Fundamente de Inginerie Software

Cap. 6

Dezvoltarea Sistemelor Software.

Conf.Dr.Ing. Dan Pescaru

Textbooks: Sommerville "Software Engineering 7", 2004, Cap. 17, 18

Sursă: http://www.comp.lancs.ac.uk/computing/resources/IanS/SE7/

2009

Dezvoltarea Rapidă a Aplicaţiilor (RAD)

- Datorită schimbărilor rapide în mediul de afaceri, companiile trebuie să răspundă noilor oportunităţi şi competiţiei
- Aceasta necesită dezvoltarea şi livrarea rapidă a sistemelor software adecvate, poate cea mai critică cerinţă pentru aceste sisteme
- Afacerile pot accepta software de calitate chiar mai scăzută dacă este posibilă astfel livrarea rapidă a funcţionalităţii esenţiale

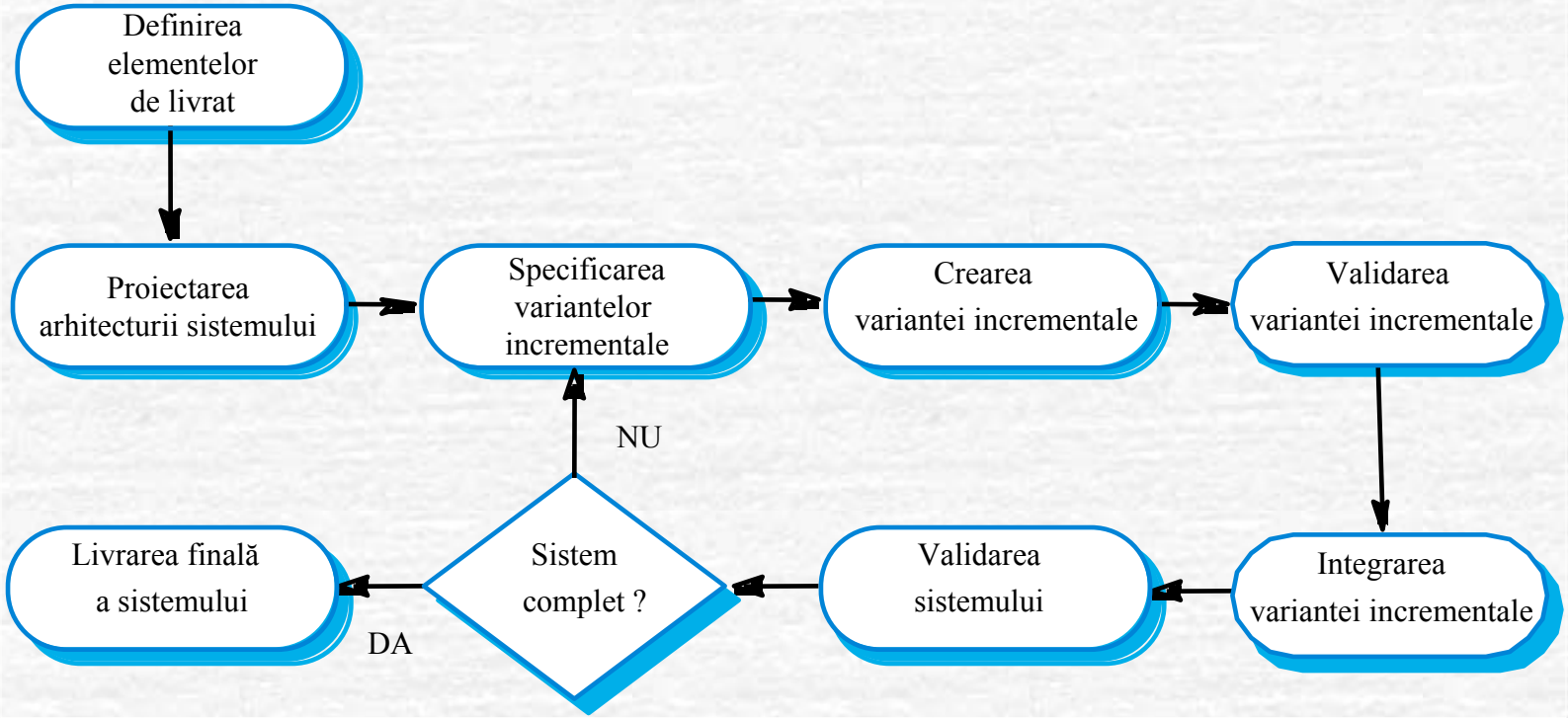
Cerinţe

- Ajungerea la un set de cerinţe de sistem consistente şi stabile este deseori imposibilă din cauza mediului în schimbare
- Ca atare un model de dezvoltare în cascadă este nepractic în această situaţie
- Cel mai potrivit este un model de dezvoltare rapidă bazat pe specificaţii şi livrare iterative

Caracteristici ale unui proces RAD

- Procesul de specificare, proiectare şi implementare este un proces concurent. Nu există o specificaţie detaliată şi documentarea proiectării este minimizată
- Sistemul este dezvoltat într-o serie incrementală. Utilizatorii finali evaluează fiecare variantă incrementală şi fac propuneri pentru următoarea variantă
- Interfeţele utilizator ale sistemului sunt dezvoltate de obicei într-un proces interactiv

Proces de dezvoltare iterativ



Avantajele dezvoltării incrementale

- Livrarea accelerată de servicii clienţilor. Fiecare variantă incrementală livrează cea mai prioritară funcţionalitate către client

