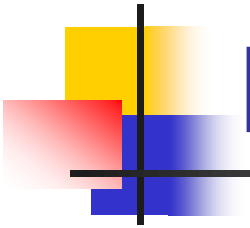
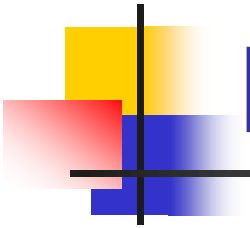


Procesul Unificat

- **Procesul Unificat** (engl. *Unified Proces* (UP)) este procesul software atasat UML [BRJ99,JBR99,Kru03].
- UP angajeaza o **abordare incrementala** in dezvoltarea software.
- UP incorporeaza multe dintre elementele caracteristice ale *Rational Unified Process* (RUP) [Kru03].

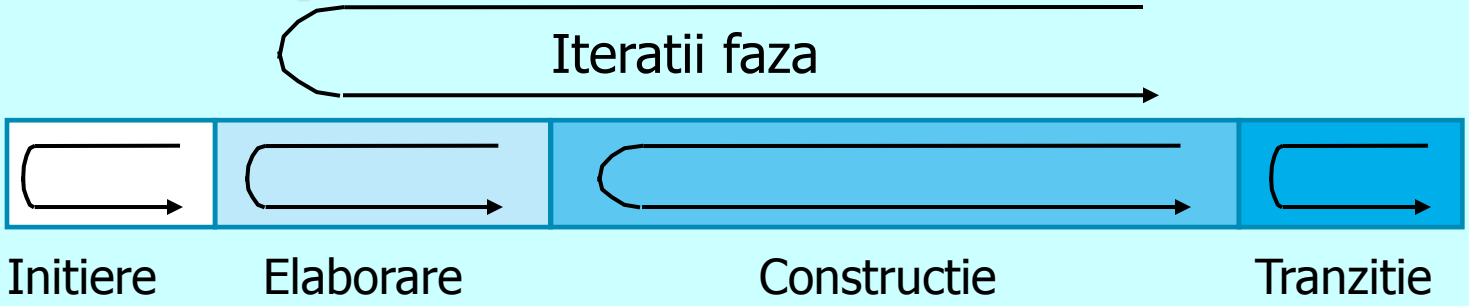


Procesul Unificat

- UP [JBR99,Kru03]
  - este orientat pe componente,
  - utilizeaza UML pentru crearea modelelor OO si
  - are trei caracteristici distinctive:
    - este ghidat de cazurile de utilizare,
    - este centrat pe arhitectura,
    - este iterativ si incremental.

Procesul Unificat

Exista patru faze in UP



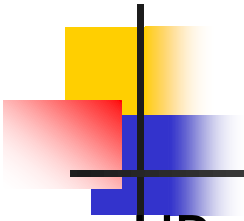
- Fiecare faza consta dintr-un numar de **iteratii**. Fiecare iteratie produce un nou **increment**.
- Dezvoltarea este iterativa si incrementala.
- Cele mai multe riscuri sunt tratate in fazele de initiere si elaborare.
- Majoritatea resurselor sunt consumate in faza de constructie.

In UP o iteratie consta din urmatoarele **fluxuri de baza**:

- Captare cerinte
- Analiza
- Proiectare
- Implementare
- Testare

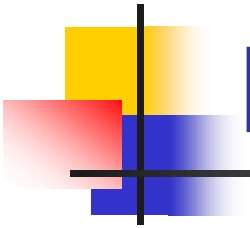
Procesul Unificat

- **Initiere**
  - Se realizeaza o estimare initiala a resurselor necesare.
  - Resursele utilizate in iteratiile din faza de initiere se consuma in special in fluxul de captare a cerintelor.
- **Elaborare**
  - Se dezvolta arhitectura sistem. Prin definitie, faza de elaborare se incheie atunci cand arhitectura se stabilizeaza.
  - Iteratiile din aceasta faza implica toate fluxurile de lucru.
- **Constructie**
  - Proiectare, implementare si testare sistem.
- **Tranzitie**
  - Se realizeaza tranzitia spre mediul utilizator.
  - Rolul fluxurilor de baza este mai mic in aceasta faza.



Procesul Unificat

- UP este ghidat de cazurile de utilizare.
  - Cazurile de utilizare ghideaza intregul proces de dezvoltare, de la captarea cerintelor pana la testarea sistemului.
  - Un caz de utilizare este o descriere a unui set de secvente de actiuni, incluzand variante, ce pot fi efectuate de un sistem si care produc un efect observabil si semnificativ pentru un actor particular [BRJ99,JBR99].
- UP este centrat pe arhitectura.
  - In UP, arhitectura software este o realizare completa (de la specificare pana la testare) a cazurilor de utilizare relevante din punct de vedere arhitectural (arhitectul decide ce este *relevant*).
- UP este iterativ si incremental.
  - Un increment este o realizare a unui set de cazuri de utilizare.
  - Abordarea este **ghidata de riscuri**.



Metode flexibile

- **Metodele flexibile** (engl. *Agile methods*) reprezinta o evolutie recenta a **modelului incremental** [Som01,Som06,Som10].
- Exemple de abordari flexibile includ: XP [Bec99], DSDM [Sta97], SCRUM [SB01], Crystal [Coc01] and FDD [PF02].
- Metodele flexibile se potrivesc cel mai bine pentru aplicatii de dimensiuni mici / medii destinate mediului de afaceri [Som06,Som10].

Metode flexibile

Principiile metodelor flexibile

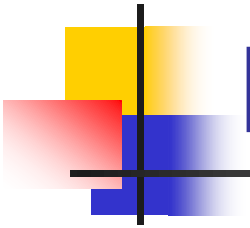
Principiu	Descriere
Implicare client	Clientii ar trebui sa fie implicati direct in intregul proces de dezvoltare. Rolul lor este de a furniza si ordona cerintele dupa prioritati si de a evalua iteratiile sistemului.
Livrare incrementala	Sistemul software este dezvoltat intr-o maniera incrementala. Clientul specifica cerintele pentru fiecare increment.
Oameni nu procese	Calitatile profesionale si de comunicare (intre persoane) ale membrilor echipei de dezvoltatori ar trebui sa fie recunoscute si exploatate. Membrii echipei pot sa dezvolte propriile metode de lucru, fara procese prestabilite. Abordaruile flexibile incurajeaza comunicarea (preferabil fata catre fata, nu prin documente) intre toti participantii la proiect (clienti si dezvoltatori).
Cerinte dinamice	Procesul de dezvoltare ar trebui sa fie ghidat de modificarea cerintelor, iar nu de planuri inflexibile prestabilite.
Simplitate	Complexitatea inutila ar trebui sa fie eliminata din sistemul software si din procesul de dezvoltare. In abordarile flexibile accentul este pe cod, nu pe documentatie.

