

Ingineria Sistemelor Soft

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!



How the customer explained it



How the project leader understood it



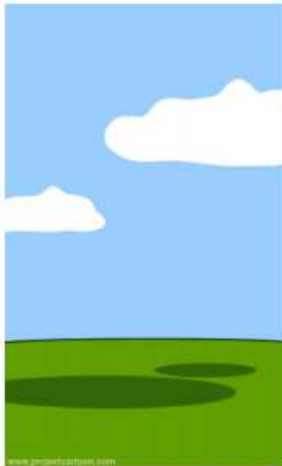
How the analyst designed it



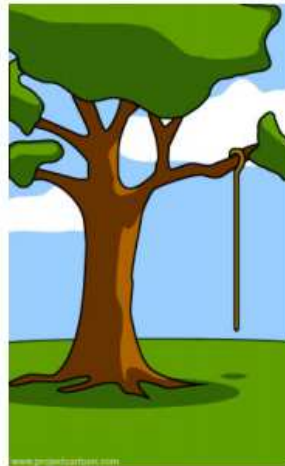
How the programmer wrote it



How the business consultant described it



How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed

Obiective curs

- Înțelegerea conceptelor legate de modelarea softului
- Cunoașterea și aplicarea tehnicilor de dezvoltare a softului pe baza modelelor
- Familiarizarea cu limbajul UML
- Abilitatea de a utiliza instrumente CASE
- Cunoașterea etapelor ciclului de viață al softului și a modelelor de procese soft
- Familiarizarea cu unele dintre metodologiile de dezvoltare, tradiționale sau agile
- Însușirea 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 (notițe curs, materiale seminar, cerințe laborator)
Team: Ingineria sistemelor soft RO 2023, Files/Class Materials
Team code: mov3xvw

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* (3rd 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, *OCF 2.3.1 Specification - 2012 ISO standard*, <https://www.omg.org/spec/OCF/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, *OCF 2.4 Specification*, 2014,
<https://www.omg.org/spec/OCF/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 cerințelor*
- Curs 4: *Analiza cerințelor*
- Curs 5: *Proiectarea de sistem*
- Curs 6: *Proiectarea obiectuală - Șabloane de proiectare*
- Curs 7: *Proiectarea obiectuală - Specificarea interfețelor folosind OCL*
- Curs 8: *Implementarea sistemului. Transformarea modelelor în cod*
- Curs 9: *Testarea sistemului*
- Curs 10: *Ciclul de viață 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:
 - Motivație
 - Definiții
 - Evoluție
 - Eșecuri celebre
 - Provocări
- Ce presupune ingineria sistemelor soft?
 - Modelare
 - Rezolvare de probleme (eng. *problem solving*)
 - Acumulare de cunoștințe (eng. *knowledge acquisition*)
 - Argumentare (eng. *rationale*)
- Concepte ale ingineriei sistemelor soft
 - Participanți și roluri
 - Sisteme și modele
 - Produse (eng. *work products*)
 - Activități, sarcini și resurse
 - Cerințe funcționale și nefuncționale
 - Notății, metode și metodologii

Sumar Curs 1 (cont.)

- Activități ale procesului de dezvoltare a softului
 - Colectarea cerințelor
 - Analiza cerințelor
 - Proiectarea de sistem
 - Proiectarea obiectuală
 - Implementarea
 - Testarea

Ingineria sistemelor soft: Motivație

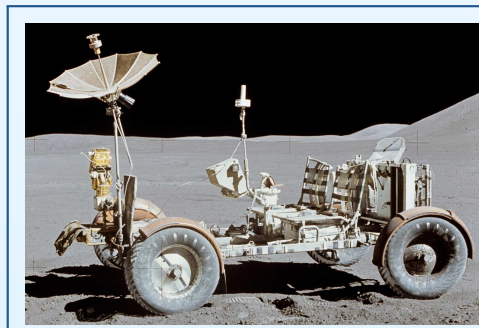
- *Criza software* de la sfârșitul anilor '60, generată de *dezvoltarea artizanală* a programelor
 - calitate slabă a softului - rezultată din cerințe nerespectate, lipsă de fiabilitate, întreținere dificilă
 - nerespectarea termenelor de livrare
 - depășirea bugetului alocat
- Dezvoltatorii de soft erau incapabili să stabilească obiective concrete, să anticipeze resursele necesare și să gestioneze așteptările clienților