- Implicarea utilizatorului vizavi de sistem. Utilizatorul trebuie să se implice în dezvoltare ceea ce duce la un sistem care se mulează mai bine pe cerinţele sale

Problemele dezvoltării incrementale

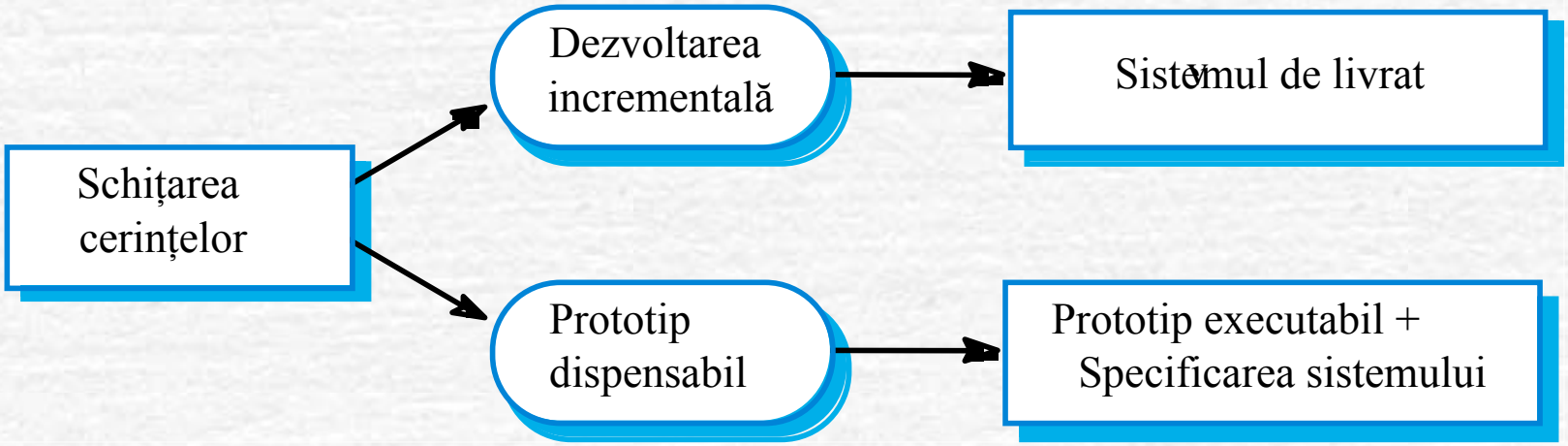
- Probleme de management.
  - Greu de judecat progresul dezvoltării. Problemele sunt greu de descoperit deoarece nu există documentaţie care să evidenţieze exact ce s-a realizat
- Probleme contractuale
  - Un contract obişnuit include specificaţii. În cazul acesta forma contractului nu va putea să includă specificaţii (greu de estimat un buget)
- Probleme la validare
  - În lipsa specificaţiilor, cum se testează sistemul?
- Probleme de mentenanţă
  - Schimbările continue tind să corupă structura sistemului ceea ce duce la costuri crescute la schimbări şi evoluţii necesare pentru satisfacerea noilor cerinţe

Prototipizarea

- În cazul unor sisteme mari, dezvoltarea iterativă incrementală poate fi nepractică, în special când mai multe echipe lucrează în locaţii diferite

- Prototipizarea se referă la dezvoltarea unui sistem experimental ce poate fi utilizat ca bază pentru formularea cerinţelor. După ce se ajunge la o înţelegere, se poate renunţa la acest sistem

Dezvoltarea Incrementală şi Prototipizarea



Obiective conflictuale

- Obiectivul dezvoltării incrementale este livrarea unui sistem funcţional utilizatorilor finali. Dezvoltarea începe cu acele cerinţe ce sunt cele mai bine înţelese

- Obiectivul prototipului dispensabil este acela de a valida sau furniza cerinţele sistemului. Procesul de prototipizare începe cu acele cerinţe care sunt mai neclare

Metode “Agile”

- Metodele “Agile” au apărut datorită modului greoi caracteristic metodelor clasice de proiectare. Acestea:
  - Se concentrează pe cod nu pe proiectare
  - Se bazează pe o dezvoltare iterativă
  - Au ca principal scop acela de a livra rapid software funcţional adaptat cerinţelor în schimbare
- Metodele “Agile” sunt probabil cel mai potrivite sistemelor de dimensiuni mici/medii

Principiile Metodelor “Agile”

- Implicarea clientului
  - Clientul trebuie implicat activ în procesul de dezvoltare. Rolul sau este acela de a furniza şi prioritiza cerinţele, respectiv de a evalua iteraţiile sistemului
- Livrare incrementală
  - Software-ul este livrat în variante incrementale
- Accent pe oameni nu pe proces
  - Abilităţile echipei de dezvoltare trebuie recunoscute şi exploatate. Echipa trebuie lăsată să-şi stabilească propria cale de a lucra la proiect
- Includerea schimbărilor
  - Sistemul trebuie proiectat astfel încât să permită schimbarea cerinţelor
- Menţinerea simplităţii
  - Simplitatea trebuie urmărită atât în cazul produsului cât şi al procesului de dezvoltare

