5: *Proiectarea de sistem*
- Curs 6: *Proiectarea obiectuală - Şabloane de proiectare*
- Curs 7: *Proiectarea obiectuală - Specificarea interfeţelor folosind OCL*
- Curs 8: *Implementarea sistemului. Transformarea modelelor în cod*
- Curs 9: *Testarea sistemului*
- Curs 10: *Ciclul de viaţă al sistemelor soft*
- Curs 11: *Gestiunea proiectelor soft*
- Curs 12: *Metodologii de dezvoltare a sistemelor soft*
- Curs 13: *Model-Driven Software Engineering*
- Curs 14: *Colocviu*

# Sumar Curs 1

- Ingineria sistemelor soft:
  - Motivaţie
  - Definiţii
  - Evoluţie
  - Eşecuri celebre
  - Provocări
- Ce presupune ingineria sistemelor soft?
  - Modelare
  - Rezolvare de probleme (eng. *problem solving*)
  - Acumulare de cunoştinţe (eng. *knowledge aquisition*)
  - Argumentare (eng. *rationale*)
- Concepte ale ingineriei sistemelor soft
  - Participanţi şi roluri
  - Sisteme şi modele
  - Produse (eng. *work products*)
  - Activităţi, sarcini şi resurse
  - Cerinţe funcţionale şi nefuncţionale
  - Notaţii, metode şi metodologii

# Sumar Curs 1 (cont.)

- Activităţi ale procesului de dezvoltare a softului
  - Colectarea cerinţelor
  - Analiza cerinţelor
  - Proiectarea de sistem
  - Proiectarea obiectuală
  - Implementarea
  - Testarea

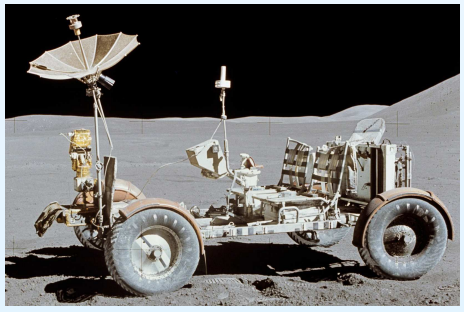
# **Ingineria sistemelor soft**: Motivaţie

- *Criza software* de la sfârşitul anilor ’60, generată de *dezvoltarea artizanală* a programelor
  - calitate slabă a softului - rezultată din cerinţe nerespectate, lipsă de fiabilitate, întreţinere dificilă
  - nerespectarea termenelor de livrare
  - depăşirea bugetului alocat
- Dezvoltatorii de soft erau incapabili să stabilească obiective concrete, să anticipeze resursele necesare şi să gestioneze aşteptările clienţilor

Promisă:

*Luna*

Dezvoltat:



*Vehicul lunar*

Livrat:

*O pereche de roţi pătrate*

## **Ingineria sistemelor soft**: Definiţii

- 1968: prima conferinţă NATO de Software Engineering - Bavaria
  - nevoia unei *abordări sistematice şi disciplinate* a dezvoltării softului
- **Definiţia 1.1 [Ingineria Programării, eng. *Software Engineering*]**: *Stabilirea şi punerea în practică a unor principii inginereşti solide, care să producă aplicaţii soft fiabile şi care să funcţioneze eficient pe maşini reale.* (Fritz Bauer, 1968)
- **Definiţia 1.2 [Ingineria Programării]**: *Aplicarea unei abordări sistematice, disciplinate şi cuantificabile la dezvoltarea, operarea şi întreţinerea softului, mai exact aplicarea ingineriei la soft*. [6]

## **Ingineria sistemelor soft**: Evoluţie

- "Software and cathedrals are very much the same. First we build them, then we pray." (Sam Redwine, 1988)
- "Despite 50 years of progress, the software industry remains years, perhaps decades, short of the mature engineering discipline needed to meet the needs of an information-age society." (Scientific American, 1994)
- În 2002, Institutul Nord American pentru Standarde şi Tehnologie a estimat costul total al erorilor soft în economia americană la 59 miliarde USD
- Studiile actuale indică faptul că majoritatea sistemelor soft dezvoltate au un număr relativ mare de erori, conducând la depăşirea termenelor de predare, a bugetului alocat, precum şi la probleme de utilizabilitate
- Costurile de întreţinere a softului reprezintă 2/3 din costurile totale de dezvoltare; o eroare de specificare este de 20 de ori mai costisitor de reparat în faza de testare, faţă de descoperirea ei la începutul dezvoltării sistemului

# **Ingineria sistemelor soft**: Eşecuri celebre

- *Therac-25* (1985-1987)
  - Între 1985 şi 1987, sistemul medical Therac-25, utilizat în terapia pacienţilor cu cancer, a fost implicat în 6 accidente (cu decese şi răniri grave), aplicând pacienţilor supradoze mari de radiaţii (de peste 100 de ori mai mari decât limita admisă).
- *Ariane 5* (1996)
  - Pe 4 iunie 1996, racheta Ariane 5 a explodat la nici 40 de secunde de la lansare. Pierderile înregistrate au fost de 500 milioane USD (racheta + cargo), plus un deceniu de dezvoltare estimat la aproximativ 7 miliarde USD.
- *Eroarea procesoarelor Pentium* (1994)
  - În 1994 s-a descoperit o eroare în unitatea de împărţire cu virgulă mobilă a procesoarelor Pentium. Pierderile înregistrate au fost de circa 400 milioane USD, plus prejudiciile de imagine.
- *Eroarea anului 1900* (1992)
  - În anul 1992, Mary din Winona, Minnesota a primit invitaţia de a se înscrie la o grădiniţă. Mary avea 104 ani atunci.