Promisă:



Luna

Dezvoltat:



Vehicul lunar

Livrat:



O pereche de roți pătrate

Ingineria sistemelor soft: Definiții

- 1968: prima conferință NATO de Software Engineering - Bavaria
 - nevoia unei *abordări sistematice și disciplinate* a dezvoltării softului
- **Definiția 1.1 [Ingineria Programării, eng. *Software Engineering*]:** *Stabilirea și punerea în practică a unor principii ingineresti solide, care să producă aplicații soft fiabile și care să funcționeze eficient pe mașini reale. (Fritz Bauer, 1968)*
- **Definiția 1.2 [Ingineria Programării]:** *Aplicarea unei abordări sistematice, disciplinate și cuantificabile la dezvoltarea, operarea și întreținerea softului, mai exact aplicarea ingineriei la soft. [6]*

Ingineria sistemelor soft: Evoluție

- "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 și 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 depășirea termenelor de predare, a bugetului alocat, precum și la probleme de utilizabilitate
- Costurile de întreținere 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, față de descoperirea ei la începutul dezvoltării sistemului

Ingineria sistemelor soft: Eșecuri celebre

- *Therac-25 (1985-1987)*
 - Între 1985 și 1987, sistemul medical Therac-25, utilizat în terapia pacienților cu cancer, a fost implicat în 6 accidente (cu decese și răni grave), aplicând pacienților supradoze mari de radiații (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 împărțire 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 invitația de a se înscrie la o grădiniță. Mary avea 104 ani atunci.

Ingineria sistemelor soft: Eșecuri celebre (cont.)

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

Ingineria sistemelor soft: Provocări

- **Complexitate**

- Sistemele soft au multiple funcționalități
- Sunt construite să îndeplinească un număr mare de obiective
- Constau dintr-un număr mare de componente
- Mulți participanți, cu pregătiri diferite, iau parte la dezvoltarea lor
- Procesul de dezvoltare durează mulți ani

- **Schimbare**

- Cerințele se modifică, la inițiativa clientului sau pe măsură ce dezvoltatorii înțeleg domeniul problemei
- În cazul în care proiectul durează mulți ani, dezvoltatorii se pot schimba
- Sistemul se schimbă, ca urmare a greșelilor descoperite la testare
- Tehnologia se schimbă, de multe ori înainte de finalizarea proiectului

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

Ce presupune ingineria sistemelor soft?

- Modelare (eng. *Modeling*)
 - Permite gestionarea *complexității*, prin focusarea pe aspectele relevante și ignorarea detaliilor
 - Pe parcursul procesului de dezvoltare, se construiesc diverse modele ale domeniului problemei și ale sistemului
- Rezolvare de probleme (eng. *Problem-solving*)
 - *Modelele* sunt folosite pentru găsirea unei soluții
 - Evaluarea diferitelor alternative este, de multe ori, empirică
- Acumulare de cunoștințe (eng. *Knowledge acquisition*)
 - Modelarea presupune colectare de informații, organizarea și formalizarea lor
 - Acumularea de cunoștințe nu este un proces secvențial; o informație nouă poate invalida modelele anterioare
- Argumentare (eng. *Rationale*)
 - Orice decizie luată ar trebui documentată prin menționarea contextului, a alternativelor posibile și a argumentelor aferente
 - Aceasta permite înțelegerea impactului unei eventuale schimbări asupra sistemului

Modelare

- Scopul științelor, în general, este *descrierea și înțelegerea sistemelor complexe*
 - științe naturale (clasice), ex.: fizică, chimie, biologie
 - științe sociale (clasice), ex.: psihologie, sociologie
 - științe "ale artificialului" (recente), ex.: știința calculatoarelor (eng. *computer science*)
- Tehnica principală folosită de științele clasice pentru gestionarea complexității este *modelarea*
- **Definiția 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]
- **Definiția 1.4 [Modelare, model]:** *Modelarea* este procesul de reprezentare a elementelor sau esenței 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 înțelege mediul în care sistemul operează
- *modele ale domeniului soluției* (eng. *solution domain models*)
 - pentru a înțelege sistemul care va fi construit și pentru a evalua soluțiile și alternativele posibile