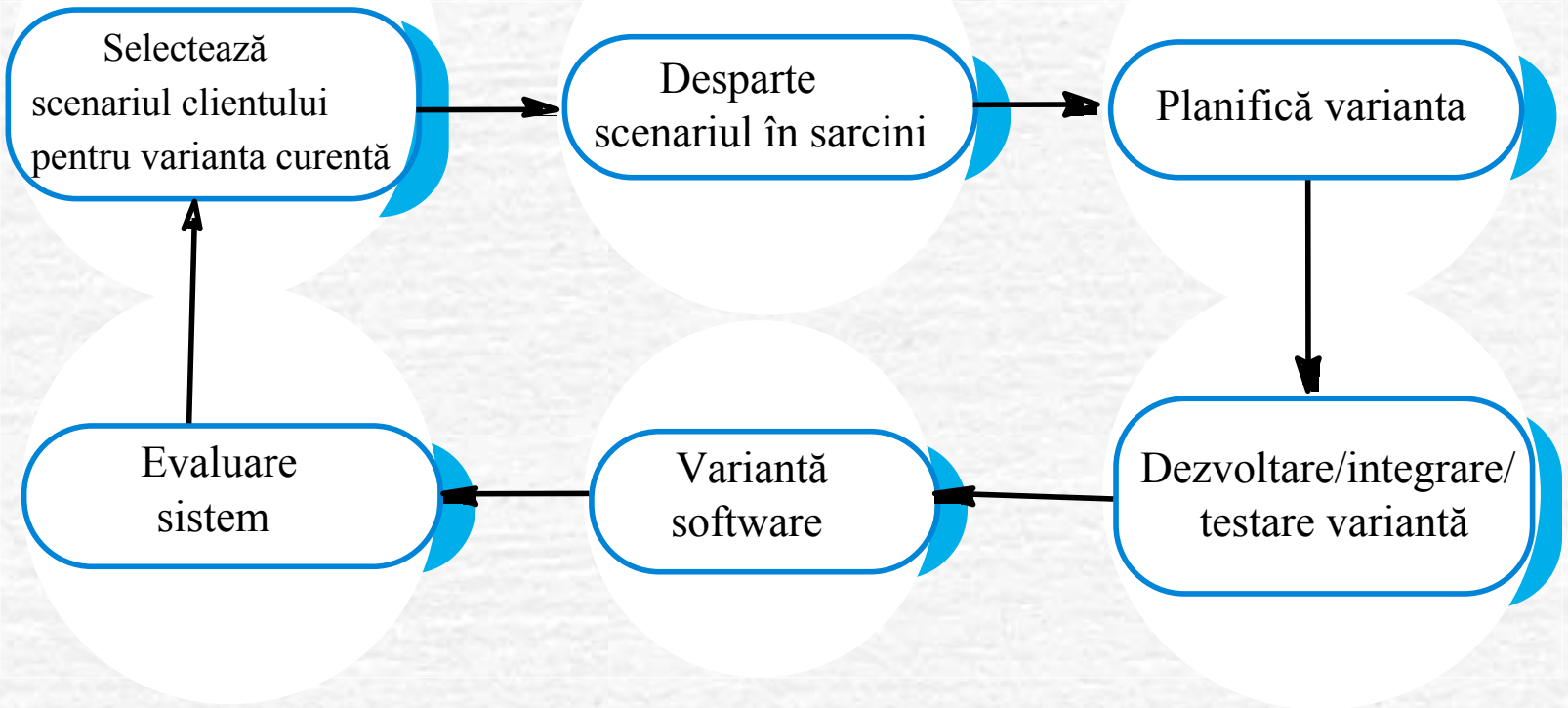
Problemele Metodelor “Agile”

- Interesul clienţilor implicaţi în proces poate fi dificil de păstrat
- Membrii echipei pot să aibă dificultăţi de adaptare la implicarea intensă care caracterizează metodele “Agile”
- Prioritizarea schimbărilor poate fi dificilă când există parteneri multiplii
- Menţinerea simplităţii poate implica muncă în plus
- Contractele pot fi o problemă la fel ca şi in cazul celorlalte abordări de dezvoltare iterativă

Extrem Programming

- Probabil cea mai cunoscută şi larg utilizată metodă "Agile"
- Extrem Programming (XP) aduce un concept 'extrem' dezvoltării iterative
  - Noi versiuni sunt dezvoltate chiar de câteva ori pe zi
  - Variante incrementale sunt livrate clienţilor la fiecare 2 săptămâni
  - Toate testările trebuie făcute pentru fiecare variantă, aceasta fiind acceptată doar dacă testele rulează cu succes

Ciclurile de Dezvoltare XP



Practici Extrem Programming (1)

- Planificarea incrementală
  - Cerinţele sunt înregistrate pe "Story Cards" şi determinarea scenariului care să fie inclus într-o variantă funcţie de timpul avut la dispoziţie şi de priorităţi. Dezvoltatorii despart aceste scenarii în sarcini
- Variante mici
  - Setul minimal de funcţionalităţi care esenţiale pentru afacere este dezvoltat primul. Variantele sistemului sunt frecvente şi adaugă funcţionalităţi în mod incremental
- Proiectare simplă
  - Proiectarea se limitează la suportul cerinţelor curente
- Testare de la început
  - O unitate cadru automată de testare va fi utilizată pentru scrierea testelor pentru fiecare funcţionalitate nouă, înainte de implementarea acesteia

Practici Extrem Programming (2)