# **Ingineria sistemelor soft**: Eşecuri celebre (cont.)

- *Eroarea anului bisect* (1988)
  - Un supermarket a fost amendat cu 1000 USD pentru că a ţinut produse pe raft o zi în plus în februarie 1988. Programul care a imprimat data expirării pe etichete nu a ţinut cont de fapul că anul era bisect.
- *Eroarea de utilizare a interfeţei* (1990)
  - Pe 10 aprilie 1990, în Londra, un metrou a pornit din staţie fără conductor. Acesta a apăsat butonul de pornire, ştiind că sistemul nu permitea plecarea trenului cu uşile deschise. Ulterior, a părăsit trenul pentru a închide o uşă care se blocase. Când aceasta s-a închis, trenul a plecat pur şi simplu.
- *Neîncadrare în timp şi buget* (1995)
  - În 1995, defecţiuni în sistemul aeroportului internaţional din Denver au determinat distrugerea bagajelor clienţilor. Aeroportul s-a deschis 16 luni mai târziu, cu 3.2 miliarde USD peste buget şi un sistem de gestiune a bagajelor preponderent manual.

# **Ingineria sistemelor soft**: Provocări

- **Complexitate**
  - Sistemele soft au multiple funcţionalităţi
  - Sunt construite să îndeplinească un număr mare de obiective
  - Constau dintr-un număr mare de componente
  - Mulţi participanţi, cu pregătiri diferite, iau parte la dezvoltarea lor
  - Procesul de dezvoltare durează mulţi ani
- **Schimbare**
  - Cerinţele se modifică, la iniţiativa clientului sau pe măsură ce dezvoltatorii înţeleg domeniul problemei
  - În cazul în care proiectul durează mulţi ani, dezvoltatorii se pot schimba
  - Sistemul se schimbă, ca urmare a greşelilor descoperite la testare
  - Tehnologia se schimbă, de multe ori înainte de finalizarea proiectului

- **Definiţia 1.3 [Ingineria Programării]**: *O colecţie de tehnici, metodologii şi instrumente care ajută la producerea unor sisteme soft de calitate, dezvoltate cu un buget dat şi cu încadrare în termene, în contextul unor permanente schimbări*. [1]

# Ce presupune ingineria sistemelor soft?

- Modelare (eng. *Modeling*)
  - Permite gestionarea *complexităţii*, prin focusarea pe aspectele relevante şi ignorarea detaliilor
  - Pe parcursul procesului de dezvoltare, se construiesc diverse modele ale domeniului problemei şi ale sistemului
- Rezolvare de probleme (eng. *Problem-solving*)
  - *Modelele* sunt folosite pentru găsirea unei soluţii
  - Evaluarea diferitelor alternative este, de multe ori, empirică
- Acumulare de cunoştinţe (eng. *Knowledge aquisition*)
  - Modelarea presupun colectare de informaţii, organizarea şi formalizarea lor
  - Acumularea de cunoştinţe nu este un proces secvenţial; o informaţie nouă poate invalida modelele anterioare
- Argumentare (eng. *Rationale*)
  - Orice decizie luată ar trebui documentată prin menţionarea contextului, a alternativelor posibile şi a argumentelor aferente
  - Aceasta permite înţelegerea impactului unei eventuale schimbări asupra sistemului

# Modelare

- Scopul ştiinţelor, în general, este *descrierea şi înţelegerea sistemelor complexe*
  - ştiinţe naturale (clasice), ex.: fizică, chimie, biologie
  - ştiinţe sociale (clasice), ex.: psihologie, sociologie
  - ştiinţe "ale artificialului" (recente), ex.: ştiinţa calculatoarelor (eng. *computer science*)
- Tehnica principală folosită de ştiinţele clasice pentru gestionarea complexităţii este *modelarea*
- **Definiţia 1.3 [Model]**: Un *model* este o reprezentare abstractă a unui sistem, care ne permite să răspundem unor întrebări cu privire la acel sistem. [1]
- **Definiţia 1.4 [Modelare, model]**: *Modelarea* este procesul de reprezentare a elementelor sau esenţei unui sistem sau fenomen. *Modelul* este o reprezentare simplificată a unui sistem sau fenomen, împreună cu orice ipoteze necesare pentru a descrie sistemul sau a explica fenomenul. [7]

# Modelare (cont.)

- Modele sunt utile pentru studiul sistemelor
  - prea mari/complexe
  - prea mici
  - costisitor/periculos de experimentat în realitate
  - care nu mai există, care se presupune că ar exista sau care urmează a fi construite

- Inginerii soft au nevoie să construiască
  - *modele ale domeniului problemei* (eng. *application domain models*) - pentru a înţelege mediul în care sistemul operează
  - *modele ale domeniului soluţiei* (eng. *solution domain models*) - pentru a înţelege sistemul care va fi construit şi pentru a evalua soluţiile şi alternativele posibile
- Avantajul major al ingineriei software orientate-obiect
  - *modelul domeniului soluţiei se obţine natural ca o transformare (rafinare) a modelului domeniului problemei*

# Rezolvare de probleme

- *Ingineria* este o activitate de *rezolvare de probleme* în 5 paşi
  I1. Formularea problemei
  I2. Analiza problemei
  I3. Căutarea de soluţii
  I4. Alegerea unei soluţii potrivite
    - evaluări empirice, cu resurse limitate şi informaţii incomplete
  I5. Specificarea soluţiei alese
- Dezvoltarea de soft orientată obiect include, în general, 6 activităţi