- Avantajul major al ingineriei software orientate-obiect

- *modelul domeniului soluției se obține natural ca o transformare (rafinare) a modelului domeniului problemei*

Rezolvare de probleme

- *Ingineria* este o activitate de *rezolvare de probleme* în 5 pași
 - I1. Formularea problemei
 - I2. Analiza problemei
 - I3. Căutarea de soluții
 - I4. Alegerea unei soluții potrivite
 - evaluări empirice, cu resurse limitate și informații incomplete
 - I5. Specificarea soluției alese
- Dezvoltarea de soft orientată obiect include, în general, 6 activități
 - D1. Colectarea cerințelor (corespondent I1)
 - D2. Analiza cerințelor (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 șabloane, evoluția iterativă a soluției
 - ceea ce o diferențiază de rezolvarea de probleme din alte domenii ingineresti sunt *schimbările ce au loc în domeniul problemei și al soluției în timpul rezolvării problemei*

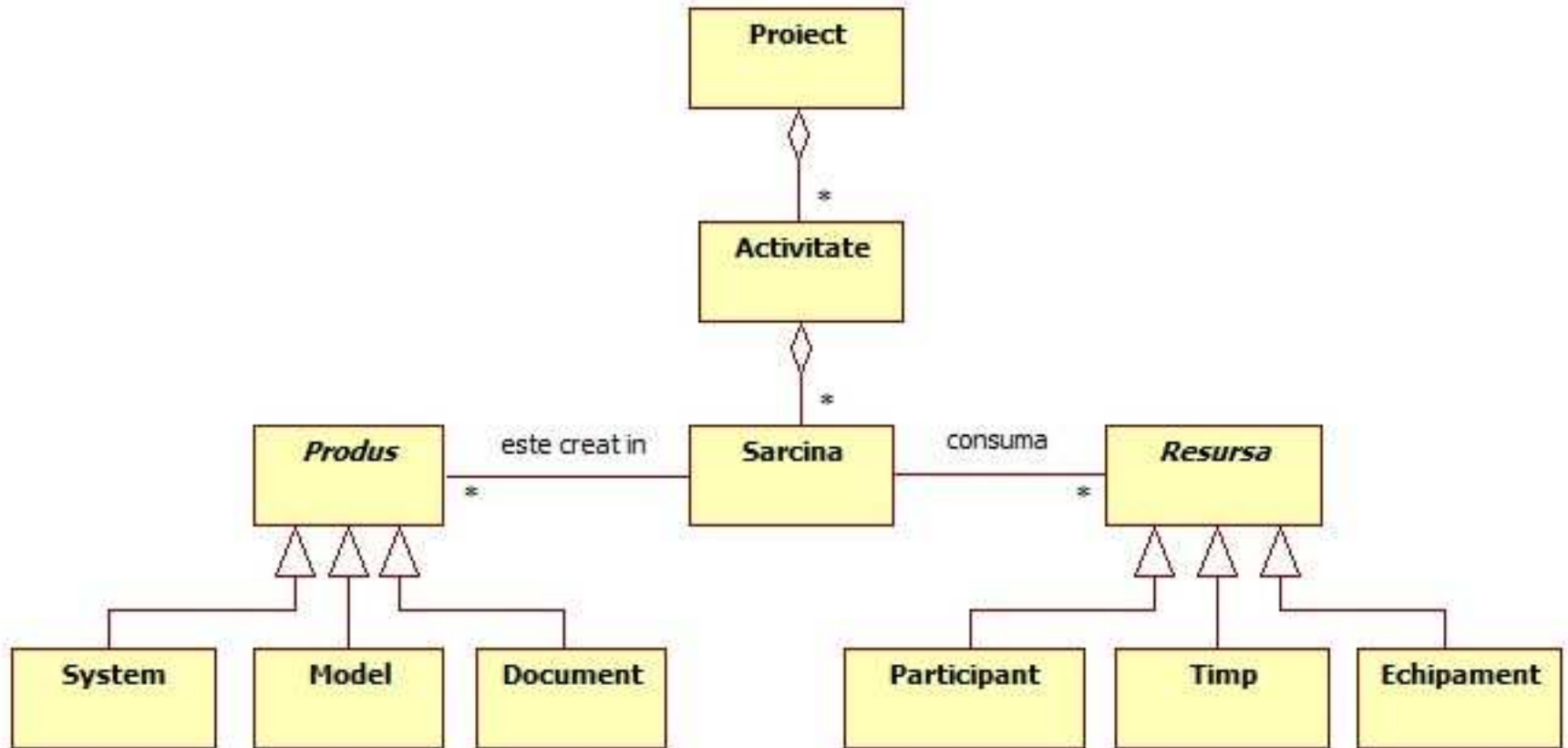
Acumulare de cunoștințe

- Nu este un proces secvențial, întrucât o informație 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 secvențială de cunoștințe corespunde așa 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 secvențial/cascadă: *risk-based development, issue-based development*
- Problema majoră a modelelor de dezvoltare non-secvențiale constă în dificultatea de a fi gestionate eficient

Argumentare

- Deși modelul domeniului se stabilizează odată ce dezvoltatorii capătă o bună înțelegere a problemei, modelul soluției continuă să sufere modificări, ca urmare a
 - erorilor de proiectare și implementare descoperite la testare
 - problemelor de utilizabilitate raportate de clienți
 - apariției unor noi tehnologii
- Modificarea softului necesită nu doar înțelegerea componentelor și comportamentului curent, ci și cunoașterea raționamentelor din spatele deciziilor luate
- Înregistrarea și accesarea acestor raționamente este un proces netrivial, ca urmare a
 - volumului mare de informație: fiecărei decizii îi corespund, cel mai probabil, diferite alternative ce au fost considerate, discutate, evaluate
 - caracterului implicit al unor decizii luate pe baza experienței, fără o evaluare explicită a alternativelor posibile