- Refactorizare
  - Codul va fi refactorizat continuu când se întrevăd posibile îmbunătăţiri. În acest fel codul va fi menţinut simplu şi uşor de întreţinut
- Programare în perechi
  - Îşi vor verifica reciproc munca şi se ajută pentru a obţine rezultate cât mai bune
- Proprietate comună
  - Proprietatea asupra codului este comună celor doi. Altcineva nu poate interveni pentru a schimba codul
- Integrarea continuă
  - Imediat ce o sarcină este completată rezultatul este integrat în sistem. După integrare trebuie trecute toate unităţile de test ale sistemului

Principii XP şi Agile

- Dezvoltarea incrementală este sprijinită prin variante ale sistemului mici şi frecvente
- Implicarea clientului înseamnă munca full-time a acestuia în cadrul echipei
- Programarea va fi axată pe oameni nu pe procesul de producţie, codul este deţinut în comun şi vor fi evitate orele suplimentare
- Schimbările vor fi înglobate "din mers" prin dezvoltarea de variante regulate ale sistemului
- Simplitatea va fi menţinută prin refactorizarea continuă a codului

Scenariile Cerinţelor

- În XP, cerinţele utilizatorilor sunt exprimate ca şi scenarii sau relatări utilizator
- Acestea sunt scrise pe cartele şi echipa de dezvoltare le desparte apoi în sarcini de implementat. Aceste sarcini sunt baza planificării şi a estimării costurilor
- Clienţii aleg scenariile care vor fi incluse în următoarea variantă pe baza priorităţilor de terminate de ei şi a panificării

Exemplu de “Story Card”

- Ex: aducerea din reţea a documentelor

Aducerea şi tipărirea unui articol
La început vei selecta articolul dorit din lista afişată. Apoi trebuie selectat cum vei plăti pentru el – printr-un abonament, printr-un cont al companiei sau printr-o carte de credit. După aceasta vei primi un formular de copyright pentru a-l completa şi, după ce îl trimiți documentul va fi salvat pe calculatorul tău. Apoi vei alege o imprimantă şi documentul va fi tipărit. Vei confirma apoi sistemului că documentul a fost corect tipărit. Dacă articolul este print-only nu poți păstra o versiune PDF şi ca atare documentul va fi şters automat din calculator.

XP şi Schimbările Cerinţelor

- Sfatul convenţional în ingineria software este să proiectezi pentru schimbare. Este util de implicat timp şi efort pentru anticiparea schimbărilor pentru că aşa se reduc costurile ulterioare pe durata ciclului de viaţă
- Cu toate acestea, XP susţine că nu este un lucru prea util cât timp schimbările nu pot fi anticipate exact
- Propune în schimb îmbunătăţirea constantă a codului (refactorizare) pentru a face schimbările mai uşoare când vor trebui implementate

Testarea în XP

- Dezvoltarea “test-first”
- Teste dezvoltate incremental din scenarii
- Implicarea utilizatorului în testare şi validare
- Teste automate utilizate pentru a gestiona testarea tuturor componentelor de fiecare dată când o nouă variantă este creată

Exemplu de “Task Cards”

Task 1: Implementarea fluxului principal

Task 2: Implementarea catalogului de articole şi a selecţiei

Task 3: Implementarea plăţii

Plata poate şi făcută în trei moduri diferite. Utilizatorul selectează în ce mod doreşte să plătească. Dacă utilizatorul are un abonament la bibliotecă poate introduce numărul de abonament care va fi verificat de sistem. Ca şi alternativă el poate introduce un număr de cont al companiei.
Dacă acesta este valid se va înregistra plata în respectivul cont.
A treia alternativă este introducerea unui număr de carte de credit şi a datei de expirare a acesteia, urmată de verificare şi înregistrarea plăţii în contul asociat.

Descrierea unui caz de test

Test 4: Testarea validității cărții de credit
Intrare: Un şir reprezentând numărul cărții de credit şi doi întregi reprezentând luna şi anul expirării ei. **Teste:** Verifică dacă toate caracterele din şir sunt cifre. Verifică dacă luna este în intervalul 1-12 şi anul este mai mare sau egal cu anul curent. Utilizând primele patru cifre ale cărții de credit se verifică dacă entitatea emitentă este validă prin verificarea tabelei cu entitățile valide. Verifică validitatea cărții de credit prin trimiterea informațiilor despre ea unității emitente. **Ieşire:** OK sau mesaj de eroare indicând invaliditatea cărții de credit.

Dezvoltare “Test-first”

- Teste sunt scrise înainte de implementarea în cod a cerinţelor
- Testele sunt scrise ca şi programe (nu ca seturi de date de test) astfel încât să poată fi executate automat. Acestea vor furniza automat date de intrare către program şi vor testa rezultatele obţinute
- Toate testele noi şi existente sunt rulate automat când sunt adăugate noi funcţionalităţi. În acest fel se testează că noile funcţionalităţi nu au introdus erori

Programarea în perechi

- În XP programatorii lucrează în perechi, dezvoltând codul împreună
- Acest fapt ajută răspândirea cunoştinţelor în cadrul echipei şi responsabilitatea comună asupra codului
- Serveşte pe post de proces de recenzie implicit deoarece fiecare linie de cod este privită de mai mult de o persoană
- Încurajează refactorizarea cât timp întreaga echipă beneficiază de aceasta
- Evaluările sugerează că productivitatea programării în perechi este similară cu cea a programării individuale pentru cei doi programatori