Metode flexibile

- Dificultati in abordarile flexibile:
  - Poate fi dificil sa se mentina interesul si implicarea clientului in intregul proces de dezvoltare.
  - Membrii echipei de dezvoltatori pot sa nu fie potriviti pentru implicarea si comunicarea intensa ce caracterizeaza metodele flexibile.
  - Ordonarea cerintelor (si a modificarilor) dupa prioritati poate fi dificila, in conditiile implicarii mai multor participanti din partea dezvoltatorului si a clientului.
  - Mentinerea simplitatii necesita munca suplimentara.
  - Ca si in alte abordari iterative contractarea este dificila.

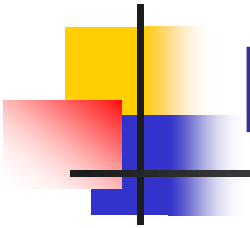
Programare extrema

- **Programarea extrema** (engl. *Extreme Programming* (XP)) este cea mai cunoscuta dintre metodele flexibile.
- XP angajeaza o abordare 'extrema' in dezvoltarea iterativa.
  - Incrementele sunt livrate spre client la fiecare cateva saptamani (in abordarile incrementale conventionale timpul alocat unei iteratii se masoara in luni).
  - Fiecare nou increment este acceptat numai daca toate testele sunt trecute cu succes. Testele (automate) sunt construite inainte de scrierea codului program.



Programare extrema

- Valori in abordarea XP [Bec99]:
  - **Comunicare**
    - XP incurajeaza colaborarea directa intre client si programatori. Un reprezentant al clientului ar trebui sa fie angajat permanent in echipa XP. In XP comunicarea directa frecventa intre participantii la proiect este preferata documentelor scrise.
  - **Simplitate**
    - XP recomanda alegerea celei mai simple solutii la inceput si imbunatatirea acesteia prin refactorizare. Se recunoaste insa ca simplitatea nu se obtine cu usurinta.
  - **Feedback**
    - Aceasta valoare este legata direct de comunicare si simplitate. Programatorii si clientul obtin feedback din partea sistemului prin teste de acceptanta si teste la nivel de modul. Clientul obtine feedback din partea echipei de dezvoltatori atunci cand aceasta (echipa de dezvoltatori) estimeaza timpul necesar pentru implementarea noilor cerinte.
  - **Curaj**
    - Programatorii trebuie sa aiba curajul sa refactorizeze codul, sau chiar sa renunte la parti din cod si sa reproiecteze arhitectura software daca acest lucru devine necesar.



Programare extrema

- In XP toate cerintele sunt exprimate sub forma de **scenarii XP** (engl. *user stories*).
  - **Scenariile** traditionale (**cazuri de utilizare** in UP) sunt descrieri de exemple de sesiuni interactive; fiecare scenariu cuprinde una sau mai multe interactiuni posibile [Som04]. In practica, scenariile XP pot fi mai fragmentare decat scenariile traditionale.
- Scenariile XP sunt scrise de client in limbaj natural, utilizand concepte din domeniul de aplicatie (nu concepte tehnice).
  - In mod tipic, un scenariu XP nu depaseste 3-4 propozitii.

Programare extrema

- Scenariile XP sunt elementele de baza in planificarea proiectelor XP.
  - Daca dezvoltatorul estimeaza ca implementarea unui scenariu necesita mai mult de 2-3 saptamani atunci acesta trebuie descompus in scenarii XP mai mici.
  - Pe baza prioritatilor de afaceri si a estimarilor realizate de dezvoltator, clientul decide care scenarii XP urmeaza sa fie implementate in urmatorul increment al sistemului.
- In abordarea XP dezvoltarea este ghidata de testare. Testele sunt dezvoltate din scenarii XP inainte de design sau implementare.
  - Testele (black box) de acceptanta sunt specificate de client. Programatorii scriu in mod continuu teste la nivel de modul.
  - Se utilizeaza componente de testare automata pentru fiecare nou increment al sistemului.

Programare extrema

Practici XP

Principiu sau practica	Descriere
Planificare incrementala	Cerintele sunt inregistrate pe carduri de scenarii XP. Scenariile XP care urmeaza a fi incluse in urmatorul increment sunt determinate pe baza prioritatilor relative si a timpului disponibil.
Incremente de dimensiuni mici	Se dezvolta mai intai o functionalitate minimala dar relevanta pentru client. Iteratiile care urmeaza sunt frecvente si adauga in mod incremental functionalitate la sistem.
Design simplu	Se proiecteaza numai cerintele planificate pentru iteratia curenta.
Dezvoltare ghidata de testare	Se utilizeaza componente de testare automata. Testele sunt dezvoltate din scenarii XP inainte de design sau implementare.
Perechi de programatori	Programatorii lucreaza in perechi, verificandu-se reciproc.
Proprietate comuna asupra codului	Perechile de programatori actioneaza in toate aspectele proiectului, fiecare putand modifica orice. Se poate vorbi despre o forma de proprietate comuna asupra codului.

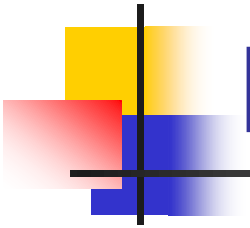
Programare extrema

Practici XP

Principiu sau practica	Description
Refactorizare	Este important ca toti dezvoltatorii sa refactorizeze codul in mod continuu, de indata ce acest lucru este posibil. Exemple de refactorizare includ: adaugarea unui sablon Façade la un pachet pentru reducerea gradului de cuplare, adaugarea unei superclase care reprezinta comportamentul comun al unei set de clase existente, inlocuirea unei instructiuni conditionale cu polimorfism, sau chiar modificarea numelui unei variabile (sau a unei functii) cu un nume mai sugestiv. Refactorizarea ar trebui sa pastreze codul simplu si mentenabil.
Integrare continua	De indata ce un increment este finalizat, este integrat in sistem. Dupa fiecare asemenea integrare, toate testele la nivel de modul ar trebui sa fie trecute cu succes.
Participare client	Un reprezentant al clientului ar trebui sa fie in mod continuu la dispozitia echipei XP. Intr-un proces XP clientul este membru al echipei de dezvoltatori, fiind direct responsabil cu stabilirea si ordonarea cerintelor care urmeaza sa fie implementate.
Ritm sustinut, dar fara excese	In abordarea XP un volum mare de ore suplimentare este considerat inacceptabil, deoarece efectul net consta in reducerea calitatii si a productivitatii pe termen mediu.