  D1. Colectarea cerinţelor (corespondent I1)
  D2. Analiza cerinţelor (corespondent I2)
  D3. Proiectarea de sistem (corespondent I3, I4)
  D4. Proiectarea obiectuală (corespondent I3, I4)
  D5. Implementarea (corespondent I5)
  D6. Testarea
- Ingineria softului este o activitate *inginerească*, *nu algoritmică*
  - presupune experimentare, reutilizare de şabloane, evoluţia iterativă a soluţiei
  - ceea ce o diferenţiază de rezolvarea de probleme din alte domenii inginereşti sunt *schimbările ce au loc în domeniul problemei şi al soluţiei în timpul rezolvării problemei*

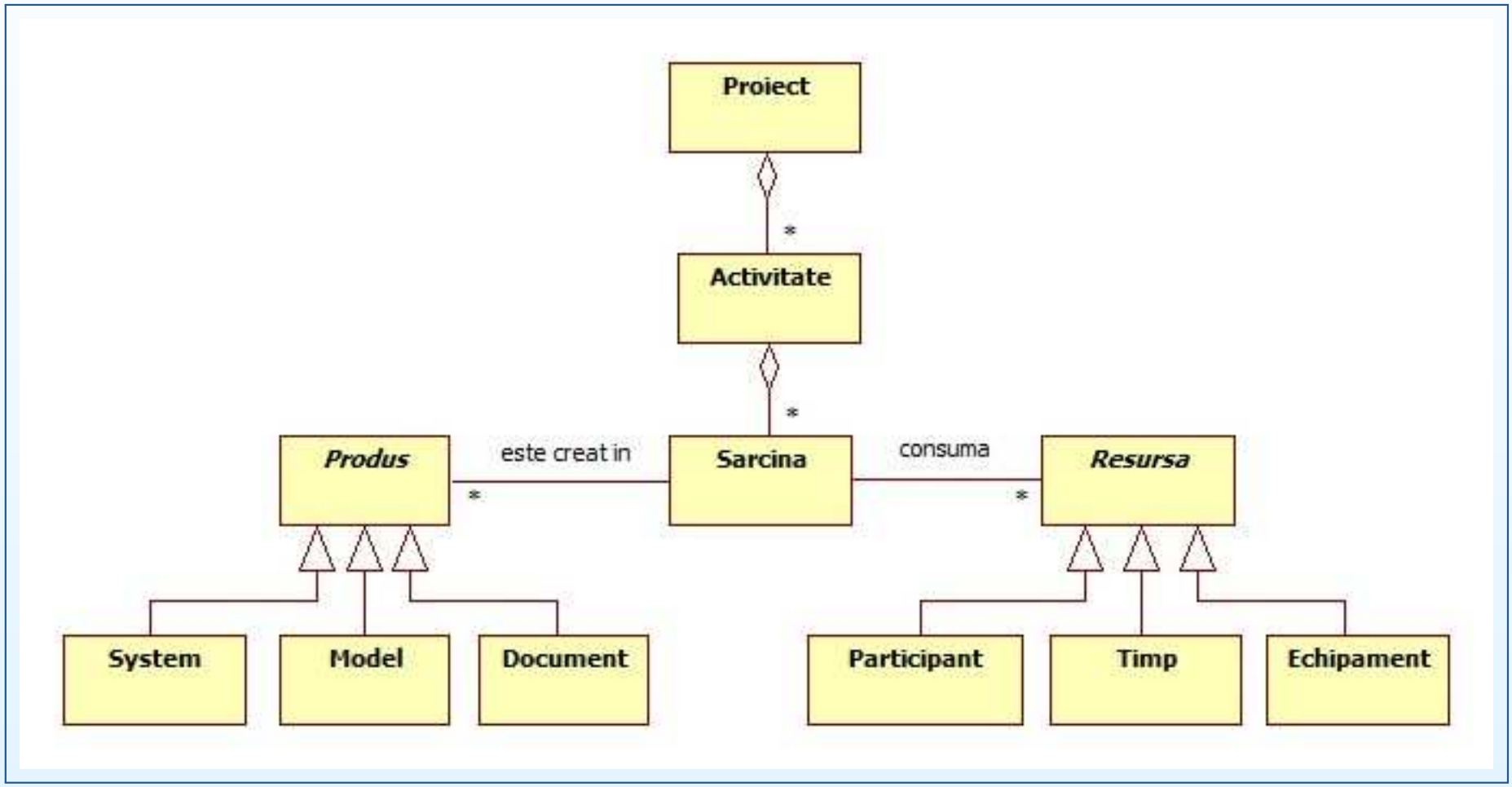
# Acumulare de cunoştinţe

- Nu este un proces secvenţial, întrucât o informaţie nou apărută poate invalida toate modelele anterior dezvoltate
  - Chiar dacă sistemul este 90% terminat, trebuie să existe disponibilitatea psihologică de a o lua de la capăt
- Acumularea secvenţială de cunoştinţe corespunde aşa numitei *bucket theory of the mind*
- În dezvoltarea softului, *bucket theory of the mind* se traduce în *modelul cascadă* al procesului soft
- Procese care evită dezavantajele modelului secvenţial/cascadă: *risk-based development*, *issue-based development*
- Problema majoră a modelelor de dezvoltare non-secvenţiale constă în dificultatea de a fi gestionate eficient

# Argumentare

- Deşi modelul domeniului se stabilizează odată ce dezvoltatorii capătă o bună înţelegere a problemei, modelul soluţiei continuă să sufere modificări, ca urmare a
  - erorilor de proiectare şi implementare descoperite la testare
  - problemelor de utilizabilitate raportate de clienţi
  - apariţiei unor noi tehnologii
- Modificarea softului necesită nu doar înţelegerea componentelor şi comportamentului curent, ci şi cunoaşterea raţionamentelor din spatele deciziilor luate
- Înregistrarea şi accesarea acestor raţionamente este un proces netrivial, ca urmare a
  - volumului mare de informaţie: fiecărei decizii îi corespund, cel mai probabil, diferite alternative ce au fost considerate, discutate, evaluate
  - caracterului implicit al unor decizii luate pe baza experienţei, fără o evaluare explicită a alternativelor posibile