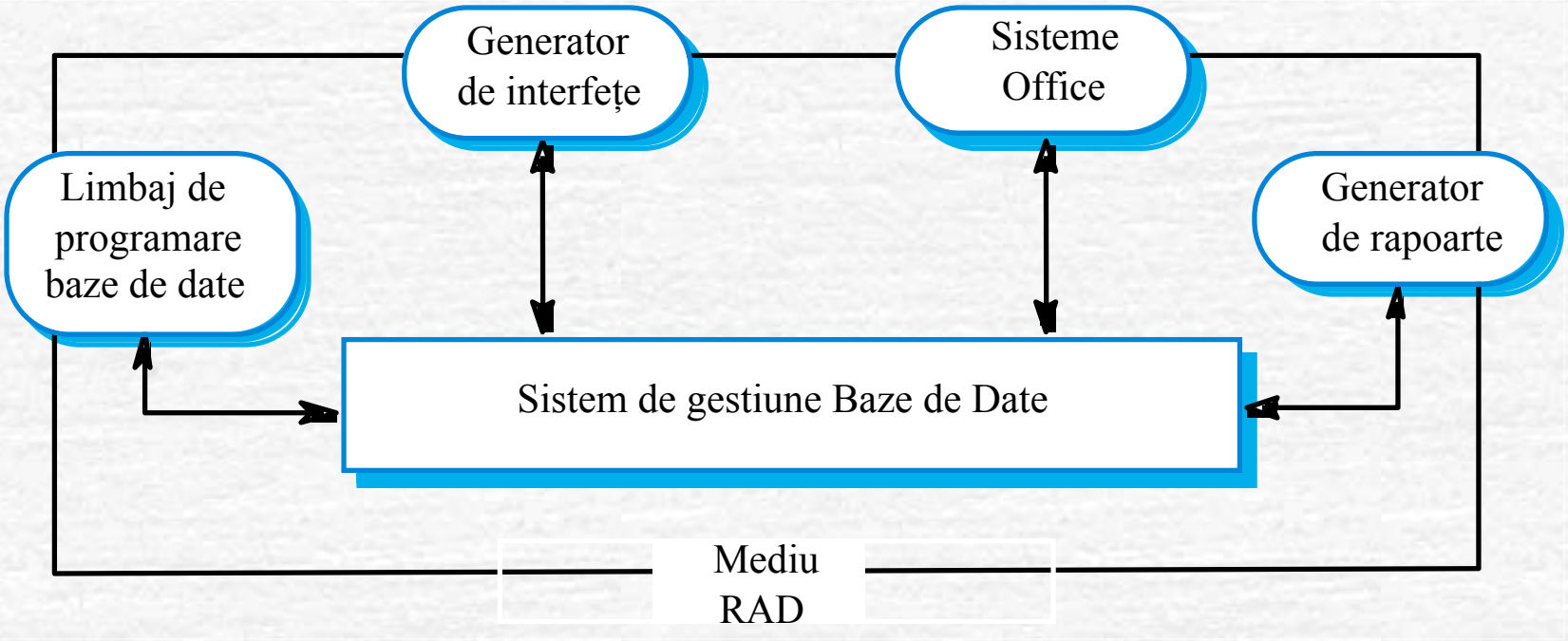
RAD

- RAD = Rapid Application Development
- Metodele Agile sunt în centrul atenţiei dar şi alte tehnici RAD au fost aplicate cu succes de mai mulţi ani
- Acestea sunt proiectate pentru aplicaţii de afaceri care prelucrează multe date şi constau în procesarea şi prezentarea informaţiilor dintr-o bază de date

Componentele unui Mediu RAD

- Limbaj de programare pentru baze de date
- Generator de interfeţe
- Legături cu aplicaţii de birou (editoare de text, foi de calcul, editoare de grafice etc.)
- Generator de rapoarte