Programare extrema

- In XP, refactorizarea constanta este preferata abordarii traditionale in care se proiecteaza anticipandu-se posibile modificari ale cerintelor.
  - In ingineria software traditionala se considera ca merita sa se investeasca timp si efort de proiectare a unor solutii care anticipeaza posibile modificari ale cerintelor, deoarece aceasta abordare poate ulterior reduce costurile de dezvoltare.
  - In XP accentul este pe proiectarea cerintelor imediate, considerandu-se ca, de fapt, modificarile nu pot fi anticipate.
  - Pentru a facilita implementarea schimbarilor de cerinte la momentul la care acestea apar in mod efectiv, XP incurajeaza imbunatatirea constanta a codului prin refactorizare.

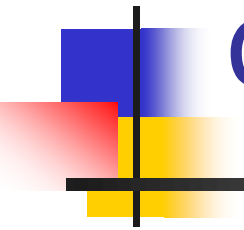


Programare extrema

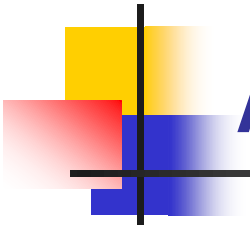
- **Dezvoltarea ghidata de testare** este una dintre cele mai importante inovatii ale XP [Som06,Som10].
- In [LL05] aceasta abordare este descrisa astfel:
  - In dezvoltarea ghidata de testare programatorul scrie mai intai un set de teste automate. Apoi, proiecteaza si se scrie codul astfel incat testele sa fie trecute. In esenta, testele reprezinta o reformulare a cerintelor intr-o forma automata. In abordarile flexibile bazate pe dezvoltare ghidata de testare, testele pot sa fie unica forma de reprezentare detaliata a cerintelor. … Prin proiectarea testelor automate se garanteaza faptul ca cerintele sunt bine intelese.

Programare extrema

- Activitatea in **perechi de programatori** reprezinta o alta practica inovativa a XP.
  - In XP, se lucreaza in perechi de (cate 2) programatori asezati in fata aceluiasi terminal. Perechile sunt formate in mod dinamic.
  - Aceasta practica
    - contribuie la o forma de proprietate comuna asupra codului si faciliteaza raspandirea cunostintelor in cadrul echipei XP,
    - serveste ca proces informal de recenzie a codului, fiecare linie de cod fiind vazuta de cel putin doua persoane si
    - incurajeaza refactorizarea in beneficiul intregii echipe XP.
  - Masuratorile efectuate sugereaza ca productivitatea unei perechi de programatori este similara cu cea a doi programatori care lucreaza independent.



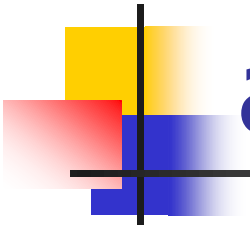
Activitati de baza in dezvoltarea software



Activitati de baza

- In acest capitol sunt discutate activitatile de baza in ingineria software:
  - specificare,
  - proiectare,
  - validare, si
  - evolutie.
- Prezentarea este adaptata in special dupa [JBR99,Pre97,Pre00,Som01,Som04,Som06, Som10].

Activitati de baza – analiza si specificare



- Activitatea de **analiza** are drept scop stabilirea cerintelor software (functionale si nonfunctionale).
- Cerintele rezultate din analiza trebuie sa fie documentate, iar aceasta activitate se numeste **specificare** [Som89]. Specificarea este un proces de reprezentare [Pre97].

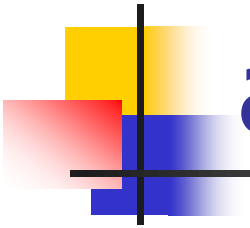
Activitati de baza – analiza si specificare

- **Principiile activitatilor de analiza si specificare** [Dav95] (cf. [Pre97]):
  1. Domeniul informational al problemei trebuie sa fie reprezentat si inteles.
  2. Functiile care vor fi implementate in sistemul software trebuie sa fie definite. Comportamentul sistemului software (descris ca o consecinta a evenimentelor externe) trebuie sa fie reprezentat.
  3. In procesul de analiza se creeaza un **model cognitiv** (iar nu un model de proiectare sau de implementare) care descrie sistemul asa cum este perceput de utilizatori.
  4. Specificarea trebuie sa tolereze incompletitudini putand fi completata atunci cand exista informatia necesara. Continutul si structura specificarii trebuie sa permita **schimbari**.

Activitati de baza – analiza si specificare

- Cerintele software pot fi impartite in doua mari categorii [Som89,Som10]:
  - **Functionale** Acestea sunt serviciile pe care utilizatorii le asteapta de la sistem. Cerintele functionale pot fi validate prin prototipizare.
  - **Nonfunctionale** Acestea stabilesc constrangerile sub care trebuie sa opereze sistemul, standardele pe care trebuie sa le satisfaca, etc.