# Concepte ale ingineriei sistemelor soft



# Participanţi şi roluri

- Dezvoltarea unui sistem soft presupune colaborarea a multe persoane, cu interese şi specializări diferite, referite ca şi *participanţi*
  - *Clientul* comandă şi achită sistemul
  - *Dezvoltatorii* construiesc sistemul
  - *Project managerul* planifică proiectul şi coordonează dezvoltatorii şi clientul
  - *Utilizatorii* folosesc sistemul
- Un *rol* referă o mulţime de responsabilităţi în cadrul proiectului
  - Un rol este asociat cu o mulţime de sarcini şi atribuit unui participant
  - Acelaşi participant poate îndeplini mai multe roluri
- **Exemplu**: sistemul AutomatBilete
  - AutomatBilete (AB) este un automat care distribuie bilete de tren. Călătorii pot opta pentru un bilet valabil pentru o singură călătorie sau pentru un card valabil o zi sau o săptămână. Sistemul calculează preţul biletului funcţie de zona în care va călători clientul şi de tipul acestuia - adult sau copil. Sistemul trebuie să gestioneze diferite excepţii, printre care tranzacţii nefinalizate de client, plăţi cu bancnote mari, precum şi pana de resurse - bilete, rest sau curent.

# Participanţi şi roluri (cont.)

| Rol | Responsabilităţi | Exemple |
|---|---|---|
| Client | furnizarea cerinţelor sistemului, fixarea datei de livrare, a bugetului alocat si a criteriilor de calitate | Companie feroviară |
| Utilizator | oferirea de cunoştinţe specifice domeniului (eng. *domain knowledge*) | Călători |
| Manager | gestionarea resurselor oferite de client, contractarea de personal, instruirea acestuia, atribuirea sarcinilor de lucru, monitorizarea progresului | Andrew (PM) |
| Dezvoltator | construcţia sistemului, incluzând specificaţie, proiectare, implementare, testare | John (analist)<br>Marc(programator)<br>Zoe(tester) |
| Specialist HCI | utilizabilitatea sistemului | Zoe (specialist HCI) |
| Responsabil documentaţie tehnică | documentaţia livrată clientului; discută cu dezvoltatori, manageri şi utilizatori, pentru a înţelege sistemul | John |

# Sisteme şi modele

- *System* = un ansamblu de părţi interconectate
  - Ex.: Sistemul AutomatBilete
  - Ex.: Proiectul de dezvoltare al sistemului AutomatBilete
- *Model* = orice abstractizare a sistemului
  - Ex.: Specificaţia sistemului AutomatBilete
  - Ex.: Scheme ale circuitelor electrice ale acestuia
  - Ex.: Modele ale softului său

  - Ex.: Planul proiectului AutomatBilete
  - Ex.: Alocarea bugetul său
  - Ex.: Termenele stabilite

# Produse

- *Produs* (eng. *work product*) = artefact realizat în timpul procesului de dezvoltare
  - Ex.: un document sau o componentă soft pentru dezvoltatori sau client
  - Clasificare
    - Produse de lucru interne (eng. *internal work products*) - utilizate doar în cadrul proiectului
    - Produse livrabile (eng. *deliverables*) - realizate pentru client şi specificate în contract

| **Produs** | **Tip** | **Descriere** |
|---|---|---|
| Specificaţie | Livrabil | Descrie detaliat sistemul AB din perspectiva utilizatorului. Este utilizată ca şi contract între client şi dezvoltator. |
| Manual de operare | Livrabil | Este utilizat de către angajaţii companiei feroviare responsabili cu instalarea şi configurarea sistemului AB. Descrie, spre exemplu, modul în care poate fi schimbat preţul biletelor. |
| Raport de stare | Intern | Descrie, pentru o dată anume, sarcinile terminate şi cele în lucru. Produs pentru manager, inaccesibil clientului. |
| Manual de teste | Intern | Înregistrează defectele prototipului AB şi starea lor curentă (eliminate, în lucru). Produs de tester, inaccesibil clientului. |

# Activităţi, sarcini şi resurse

- *Activitate* = o mulţime de sarcini realizate cu un anumit scop. Activităţile pot fi compuse din alte (sub)activităţi
  - Colectarea cerinţelor este o activitate având drept scop definirea funcţionalităţilor sistemului şi a constrângerilor impuse asupra lui
  - Gestiunea proiectului este o activitate având drept scop monitorizarea şi controlul proiectului, astfel încât acesta să-şi atingă obiectivele (termen, buget, calitate)
  - Predarea la beneficiar este o activitate având drept scop instalarea sistemului la o locaţie operaţională
    - Activitatea de predare include o activitate de instalare a softului şi una de instruire a operatorilor
- *Sarcină* = o unitate atomică, gestionabilă de lucru
  - Managerul o atribuie dezvoltatorului, dezvoltatorul o îndeplineşte, managerul urmăreşte progresul şi finalizarea sarcinii
  - Sarcinile consumă resurse, generează produse şi depind de produsele altor sarcini
- *Resursă* = bun utilizat pentru realizarea unor sarcini
  - Ex.: timp, echipamente, persoane