Componentele unui Mediu RAD



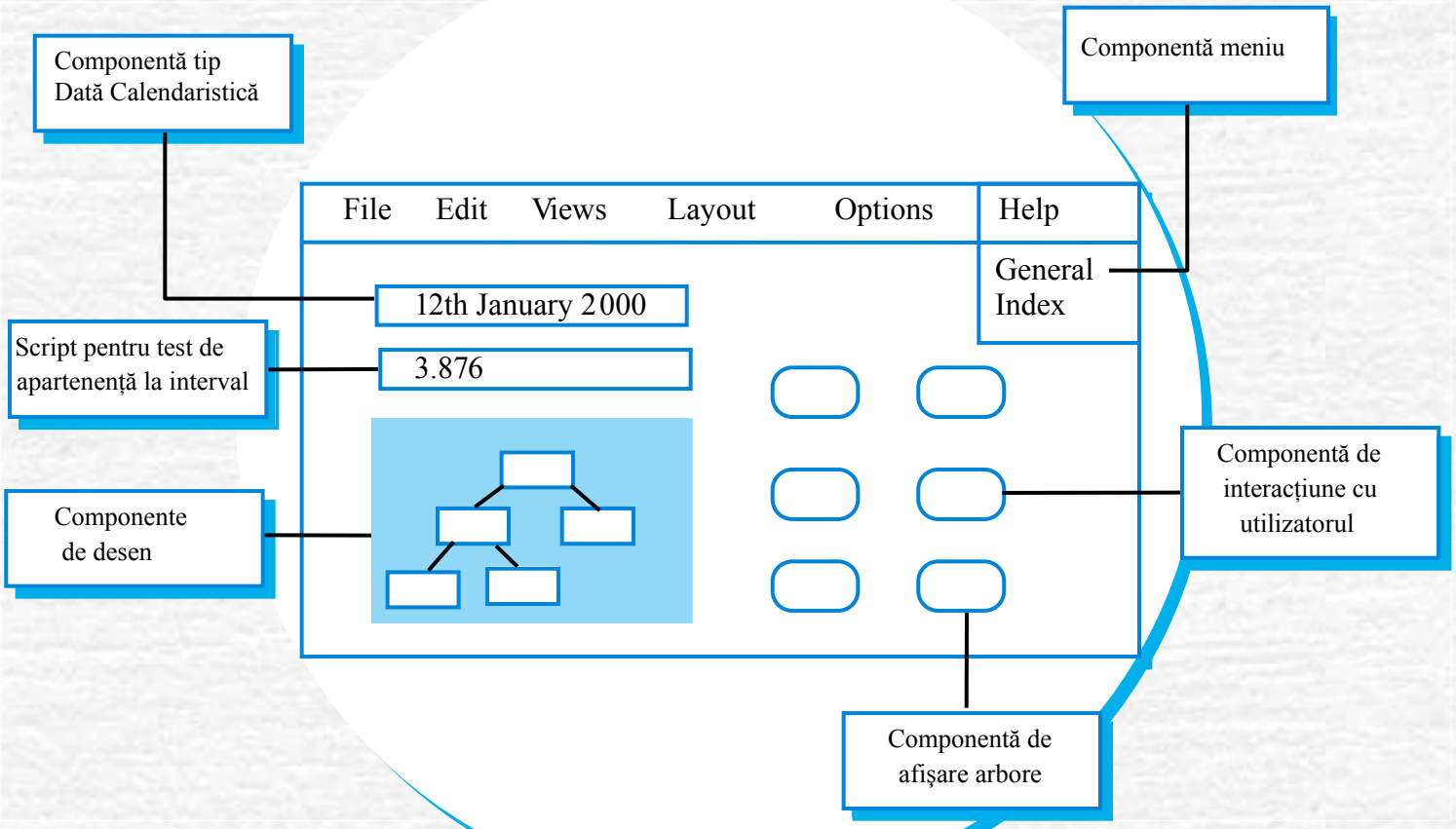
Generarea Interfeţelor

- Multe aplicaţii se bazează pe formulare de intrare complexe. Dezvoltarea unor astfel de formulare manual este o activitate mare consumatoare de timp
- Mediile RAD includ suport pentru generarea de ecrane de intrare, cuprinzând:
  - Definirea interactivă (cu mouse-ul) utilizând tehnici de drag-and-drop
  - Legarea formularelor pentru o secvenţă de formulare specificată
  - Verificarea formularelor pentru încadrarea câmpurilor în limite predefinite

Programarea Vizuală

- Limbaje de programare tip script, de exemplu Visual Basic, suportă programarea vizuală unde prototipul este dezvoltat prin crearea interfeţei utilizator pornind de la elemente standard şi asociind componente cu aceste elemente
- Există o bibliotecă bogată de componente care să sprijine acest tip de dezvoltare
- Acestea pot fi adaptate să potrivească unor cerinţe specifice ale aplicaţiei

Programarea Vizuală cu componente reutilizabile



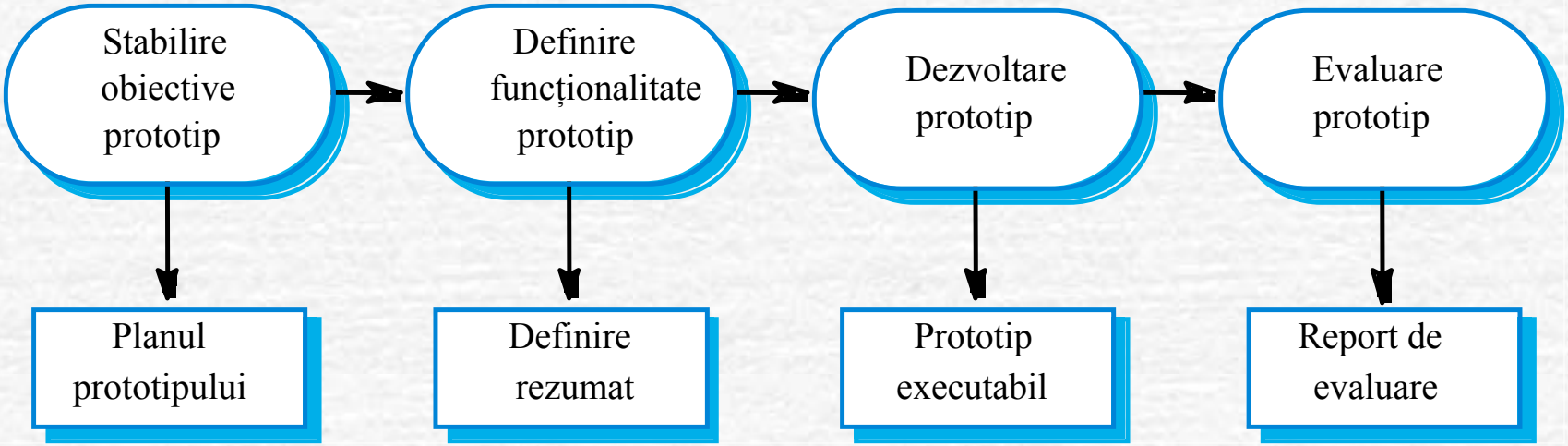
Programarea Vizuală. Probleme

- Coordonare dificilă a echipei de dezvoltare
- Nu există o arhitectură explicită a sistemului
- Dependenţele complexe între părţile de program pot cauza probleme de mentenanţă