Activitati de baza – analiza si specificare

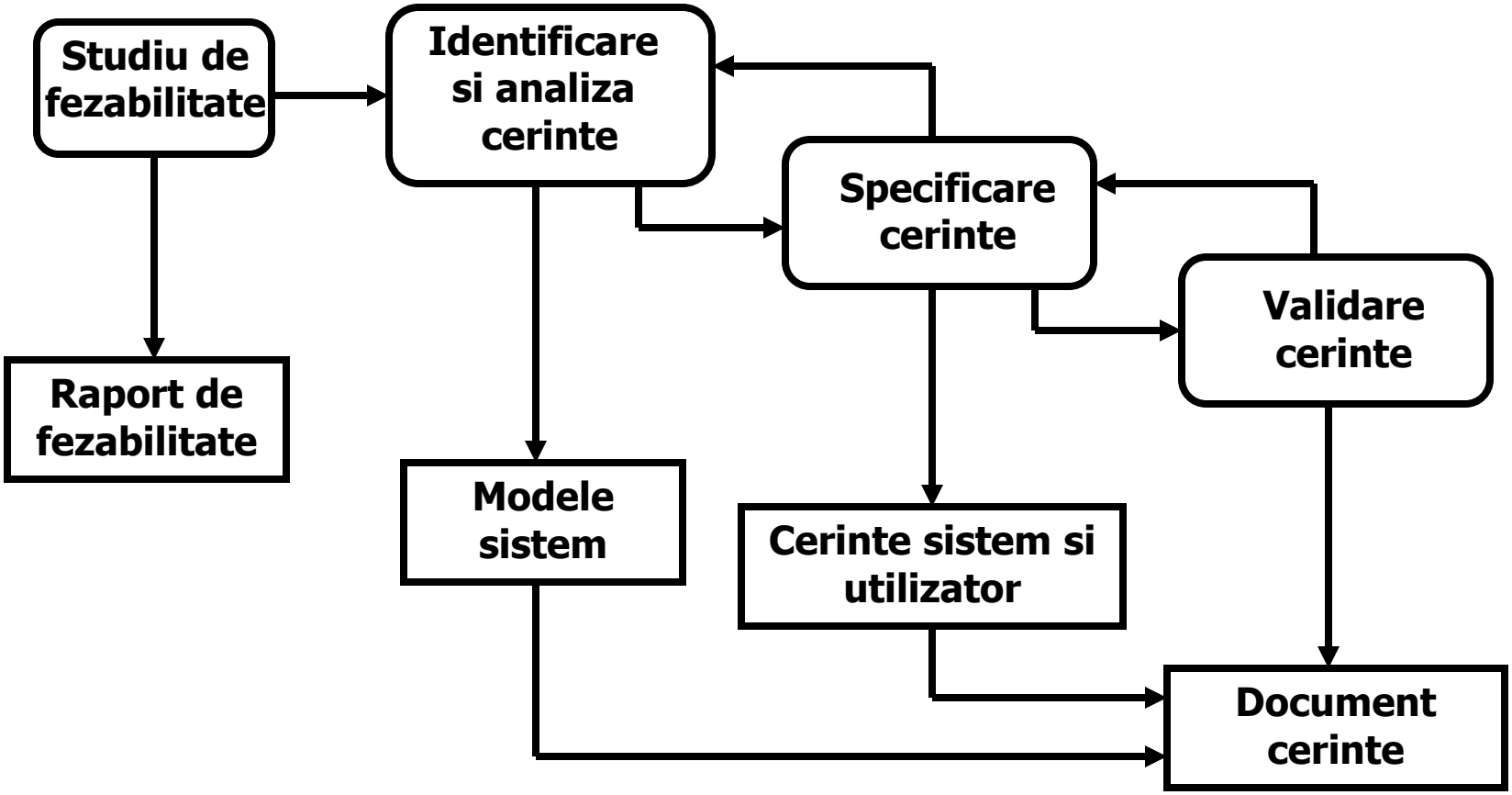


- Produsele software furnizeaza
  - **functionalitate** (servicii), si
  - **atribute nonfunctionale** care reflecta **calitatea**.
- **Atributele de calitate** (atributele nonfunctionale) reflecta [Som01,Som10]
  - Comportamentul produsului la executie
  - Structura si organizarea software
- Example de **atribute de calitate**: fiabilitate, eficienta, mentenabilitate (usurinta in intretinere), viteza (timpul de raspuns), siguranta, securitate, etc.

Activitati de baza – analiza si specificare

- Ingineria cerintelor se refera la toate activitatile legate de crearea si intretinerea unui **document de specificare a cerintelor**.
- In organizatii mari, structura documentului de specificare a cerintelor poate fi standardizata. IEEE propune propriul standard pentru asemenea documente [IEEE93b].
- Figura de mai jos [Som01,Som06] prezinta un **proces de ingineria cerintelor** (avand drept rezultat principal un document de specificare a cerintelor).

Activitati de baza – analiza si specificare



Activitati de baza – analiza si specificare

- **Faze ale procesului de inginerie a cerintelor [Som06]:**
  - **Studiu fezabilitate** Studiul decide daca sistemul propus poate fi economic in exploatare si daca poate fi dezvoltat utilizand tehnologia disponibila in limitele bugetului alocat proiectului. Studiul de fezabilitate ar trebui sa consume relativ putine resurse (inclusiv timp).
  - **Identificare si analiza cerinte** Acesta este procesul de captare a cerintelor sistem prin observarea comportamentului sistemelor existente, discutii cu clientul, analiza sarcini, etc. Poate implica dezvoltarea de modele sistem si prototipuri.
  - **Specificare cerinte** Cerintele colectate in faza de analiza sunt organizate in vederea alcatuirii documentului de specificare a cerintelor. Utilizatorii si clientii au nevoie de o descriere de nivel inalt a cerintelor; dezvoltatorii sistem opereaza cu o specificare sistem mai detaliata.
  - **Validare cerinte** Sunt studiate realismul, consistenta si completitudinea cerintelor obtinute. Se corecteaza defectele descoperite.

Activitati de baza – analiza si specificare

- Metode de identificare si analiza a cerintelor:
  - *Analiza structurata* [Mar79]
  - *Prototipizare*
  - *Scenarii* (**cazuri de utilizare** in OOSE [Jac92,JBR99])
  - *Analiza vederi* Aceasta abordare are la baza ideea ca un sistem poate fi descris prin vederi multiple, care permit structurarea cerintelor si organizarea procesului de analiza. CORE [Mul79] si VORD [KS96] sunt metode care utilizeaza analiza vederi.
    - Ex. – vederi mai importante pentru un sistem ATM: client, automat bancar, baza de date.
  - *Analiza si specificare formala (matematica)* (utilizate in subdomenii specializate).

Activitati de baza – proiectare

- Intrarea principala pentru activitatea de proiectare este documentul de specificare a cerintelor software.
- **O specificare este un model cognitiv**, bazat pe concepte din domeniul problemei (asa cum sunt percepute de comunitatea de utilizatori).
- **Un design este o schita de implementare** [JBR99], care utilizeaza concepte din domeniul solutiei tehnice.
- Deosebirea intre specificare si proiect (design) este uneori exprimata astfel:
  - **specificarea** spune **CE** anume trebuie implementat;
  - **designul** este o reprezentare a modului **CUM** se poate construi implementarea.

Activitati de baza – proiectare

- **Principii si ghid de proiectare** [Dav95] (cf. [Pre97]):
  1. Un bun proiectant ar trebui sa ia in considerare abordari (solutii tehnice) alternative, comparate pe baza cerintelor, a resurselor disponibile si a conceptelor de proiectare.
  2. Designul nu ar trebui sa reinventeze roata. Resursele sunt limitate. Timpul alocat proiectarii ar trebui sa fie investit in reprezentarea ideilor cu adevarat noi precum si in integrarea sabloanelor de design existente.
  3. Designul ar trebui sa fie uniform (stil unitar) si integrat (interfete de calitate).
  4. Designul ar trebui sa fie flexibil la schimbari.
  5. Pe baza designului ar trebui sa se poata obtine inapoi cerintele.
  6. Designul ar trebui sa trateze apectele functionale, comportamentale si structurale din perspectiva implementarii.
  7. Designul nu este identic cu implementarea. Nivelul de abstractizare a designului este mai inalt decat cel din codul sursa.

Activitati de baza – proiectare

8. Designul ar trebui sa minimizeze distanta intelectuala intre produsul software si problema, asa cum exista ea in lumea reala (structura designului ar trebui sa mimeze structura problemei).
9. Designul ar trebui sa fie alcatuit din module conectate (intre ele si cu mediul extern si utilizatorii) prin interfete simple.
10. Designul ar trebui sa cuprinda module software cu inalt grad de coeziune si grad redus de cuplare (intre module).