# Activităţi, sarcini şi resurse (cont.)

| Exemplu | Tip | Descriere |
|---|---|---|
| Colectarea cerinţelor | Activitate | Include obţinerea şi validarea cerinţelor şi a cunoştinţelor legate de domeniul problemei de la client şi utilizatori. Produce specificaţia sistemului. |
| Dezvoltarea cazului de test "Pană rest" | Sarcină | Se referă la verificarea comportamentului sistemului AB în cazul în care rămâne fără bani şi nu poate returna rest. Include specificarea contextului, a datelor de intrare şi a rezultatelor aşteptate. Atribuită lui Zoe (tester). |
| Revizuirea cazului de utilizare "Accesare help online" pentru evaluarea utilizabilităţii | Sarcină | Se referă la detectarea problemelor de utilizabilitate în accesarea funcţionalităţilor de help online. Atribuită lui Zoe (specialist HCI). |
| Baza de date "Tarife" | Resursă | Include un exemplu de tarifare şi o structurare pe zone a reţelei. Oferită de client pentru colectarea cerinţelor şi testare. |

# Cerinţe funcţionale şi nefuncţionale

- *Cerinţă funcţională* = specificare a unei funcţionalităţi pe care sistemul va trebui să o ofere
  - Ex.: Utilizatorul va putea să achiziţioneze bilete
  - Ex.: Utilizatorul va putea să acceseze informaţii legate de tarifare
- *Cerinţă nefuncţională* = constrângere legate de operarea sistemului (fără a fi asociată unei funcţionalităţi anume)
  - Ex.: Sistemul va oferi feedback utilizatorului în mai puţin de o secundă
  - Ex.: Culorile utilizate în interfaţa grafică vor fi cele ale companiei
  - Ex.: Se va folosi o anumită platformă hardware
  - Ex.: Se va asigura compatibilitatea cu un sistem mai vechi

# Notaţii, metode, metodologii

- *Notaţie* = mulţime de reguli, textuale sau grafice, folosite pentru reprezentarea modelelor
  - Ex.: UML (Unified Modeling Language) - notaţie grafică pentru reprezentarea modelelor orientate obiect
  - Ex.: OCL (Object Constraint Language) - notaţie textuală pentru reprezentarea constrângerilor pe modelele UML
  - Ex:. diagrama de flux de date, diagrama de flux de control - notaţii grafice pentru reprezentarea modelelor în modelarea clasică / structurată
  - Ex.: B - notaţie formală textuală pentru reprezentarea sistemelor, bazată pe logică şi teoria mulţimilor
- *Metodă* = tehnică cu caracter repetabil, constând într-o succesiune de paşi aplicaţi în scopul rezolvării unei anumite probleme
  - Ex.: o reţetă este o metodă de a găti un anumit fel de mâncare
  - Ex.: un algoritm de sortare este o metodă de ordonare a unui şir
- *Metodologie* = colecţie de metode folosite pentru rezolvarea unei anumite clase de probleme, incluzând informaţii referitoare la modalitatea de aplicare a acestora (când şi cum ar trebui aplicate)

# Notaţii, metode, metodologii (cont.)

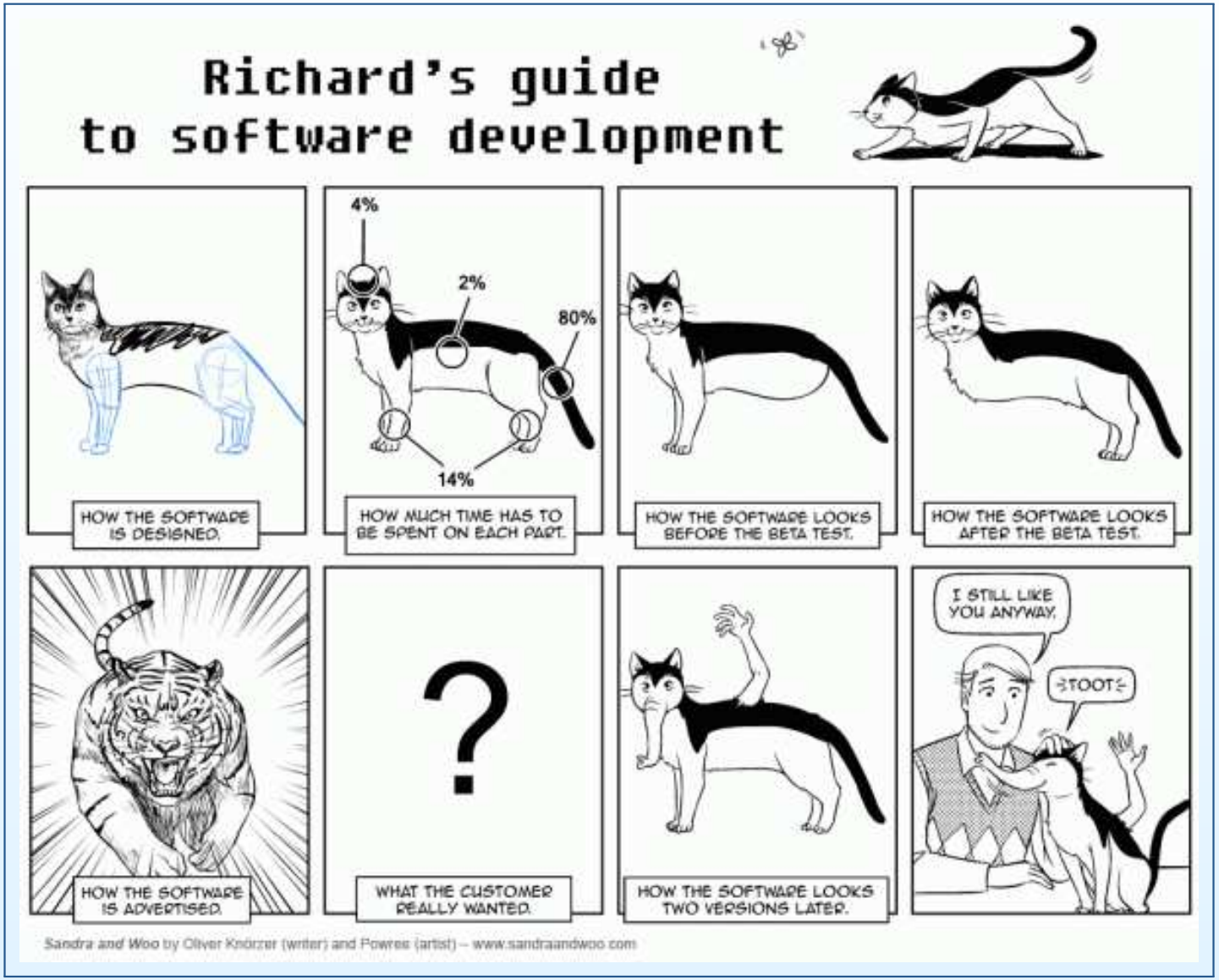
- Ex.: o carte de bucate raw reprezintă o metodologie de preparare a alimentelor raw, în cazul în care conţine indicaţii legate de utilizarea ingredientelor sau substituirea acestora
- Ex.: OMT (Object Modeling Technique), OOSE (Object Oriented Software Engineering), metodologia Booch, USDP (Unified Software Development Process) sau Catalysis sunt metodologii de dezvoltare a softului