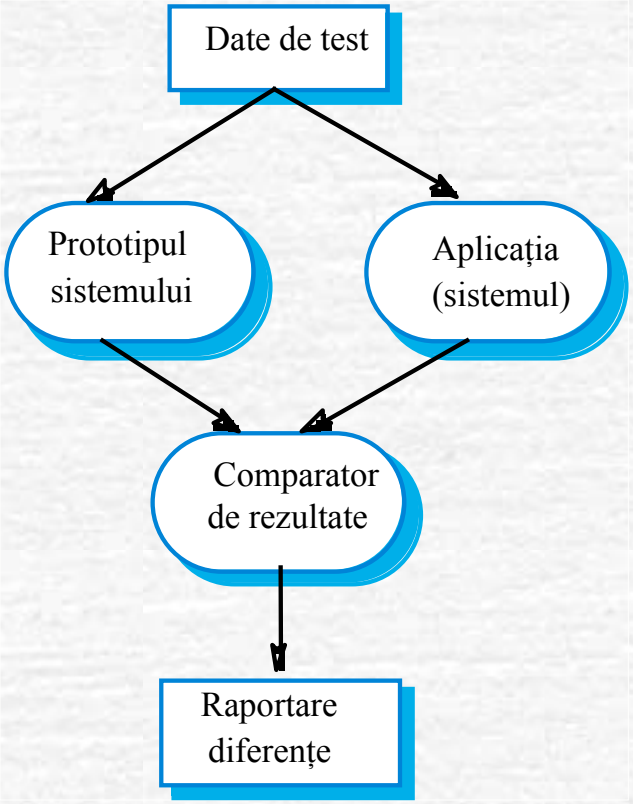
Prototipizarea Software

- Un prototip este o versiune iniţială a unui sistem utilizată pentru a demonstra concepte şi a încerca opţiuni de proiectare
- Un prototip poate fi utilizat în:
  - Procesul de inginerie a cerinţelor pentru a obţine şi valida cerinţele
  - În procesul de proiectare pentru a explora opţiunile şi a dezvolta interfeţe utilizator
  - În procesul de testare pentru a rula testele *back-to-back* (comparaţie între rezultatele date de sistem şi cele date de prototip pe acelaşi set de date de intrare

Procesul de Prototipizare



Testarea back-to-back



Beneficiile Prototipizării

- Îmbunătăţeşte utilizabilitatea sistemului
- O apropiere mai mare de necesităţile reale ale utilizatorului
- Creşterea calităţii proiectării
- Creşterea mentenabilităţii
- Reducerea efortului de dezvoltare

Utilitatea Prototipurilor. Probleme

- După dezvoltare în general se renunţă la prototipuri, nefiind utile ca bază pentru sistemul de producţie deoarece:
  - Poate fi foarte greu de ajustat sistemul pentru a satisface cerinţele non-funcţionale
  - În general prototipurile nu sunt documentate
  - Structura prototipului este in general degradată de schimbări rapide
  - Prototipul probabil nu va satisface cerinţele standardelor de calitate ale organizaţiei

Reutilizare

- În majoritatea disciplinelor inginereşti, sistemele sunt proiectate prin compunerea componentelor existente, care au fost deja utilizate în alte sisteme

- Ingineria software s-a concentrat mai mult spre dezvoltarea de cod original. La momentul actual este recunoscut faptul că pentru a obţine un software de bună calitate, rapid şi cu costuri mici, este nevoie adoptarea unui proces de proiectare bazat pe reutilizare sistematică a software-ului

Inginerie Software Bazată pe Reutilizare

- Reutilizare sisteme de aplicaţii
  - Un întreg sistem poate fi reutilizat fie prin incorporarea fără schimbări în alte sisteme (reutilizare COTS - Commercial off-the-shelf) sau prin dezvoltarea de familii de aplicaţii
- Reutilizare componente
  - Se pot reutiliza componentele unei aplicaţii, de la sub-sisteme până la obiecte singulare.
- Reutilizare obiecte şi funcţii
  - Componentele software care implementează un singur obiect sau funcţie bine definite pot fi de asemenea reutilizate

Reutilizare. Beneficii

- Creşte dependabilitatea (gradul de încredere)
  - Software-ul reutilizat, care a fost deja testat în sisteme funcţionale, este mai de încredere decât un software nou. Utilizarea iniţială scoate de obicei la lumină greşeli de proiectare sau implementare. Acestea sunt mai apoi reparate reducând astfel numărul de căderi când softul este reutilizat
- Risc redus al procesului de producţie
  - Incertitudinea legată de costuri la reutilizare este mai mică decât la dezvoltare. Acest lucru determină reducerea marjei de erori la estimarea costului proiectului, mai ales în situaţia reutilizării de componente mari (ex. sub-sisteme)
- Utilizarea eficientă a specialiştilor
  - În loc de a face acelaşi lucru la mai multe proiecte, specialiştii vor dezvolta software reutilizabil care să încapsuleze cunoştinţele lor