Activitati de baza – proiectare

- **Concepte**:
  - *Architectura* se refera la structura de ansamblu a sistemului software si la modul in care aceasta structura furnizeaza sistemului integritate semantica [SG96].
    - In unele abordari numai o parte dintre module sunt considerate a fi relevante pentru vederea arhitecturala a sistemului [JBR99].
    - Notiunea de *arhitectura* are la baza conceptul de *modularitate*.
  - *Modulul* reprezinta 'caramida' de baza in modularizarea sistemelor software. In general, un modul nu este considerat a fi un sistem independent.
  - Un *subsistem* este o parte a unui sistem care apare de sine statator intr-un proces.
    - Un (sub) sistem este alcatuit din module.

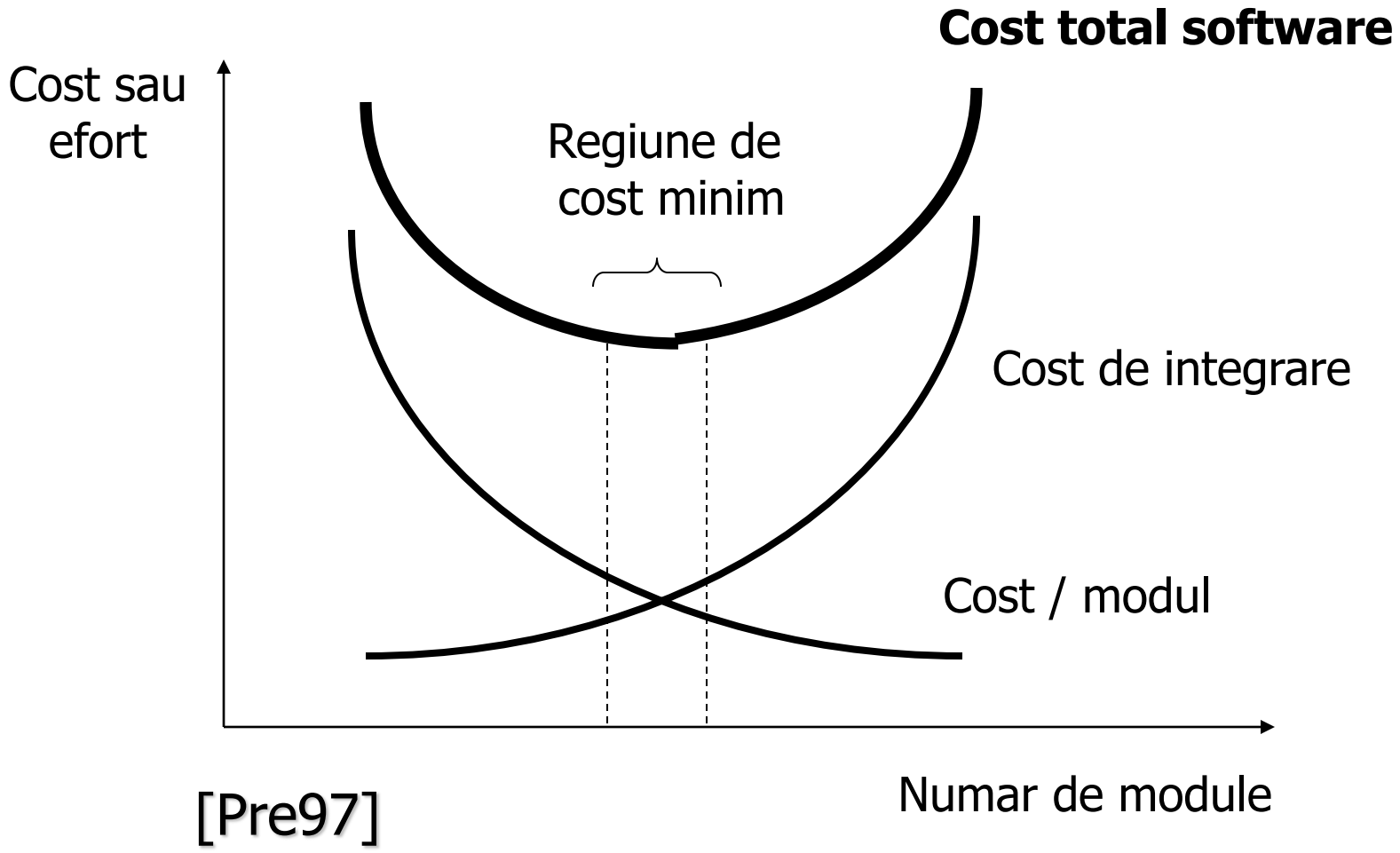
Activitati de baza – proiectare

- Un argument in favoarea modularizarii este dat de urmatoarea 'lege' (empirica) [Pre97]:

  E(p1 +…+ pn) > E(p1) +…+ E(pn)

  - $p_1,\ldots,p_n$ este o descompunere a unei probleme p in subprobleme.
  - $E(x)$ este efortul necesar rezolvarii problemei x
- In practica atat *submodularizarea* cat si *supramodularizarea* ar trebui evitate.