- Metodologiile de dezvoltare a softului descompun procesul în activităţi
  - OMT oferă metode pentru 3 activităţi
    - Analiză
    - Proiectare de sistem
    - Proiectare obiectuală
  - USDP oferă metode pentru 3 activităţi
    - Colectarea cerinţelor
    - Analiză
    - Proiectare

# Activităţi ale procesului de dezvoltare a softului

- Activităţi tehnice ale ingineriei soft orientate obiect (conform metodologiei din [1])
  - Colectarea cerinţelor (eng. *Requirements Elicitation*)
  - Analiza cerinţelor (eng. *Analysis*)
  - Proiectarea de sistem (eng. *System Design*)
  - Proiectarea obiectuală (eng. *Object Design*)
  - Implementarea (eng. *Implementation*)
  - Testarea (eng. *Testing*)
- Aceste activităţi gestionează complexitatea prin construirea şi validarea de modele ale domeniului problemei şi domeniului soluţiei

# Software Engineering, now with cats



# Colectarea cerinţelor

- Finalitatea acestei etape o constituie definirea, de către client şi dezvoltatori, a scopului/cerinţelor sistemului
- Produse
  - *Modelul funcţional*
    - descriere a sistemului în termeni de actori şi cazuri de utilizare
      - *Actor* = rol jucat de o entitate externă sistemului, care interacţionează cu sistemul (utilizator uman, alt calculator/dispozitiv/sistem)
      - *Caz de utilizare* = secvenţă generală de evenimente descriind toate interacţiunile posibile între un actor şi sistem, pentru îndeplinirea unei anumite funcţionalităţi
    - instrumente de reprezentare
      - *diagramă UML a cazurilor de utilizare*
      - sabloane pentru descrierea textuală explicită a cazurilor de utilizare
  - *Cerinţele nefuncţionale*

# Colectarea cerinţelor (cont.)

- Ex.: cazul de utilizare *Achiziţionează bilet pentru o călătorie*

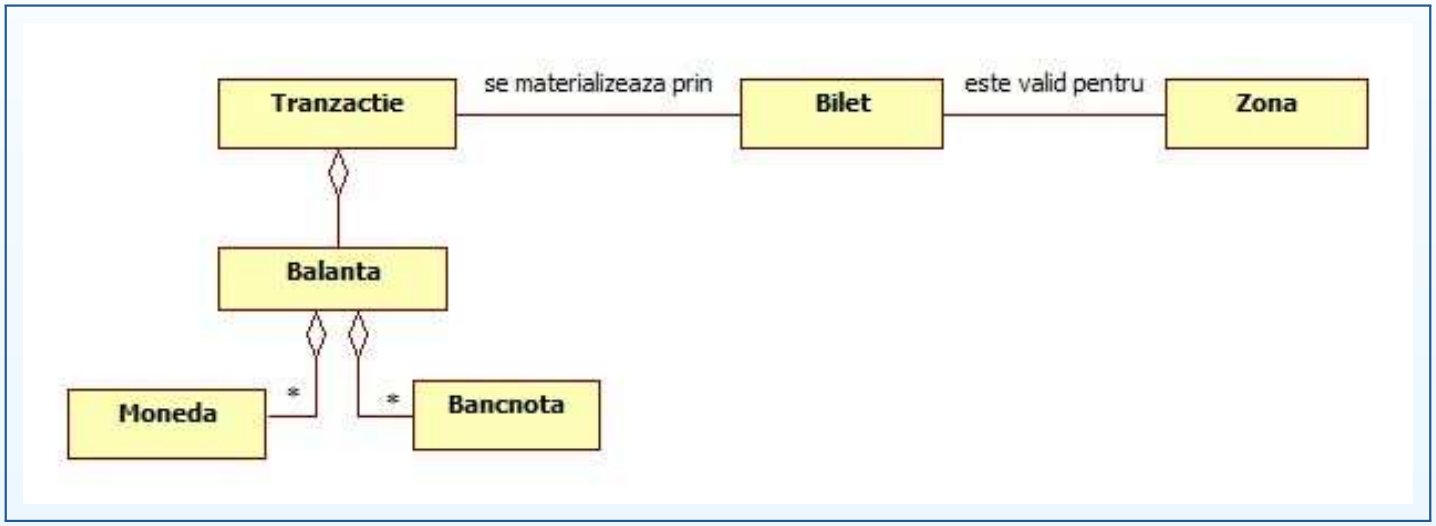
| | |
|---|---|
| **Nume caz de utilizare** | *Achiziţionează bilet pentru o călătorie* |
| **Actor** | Iniţiat de *Călător* |
| **Flux de evenimente** (scenariu normal) | 1. *Călătorul* selectează zona staţiei destinaţie.<br>2. Sistemul *AutomatBilete* afişează preţul biletului.<br>3. *Călătorul* inserează o sumă de bani cel puţin egală cu preţul biletului.<br>4. Sistemul *AutomatBilete* oferă biletul solicitat şi restul. |
| **Condiţie de intrare** | *Călătorul* se află în faţa automatului, localizat în staţia de plecare sau într-o altă staţie. |
| **Condiţie de ieşire** | *Călătorul* primeşte biletul solicitat şi restul. |
| **Cerinţă de calitate** | În cazul în care tranzacţia nu se finalizează, după un minut de inactivitate, sistemul restituie călătorului întreaga sumă inserată. |