Concepte ale ingineriei sistemelor soft



Participanți și roluri

- Dezvoltarea unui sistem soft presupune colaborarea a multe persoane, cu interese și specializări diferite, referite ca și *participanți*
 - *Clientul* comandă și achită sistemul
 - *Dezvoltatorii* construiesc sistemul
 - *Project managerul* planifică proiectul și coordonează dezvoltatorii și clientul
 - *Utilizatorii* folosesc sistemul
- Un *rol* referă o mulțime de responsabilități în cadrul proiectului
 - Un rol este asociat cu o mulțime de sarcini și atribuit unui participant
 - Același 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 prețul biletului funcție de zona în care va călători clientul și de tipul acestuia - adult sau copil. Sistemul trebuie să gestioneze diferite excepții, printre care tranzacții nefinalizate de client, plăți cu bancnote mari, precum și pana de resurse - bilete, rest sau curent.

Participanți și roluri (cont.)

Rol	Responsabilități	Exemple
Client	furnizarea cerințelor sistemului, fixarea datei de livrare, a bugetului alocat și a criteriilor de calitate	Companie feroviară
Utilizator	oferirea de cunoștințe specifice domeniului (eng. <i>domain knowledge</i>)	Călători
Manager	gestionarea resurselor oferite de client, contractarea de personal, instruirea acestuia, atribuirea sarcinilor de lucru, monitorizarea progresului	Andrew (PM)
Dezvoltator	construcția sistemului, incluzând specificație, proiectare, implementare, testare	John (analist) Marc(programator) Zoe(tester)
Specialist HCI	utilizabilitatea sistemului	Zoe (specialist HCI)
Responsabil documentație tehnică	documentația livrată clientului; discută cu dezvoltatori, manageri și utilizatori, pentru a înțelege sistemul	John

Sisteme și modele

- *System* = un ansamblu de părți interconectate
 - Ex.: Sistemul AutomatBilete
 - Ex.: Proiectul de dezvoltare al sistemului AutomatBilete
- *Model* = orice abstractizare a sistemului
 - Ex.: Specificația 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 și specificate în contract

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

Activități, sarcini și resurse

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

Activități, sarcini și resurse (cont.)