Activitati de baza – proiectare

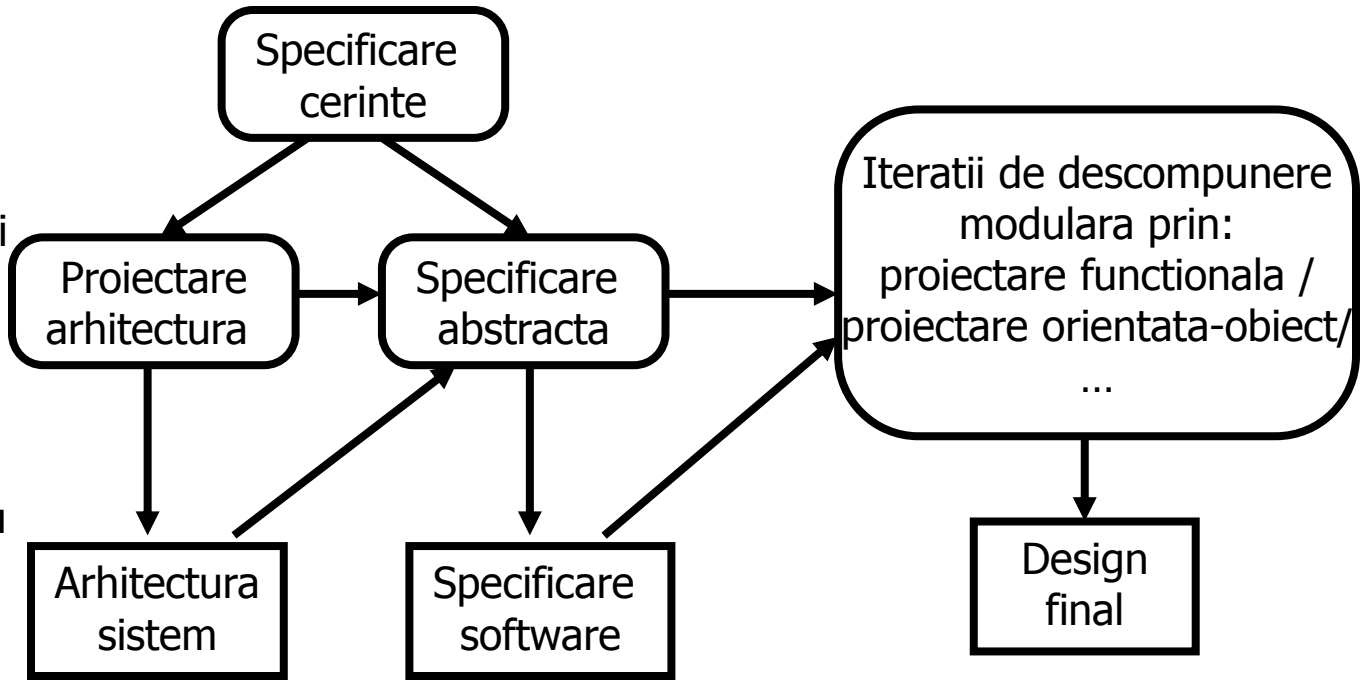


[Pre97]

Activitati de baza – proiectare

- Activitatea de proiectare necesita experienta si creativitate din partea inginerului software.
- Forma finala a designului este obtinuta prin iteratii din designuri preliminare.
- Activitatea de proiectare este un proces de formalizare si detaliere, in care pot fi necesare reveniri pentru corectarea unor designuri anterioare.

Un model foarte general al procesului de proiectare:



Activitati de baza – proiectare

- Fazele activitatii de proiectare [Som10,Som06,Som89]:
  - *Proiectare arhitectura* Sistemul este descompus intr-o structura de subsisteme. Rezultatul acestei activitati este o descriere a *arhitecturii sistem*.
  - *Specificare abstracta* Pentru fiecare subsistem, se realizeaza specificarea abstracta a serviciilor si constrangerilor sub care trebuie sa opereze.
  - *Descompunere modulara* In aceasta faza, se utilizeaza o metoda potrivita de proiectare (proiectare functionala / proiectare orientata obiect) si dupa un numar de iteratii se produce un design final.

Activitati de baza – proiectare

- Pentru descompunerea in module a subsistemelor vom considera:
  - *Modele data-flow* (proiectare functionala)
  - *Modele orientate obiect* (proiectarea OO)
- Strategii de baza in descompunerea modulara:
  - ***Proiectare top-down*** Aceasta strategie se bazeaza pe ideea ca structura problemei ar trebui sa determine structura solutiei software.
  - ***Proiectare bottom-up*** In practica, strategia top-down este combinata cu strategia bottom-up (proiectantii gasesc metode de reutilizare pe masura ce rafinarea top-down progreseaza).

Activitati de baza – testare

- In activitatea de **testare** scopul inginerului software este de a verifica si valida (V & V) implementarea.
  - Toate activitatile de dezvoltare – de la specificare la elaborare cod – implica activitati V & V, cum ar fi inspectii si recenzii.
  - Totusi, cea mai importanta activitate V & V este testarea software, care este realizata dupa implementare.
    - **Verificarea** stabileste daca sistemul se conformeaza cerintelor (specificarii).
    - **Validarea** stabileste daca produsul software satisface asteptarile clientului.
    - In limba engleza se utilizeaza urmatorul joc de cuvinte [Boe79]:
      - Verification: are we building the product right?
      - Validation: are we building the right product?

Activitati de baza – testare

- Activitatea de testare este o mare consumatoare de resurse:
  - 30%-40% in proiecte ‘obisnuite’;
  - pana la de 5 ori resursele tuturor celorlalte activitati de dezvoltare in proiecte cu cerinte stringente in ceea ce priveste fiabilitatea, securitatea, etc. (cf. [Pre97]).
- In proiecte dezvoltate prin metode formale testarea la nivel de modul poate fi inlocuita cu o verificare formala a specificatiei matematice. Aceast proces are loc inainte de implementare.
  - Dar metodele formale sunt utilizate numai in domenii specializate.
- In majoritatea proiectelor, activitatea V & V care consuma cele mai multe resurse se desfasoara dupa implementare, atunci cand se realizeaza testarea sistemului.

Activitati de baza – testare

- Doua abordari complementare:
  - *Testare functionala sau black-box*
    - In aceasta abordare se testeaza comportamentul functional al produsului software (relatia intrari-iesiri).
    - Testarea functionala se bazeaza pe cerinte. Nu se utilizeaza informatie legata de structura interna a sistemului software.
  - *Testare white-box (sau glass-box)*
    - Este o modalitate de a testa structura sistemului software utilizand cunostinte despre structura sa interna.
    - Se utilizeaza informatii din fazele de proiectare si implementare.

Activitati de baza – testare

- Obiective ale testarii [Mye79]:
  - Testarea este procesul de executare a unui program cu scopul de a detecta defecte.
  - Un caz de test bun este acela care are o mare probabilitate de a detecta un defect care nu a fost descoperit pana in acel moment.
  - Un test este reusit daca descopera un defect care nu a fost descoperita inainte.
- Se subliniaza insa ca [Dij72]
  - Testarea nu poate detecta absenta defectelor, ci doar prezenta defectelor intr-un sistem software.