# Analiza cerinţelor

- În cadrul acestei etape, analiştii derivă din descrierea cazurilor de utilizare ale etapei precedente un model obiectual al sistemului
  - Scopul este realizarea unui model *complet*, *consistent* şi *neambiguu*
  - În cadrul procesului de realizare a modelului obiectual de analiză, analiştii identifică inconsistenţe, incompletitudini sau ambiguităţi în modelul funcţional, pe care le rezolvă cu clientul
- Modelul obiectual de analiză poate fi descris
  - în termenii structurii sale $\Longrightarrow$ *model conceptual*, reprezentat printr-o *diagramă de clase*
  - în termenii comportamentului său $\Longrightarrow$ *model dinamic*, reprezentat prin *diagrame de interacţiune*
- Modelul obiectual de analiză conţine *doar concepte caracteristice domeniului problemei* (fără referire la domeniul soluţiei) !
  - În colectarea şi analiza cerinţelor răspundem la întrebarea "*ce*?" (funcţionalităţi oferă sistemul / constrângeri se impun asupra lui / concepte manipuează), fără a face vreo referire la modul "*cum*?" se realizează acele cerinţe (probleme de proiectare)

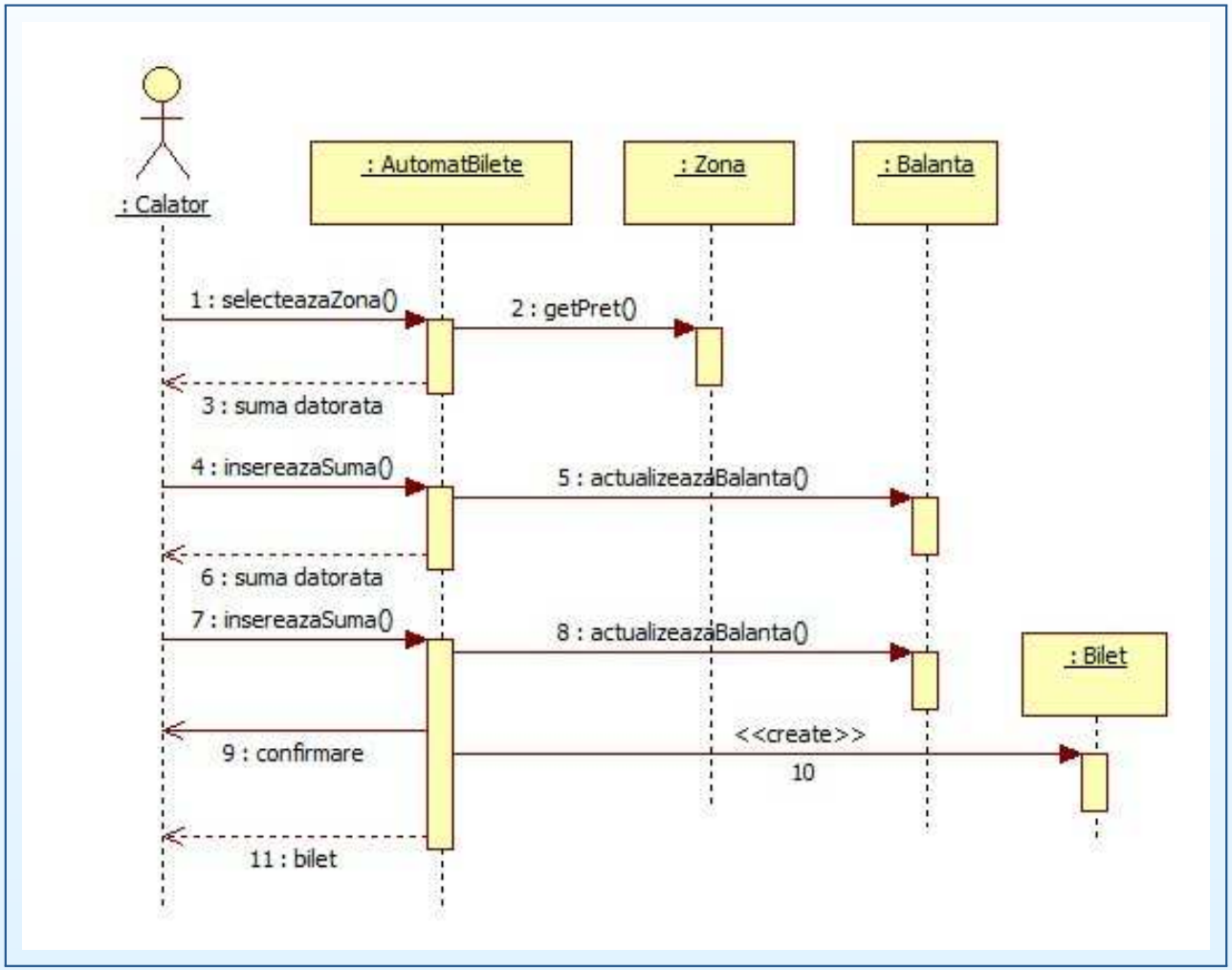
# Analiza cerinţelor (cont.)