Exemplu	Tip	Descriere
Colectarea cerințelor	Activitate	Include obținerea și validarea cerințelor și a cunoștințelor legate de domeniul problemei de la client și utilizatori. Produce specificația sistemului.
Dezvoltarea cazului de test "Pană rest"	Sarcină	Se referă la verificarea comportamentului sistemului AB în cazul în care rămâne fără bani și nu poate returna rest. Include specificarea contextului, a datelor de intrare și a rezultatelor așteptate. Atribuită lui Zoe (tester).
Revizuirea cazului de utilizare "Accesare help online" pentru evaluarea utilizabilității	Sarcină	Se referă la detectarea problemelor de utilizabilitate în accesarea funcționalităților de help online. Atribuită lui Zoe (specialist HCI).
Baza de date "Tarife"	Resursă	Include un exemplu de tarificare și o structurare pe zone a rețelei. Oferită de client pentru colectarea cerințelor și testare.

Cerințe funcționale și nefuncționale

- *Cerință funcțională* = specificare a unei funcționalități pe care sistemul va trebui să o ofere
 - Ex.: Utilizatorul va putea să achiziționeze bilete
 - Ex.: Utilizatorul va putea să acceseze informații legate de tarifyare
- *Cerință nefuncțională* = constrângere legate de operarea sistemului (fără a fi asociată unei funcționalități anume)
 - Ex.: Sistemul va oferi feedback utilizatorului în mai puțin de o secundă
 - Ex.: Culoarele utilizate în interfața grafică vor fi cele ale companiei
 - Ex.: Se va folosi o anumită platformă hardware
 - Ex.: Se va asigura compatibilitatea cu un sistem mai vechi

Notății, metode, metodologii

- *Notăție* = mulțime de reguli, textuale sau grafice, folosite pentru reprezentarea modelelor
 - Ex.: UML (Unified Modeling Language) - notăție grafică pentru reprezentarea modelelor orientate obiect
 - Ex.: OCL (Object Constraint Language) - notăție textuală pentru reprezentarea constrângerilor pe modelele UML
 - Ex.: diagrama de flux de date, diagrama de flux de control - notății grafice pentru reprezentarea modelelor în modelarea clasică / structurată
 - Ex.: B - notăție formală textuală pentru reprezentarea sistemelor, bazată pe logică și teoria mulțimilor
- *Metodă* = tehnică cu caracter repetabil, constând într-o succesiune de pași aplicați în scopul rezolvării unei anumite probleme
 - Ex.: o rețetă este o metodă de a găti un anumit fel de mâncare
 - Ex.: un algoritm de sortare este o metodă de ordonare a unui șir
- *Metodologie* = colecție de metode folosite pentru rezolvarea unei anumite clase de probleme, incluzând informații referitoare la modalitatea de aplicare a acestora (când și cum ar trebui aplicate)