Activitati de baza – testare

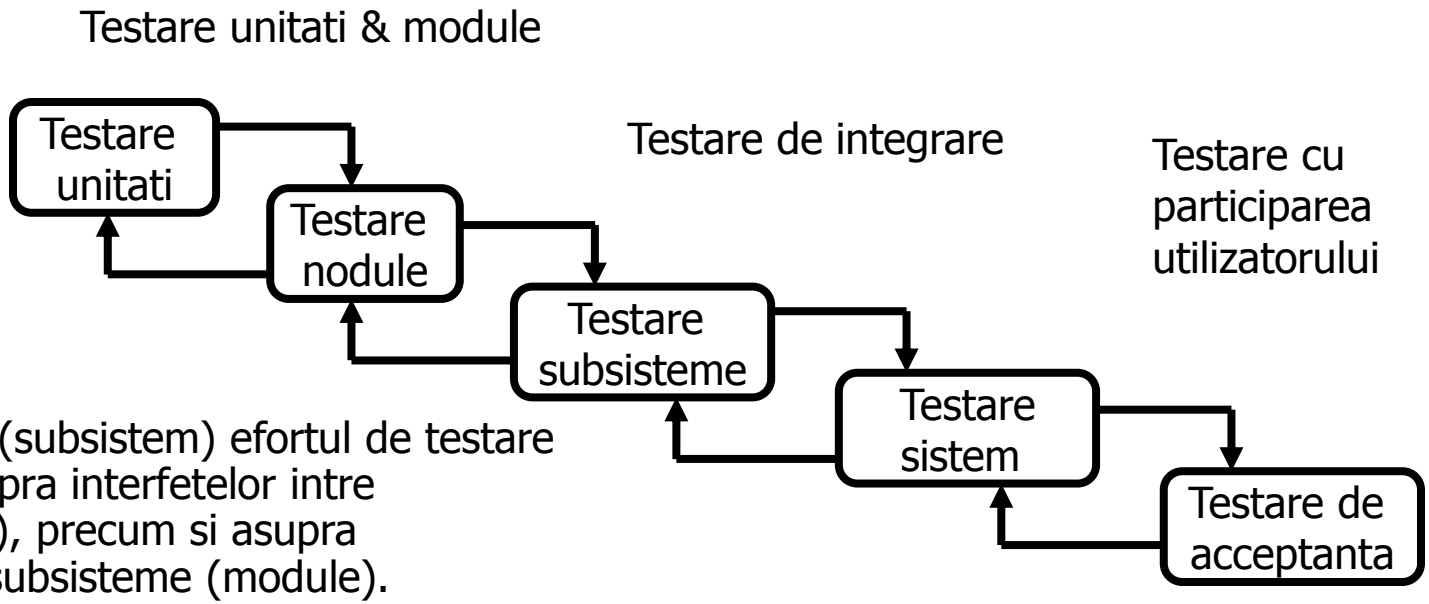
- **Principiile activitatii de testare** [Dav95] (cf. [Pre97]):
  1. Testele trebuie sa fie ghidate de cerinte, deoarece cele mai severe defecte sunt cele care conduc la nesatisfacerea cerintelor.
  2. Testele ar trebui sa fie planificate (mult) inainte de inceperea activitatii de testare. Testele black-box pot fi planificate pe baza cerintelor. Testele de tip white-box pot fi planificate pe baza modelului de design.
  3. Testarea trebuie realizata de la simplu (testare module) spre complex (testare de integrare si testare de acceptanta).
  4. Testarea exhaustiva nu este posibila.
  5. Pentru a fi eficace, testarea ar trebui sa fie condusa de un grup independent (care nu este subordonat dezvoltatorului, ci organizatiei care se ocupa cu asigurarea calitatii).

Activitati de baza – testare

- Testarea software se realizeaza in urmatoarele stadii:
  - testare unitati program (proceduri, functii),
  - testare la nivel de modul (clasa, ADT, modul fisier),
  - testare la nivel de subsistem,
  - testare la nivel de sistem,
  - testare de acceptanta (pe date furnizate de client, numita uneori *testare alfa*).
  - (Poate urma o faza de *testare beta*, in care sistemul este livrat spre utilizare si testare catre un numar de clienti si utilizatori potentiali).

Activitati de baza – testare

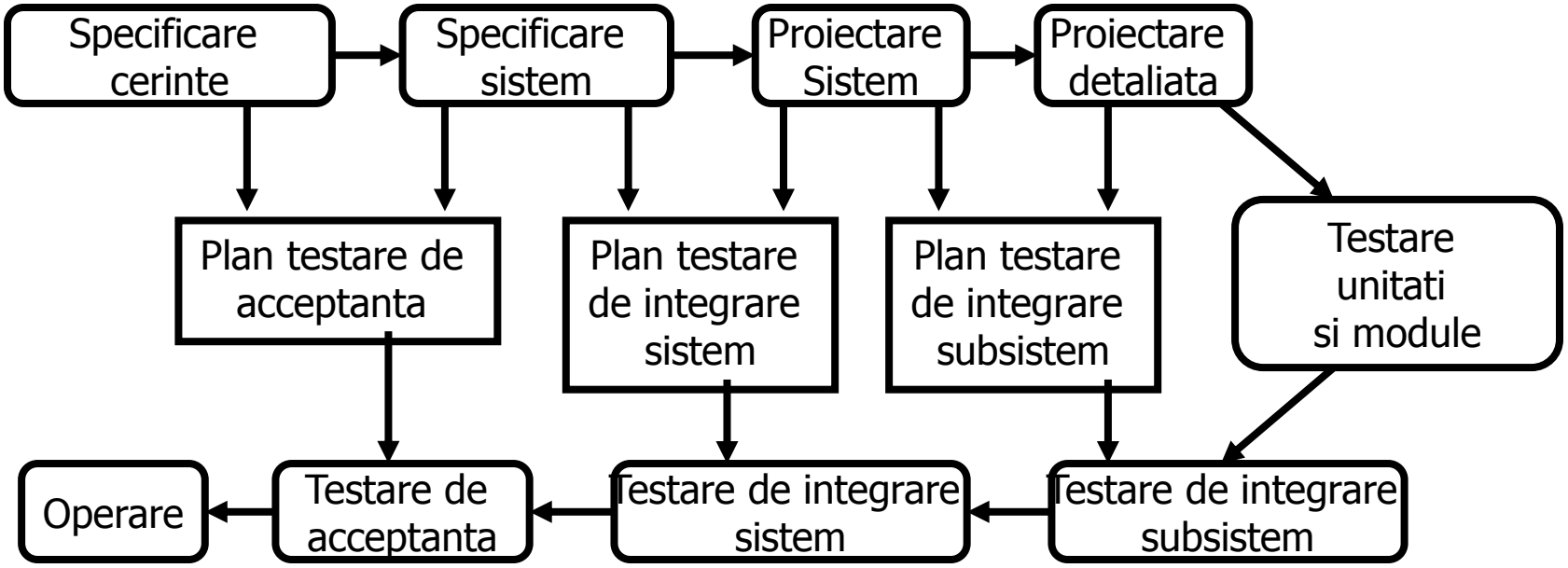
Procesul de testare:



- La nivel de sistem (subsistem) efortul de testare se concentreaza asupra interfetelor intre subsisteme (module), precum si asupra interactiunilor intre subsisteme (module).
- In faza de testare de acceptanta sistemul este testat cu date furnizate de client (iar nu cu date de test simulate). In acest stadiu se pot valida atat cerinte functionale cat si cerinte nonfunctionale.

Activitati de baza – testare

- Figura de mai jos prezinta relatia intre activitatile de dezvoltare si cele de testare prin planuri de testare.