- Ex.: model conceptual pentru sistemul *AutomatBilete* (reprezentat printr-o diagramă UML de clase)



# Analiza cerinţelor (cont.)

- Ex.: model dinamic pentru sistemul *AutomatBilete* (reprezentat printr-o diagramă UML de secvenţă)

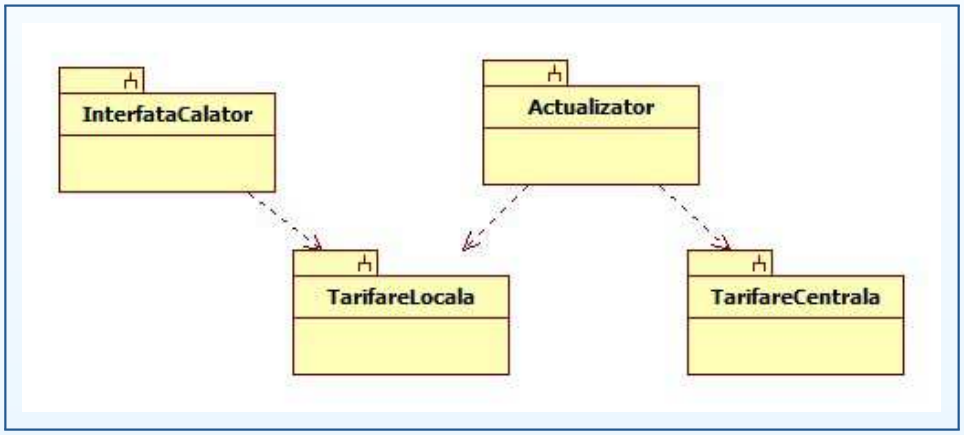


# Proiectarea de sistem

- Scopul acestei etape constă în definirea obiectivelor de proiectare şi descompunerea sistemului în subsisteme
  - În cadrul acestei activităţi, se decide asupra strategiilor utilizate pentru dezvoltarea sistemului
    - platforma hardware/software pe care va rula sistemul
    - modalitatea de asigurare a persistenţei datelor
    - fluxul de control global
    - politicile de control a accesului
- Rezultatul proiectării de sistem conţine descrierea explicită a strategiilor mai sus menţionate, descompunerea sistemului şi o *diagramă de repartiţie a resurselor* (eng. *deployment diagram*) ilustrând maparea harware/software aferentă sistemului
- Ca şi primă etapă a proiectării, proiectarea de sistem trece din domeniul problemei în domeniul soluţiei (răspunde întrebării "*cum?*" şi introduce concepte nefamiliare clientului)

# Proiectarea de sistem (cont.)

- Ex.: descompunere a sistemului *AutomatBilete* (diagramă de clase UML - pachetele corespund subsistemelor, liniile punctate reprezintă dependenţe)




  - Subsistemul *InterfaţăCălător* colectează inputul călătorului şi asigură feedback (afişare preţ bilete, returnare rest, etc.)
  - Subsistemul *TarifareLocală* calculează preţul biletelor utilizând o bază de date locală
  - Subsistemul *TarifareCentrală* păstrează o copie centralizată a bazei de date cu tarife
  - Subsistemul *Actualizator* este responsabil cu actualizarea bazelor de date locale ale automatelor în condiţiile schimbării preţului biletelor

# Proiectarea obiectuală

- Scopul proiectării obiectuale este definirea de obiecte/clase din domeniul soluţiei, pentru a realiza legătura dintre modelul obiectual de analiză şi platforma hardware/software stabilită la proiectarea de sistem
- Proiectarea obiectuală include
  - descrierea detaliată a interfeţelor obiectelor şi subsistemelor
  - selectarea componentelor ce urmează a fi reutilizate
  - restructurarea modelului obiectual pentru a satisface obiectivele de proiectare (generalitate/extensibilitate)
  - optimizarea modelului obiectual în vederea asigurării obiectivelor de performanţă
- Rezultatul proiectării obiectuale îl constituie un model obiectual detaliat, îmbogăţit cu constrângeri şi descrieri precise ale tuturor elementelor componente

# Implementarea şi testarea

- Implementarea presupune translatarea modelului obiectual de proiectare (modelul domeniului soluţiei) în cod sursă
- Testarea presupune identificarea diferenţelor între sistemul implementat şi modelele sale, ca urmare e execuţiei sistemului (sau a unor părţi ale acestuia) cu seturi de date de test
  - *Testarea unitară* - se compară modelul obiectual de proiectare cu fiecare obiect şi subsistem
  - *Testarea de integrare* - se integrează diferite subsisteme, comparându-se cu modelul corespunzător proiectării de sistem
  - *Testarea de sistem* - se compară comportamentul sistemului în ansamblu cu modelul cerinţelor
- Scopul testării este identificarea a cât mai multe defecte, permiţând remedierea lor anterior predării sistemului la beneficiar
- Planificarea testelor se face simultan cu dezvoltarea sistemului
  - Testele de sistem se planifică în timpul colectării şi analizei cerinţelor
  - Testele de integrare se planifică în timpul proiectării de sistem
  - Testele unitare se planifică în timpul proiectării obiectuale