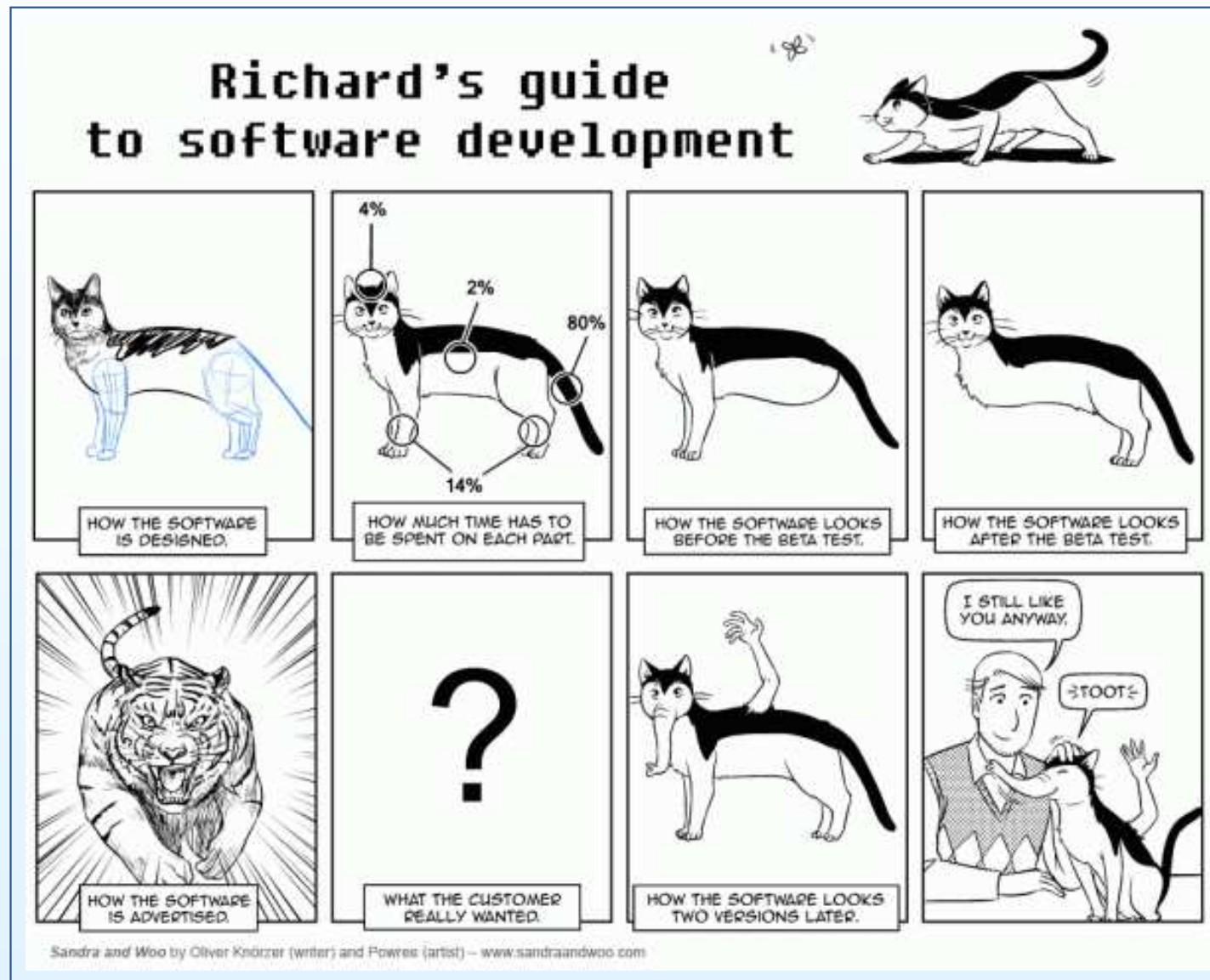
Notății, metode, metodologii (cont.)

- Ex.: o carte de bucate raw reprezintă o metodologie de preparare a alimentelor raw, în cazul în care conține indicații 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 activități
 - OMT oferă metode pentru 3 activități
 - Analiză
 - Proiectare de sistem
 - Proiectare obiectuală
 - USDP oferă metode pentru 3 activități
 - Colectarea cerințelor
 - Analiză
 - Proiectare

Activități ale procesului de dezvoltare a softului

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

Software Engineering, now with cats



Colectarea cerințelor

- Finalitatea acestei etape o constituie definirea, de către client și dezvoltatori, a scopului/cerințelor sistemului
- Produse
 - *Modelul funcțional*
 - descriere a sistemului în termeni de actori și cazuri de utilizare
 - *Actor* = rol jucat de o entitate externă sistemului, care interacționează cu sistemul (utilizator uman, alt calculator/dispozitiv/sistem)
 - *Caz de utilizare* = secvență generală de evenimente descriind toate interacțiunile posibile între un actor și sistem, pentru îndeplinirea unei anumite funcționalități
 - instrumente de reprezentare
 - *diagramă UML a cazurilor de utilizare*
 - sabloane pentru descrierea textuală explicită a cazurilor de utilizare
 - *Cerințele nefuncționale*

Colectarea cerințelor (cont.)

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

Nume caz de utilizare	<i>Achiziționează bilet pentru o călătorie</i>
Actor	Inițiat de <i>Călător</i>
Flux de evenimente (scenariu normal)	<ol style="list-style-type: none">1. <i>Călătorul</i> selectează zona stației destinație.2. Sistemul <i>AutomatBilete</i> afișează prețul biletului.3. <i>Călătorul</i> inserează o sumă de bani cel puțin egală cu prețul biletului.4. Sistemul <i>AutomatBilete</i> oferă biletul solicitat și restul.
Condiție de intrare	<i>Călătorul</i> se află în fața automatului, localizat în stația de plecare sau într-o altă stație.
Condiție de ieșire	<i>Călătorul</i> primește biletul solicitat și restul.
Cerință de calitate	În cazul în care tranzacția nu se finalizează, după un minut de inactivitate, sistemul restituie călătorului întreaga sumă inserată.