Activitati de baza – evolutie software

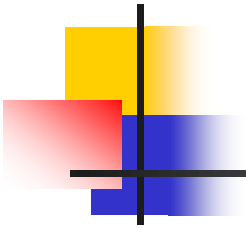
- Majoritatea sistemelor software trebuie sa evolueze (apar cerinte noi iar cerintele mai vechi se pot modifica).
  - Demarcatia care se realiza in mod traditional intre procesele de dezvoltare si evolutie devine tot mai putin relevanta [Som10].
    - Au aparut noi tehnologii, in particular inginerie software orientata pe reutilizare, si
    - In prezent putine sisteme software sunt sisteme complet noi.
  - Costurile legate de modificarea (evolutia) produselor software pot fi mai ridicate decat costurile de dezvoltare. Tehnologia software ofera insa beneficii semnificative [Som04].
    - Sistemele software sunt mai flexibile decat cele hardware (si decat alte sisteme ingineresti).
    - Costurile de modificare sunt mai reduse in cazul produsele software decat in cazul unor produse hardware corespunzatoare.
    - Produsele ingineresti incorporeaza o cantitate tot mai mare de componente software.

Activitati de baza – evolutie software

- Strategii de evolutie software [Som01,Som04,Som10]:
  - *Intretinere software* (engl. *software maintenance*):
    - corectare defecte,
    - adaptare software la noi medii de operare,
    - modificare sau adaugare functionalitate.
  - *Evolutie arhitecturala* (de exemplu, transformarea unei arhitecturi centralizate intr-o arhitectura distribuita client-server pentru un produs software mai vechi)

Activitati de baza – evolutie software

- *Reinginerie software* (engl. *re-engineering*)
  - Sistemul software este modificat cu scopul de a deveni mai usor de inteles si modificat.
  - Nu se adauga functionalitate noua (in mod normal nici arhitectura nu se modifica).
  - Modificarile pot include:
    - redocumentarea sistemului,
    - reorganizarea si restructurarea sistemului,
    - translatarea implementarii intr-un limbaj de programare nou,
  - In acest proces este important (sau chiar esential) sa se utilizeze instrumente CASE, care pot furniza suport pentru: translatare automata cod sursa, inginerie directa (forward engineering), inginerie inversa (reverse engineering), etc.



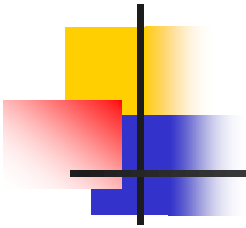
Bibliografie

[Bec99] K. Beck. Embracing Change with Extreme Programming. *IEEE Computer* 32(10):70-78, 1999.

[Boe79] B. Boehm. Software engineering; R & D trends and defense needs. In *Research Directions in Software Technology*, pages 1-9, MIT Press, 1979.

[BRJ99] G. Booch, J. Rumbaugh and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[Coc01] A. Cockburn. *Agile software development*. Addison-Wesley, 2001.



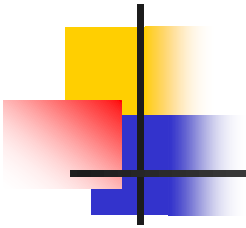
Bibliografie

[Dav95] A. Davis. *201 Principles of Software Development*. McGraw-Hill, 1995.

[Dij72] E. Dijkstra, et al. *Structured programming*. London: Academic Press, 1972.

[FP13] A. Fox, D. Patterson. *Engineering Software as a Service: An Agile Approach using Cloud Computing*. Strawberry Canyon Publisher, 2013.

[IEEE93b] IEEE recommended practice for software requirements specifications. *Software Engineering Requirements Engineering*, R.H. Thayer, M. Dorfman, editors, Los Alamitos, CA: IEEE Computer Society Press.



Bibliografie

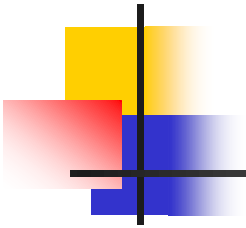
[Jac92] I. Jacobson. *Object-Oriented Software Engineering*. Addison-Wesley, 1992.

[JBR99] I. Jacobson, G. Booch and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[KS96] G. Katonya, I. Sommerville. Requirements engineering with viewpoints. *BCS/IEEE Software Engineering J.*, 11(1):5-18, 1996.

[Kru03] P. Kruchten. *The Rational Unified Process: An Introduction* (3rd edition). Addison-Wesley, 2003.

[LL05] T. Lethbridge and R. Laganiere. *Object-Oriented Software Engineering* (2nd edition). McGraw-Hill, 2005.



Bibliografie

[Mar79] T. DeMarco. *Structured Analysis and System Specification*. Prentice-Hall, 1979.

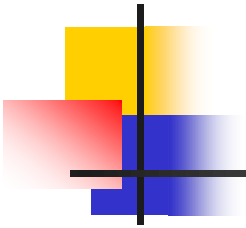
[Mul79] G. Mullery. CORE – a method for controlled requirements specification. In *Proc. 4th Int. Conf. On Software Engineering*, IEEE Press, 1979.

[Mye79] G. Myers. *The Art of Software Testing*. Wiley, 1979.

[Pre97] R. Pressman. *Software Engineering: A Practitioner's Approach*, (4th edition). McGraw-Hill, 1997.

[Pre00] R. Pressman. *Software Engineering: A Practitioner's Approach*, (5th edition). McGraw-Hill, 2000.

[Pre14] R. Pressman. *Software Engineering: A Practitioner's Approach*, (8th edition). McGraw-Hill, 2014.



Bibliografie

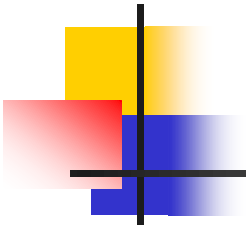
[SB01] K. Schwaber, M. Beedle. *Agile software development with SCRUM*. Prentice Hall, 2001.

[SG96] M. Shaw and D. Garlan. *Software Architecture*. Prentice-Hall, 1996.

[Som89] I. Sommerville. *Software Engineering,* (3rd edition). Addison-Wesley, 1989.

[Som01] I. Sommerville. *Software Engineering,* (6th edition). Addison-Wesley, 2001.

[Som04] I. Sommerville. *Software Engineering,* (7th edition). Addison-Wesley, 2004.



Bibliografie

[Som06] I. Sommerville. *Software Engineering,* (8th edition). Addison-Wesley, 2006.

[Som10] I. Sommerville. *Software Engineering,* (9th edition). Addison-Wesley, 2010.

[Som15] I. Sommerville. *Software Engineering,* (10th edition). Addison-Wesley, 2015.

[Sta97] J. Stapleton. *DSDM Dynamic Systems Development Method*. Addison-Wesley, 1997.