Analiza cerințelor

- În cadrul acestei etape, analiștii derivă din descrierea cazurilor de utilizare ale etapei precedente un model obiectual al sistemului
 - Scopul este realizarea unui model *complet, consistent și neambiguu*
 - În cadrul procesului de realizare a modelului obiectual de analiză, analiștii identifică inconsistențe, incompletitudini sau ambiguități în modelul funcțional, pe care le rezolvă cu clientul
- Modelul obiectual de analiză poate fi descris
 - în termenii structurii sale \implies *model conceptual*, reprezentat printr-o *diagramă de clase*
 - în termenii comportamentului său \implies *model dinamic*, reprezentat prin *diagrame de interacțiune*
- Modelul obiectual de analiză conține *doar concepte caracteristice domeniului problemei* (fără referire la domeniul soluției) !
 - În colectarea și analiza cerințelor răspundem la întrebarea "ce?" (funcționalități oferă sistemul / constrângeri se impun asupra lui / concepte manipuează), fără a face vreo referire la modul "cum?" se realizează acele cerințe (probleme de proiectare)

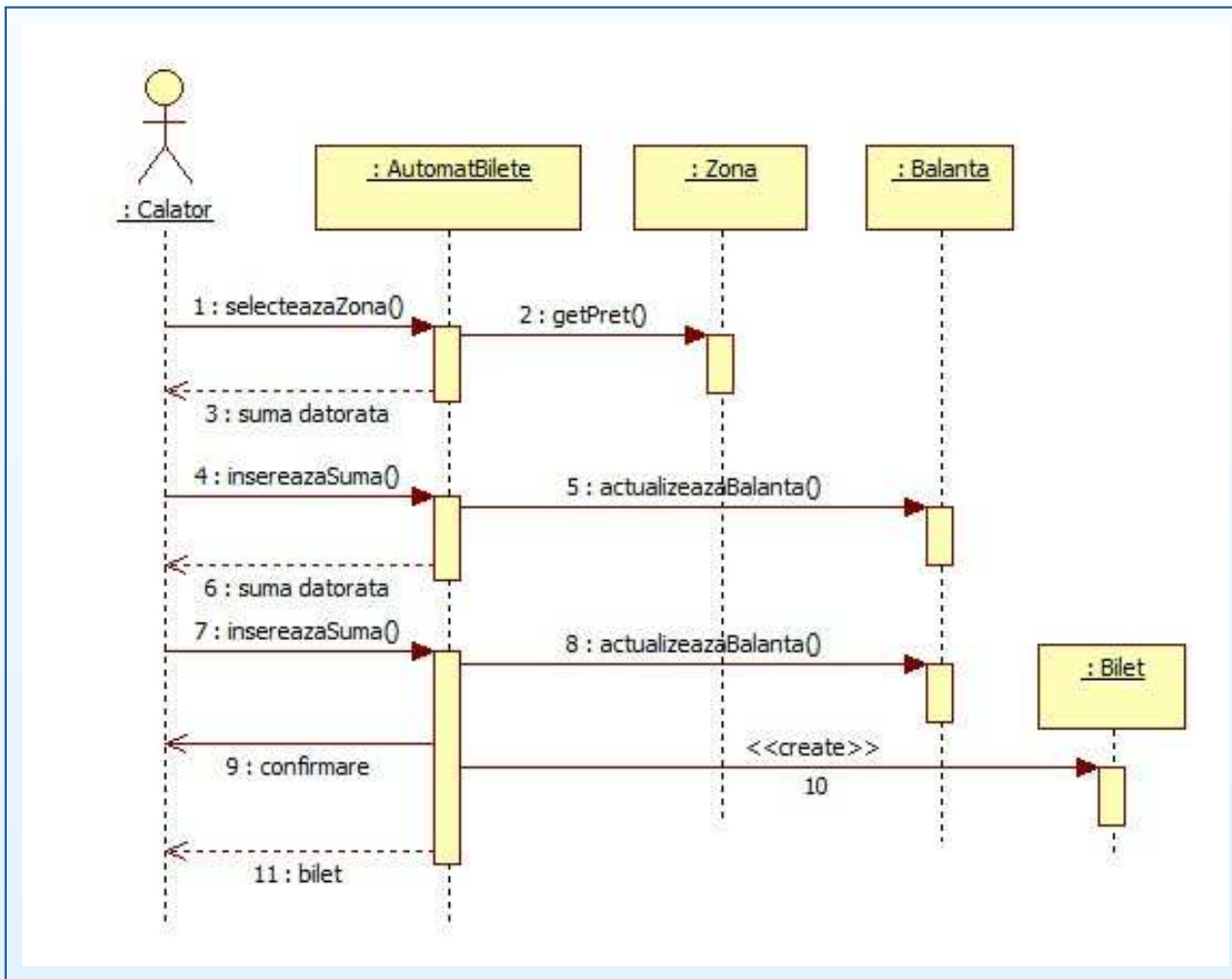
Analiza cerințelor (cont.)

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



Analiza cerințelor (cont.)

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

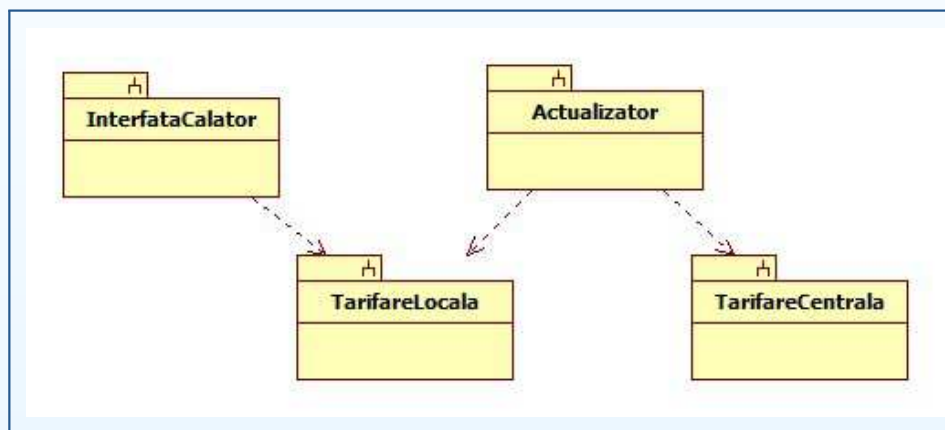


Proiectarea de sistem

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

Proiectarea de sistem (cont.)

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



- Subsistemul *InterfațăCălător* colectează inputul călătorului și asigură feedback (afișare preț bilete, returnare rest, etc.)
- Subsistemul *TarifareLocală* calculează prețul билетelor 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 condițiile schimbării prețului билетelor

Proiectarea obiectuală

- Scopul proiectării obiectuale este definirea de obiecte/clase din domeniul soluției, pentru a realiza legătura dintre modelul obiectual de analiză și platforma hardware/software stabilită la proiectarea de sistem
- Proiectarea obiectuală include
 - descrierea detaliată a interfețelor obiectelor și 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 performanță
- Rezultatul proiectării obiectuale îl constituie un model obiectual detaliat, îmbogățit cu constrângeri și descrieri precise ale tuturor elementelor componente

Implementarea și testarea

- Implementarea presupune translatarea modelului obiectual de proiectare (modelul domeniului soluției) în cod sursă
- Testarea presupune identificarea diferențelor între sistemul implementat și modelele sale, ca urmare a execuției sistemului (sau a unor părți ale acestuia) cu seturi de date de test
 - *Testarea unitară* - se compară modelul obiectual de proiectare cu fiecare obiect și 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 cerințelor
- Scopul testării este identificarea a cât mai multe defecte, permițând remedierea lor anterior predării sistemului la beneficiar
- Planificarea testelor se face simultan cu dezvoltarea sistemului
 - Testele de sistem se planifică în timpul colectării și analizei cerințelor
 - Testele de integrare se planifică în timpul proiectării de sistem
 - Testele unitare se planifică în timpul proiectării obiectuale