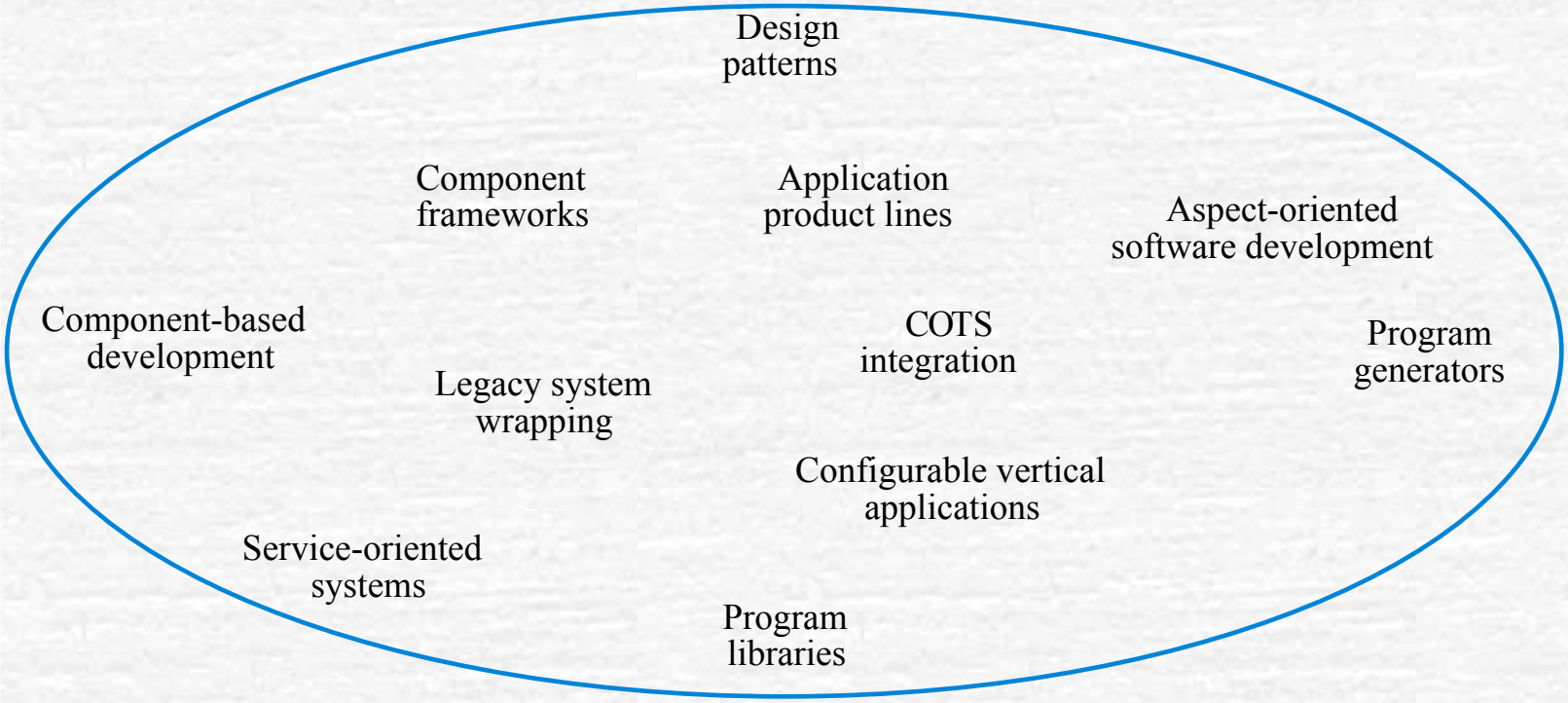
Reutilizare. Beneficii

- Alinierea la standarde
  - Anumite standarde, precum cele legate de interfeţele utilizator, pot fi implementate ca şi componente reutilizabile standard. În acest fel toate aplicaţiile vor arăta şi funcţiona la fel din punct de vedere al meniurilor, ferestrelor, butoanelor etc. Ca atare scade probabilitatea greşelilor utilizatorilor şi timpul de familiarizare cu o nouă interfaţă
- Accelerarea dezvoltării
  - Lansarea pe piaţă cât mai repede posibil este de multe ori mai importantă decât toate costurile de dezvoltare. Prin reutilizare se poate micşora şi timpul de dezvoltare şi cel de testare şi validare al sistemului

Reutilizare. Probleme

- Creşterea costului de mentenanţă
  - Dacă codul sursă a componentelor reutilizate nu este disponibil costul de mentenanţă poate creşte datorită incompatibilităţii componentelor cu schimbările ce apar
- Suportul scăzut în uneltele de dezvoltare
  - Nu toate uneltele CASE suportă dezvoltarea cu reutilizarea componentelor. Poate fi foarte dificil sau chiar imposibil de integrat o bibliotecă de componente în aceste unelte.
- Sindromul neîncrederii
  - Unii ingineri software preferă să rescrie cod în parte pentru că aşa se poate îmbunătăţi funcţionalitatea, în parte pentru că e mai interesant decât utilizarea unor componente existente
- Înţelegerea şi adaptarea componentelor
  - Componentele trebuie descoperite în biblioteci, înţelese şi adaptate

Posibilităţi de Reutilizare



Posibilităţi de Reutilizare

- *Design Patterns*
  - Reutilizarea cunoştinţelor abstracte despre probleme comune multor aplicaţii şi soluţiile cele mai potrivite ale acestora
- *Component-based Development*
  - Sistemele sunt dezvoltate prin integrarea de componente conform cu standardele unui model de compoziţie
- *Application Frameworks*
  - Colecţii de clase abstracte şi concrete care pot fi adaptate şi extinse pentru a crea un sistem
- *Legacy System Wrapping*
  - Sistemele moştenite pot fi înglobate prin definirea unui set de interfeţe care să asigure accesul la aceste sisteme

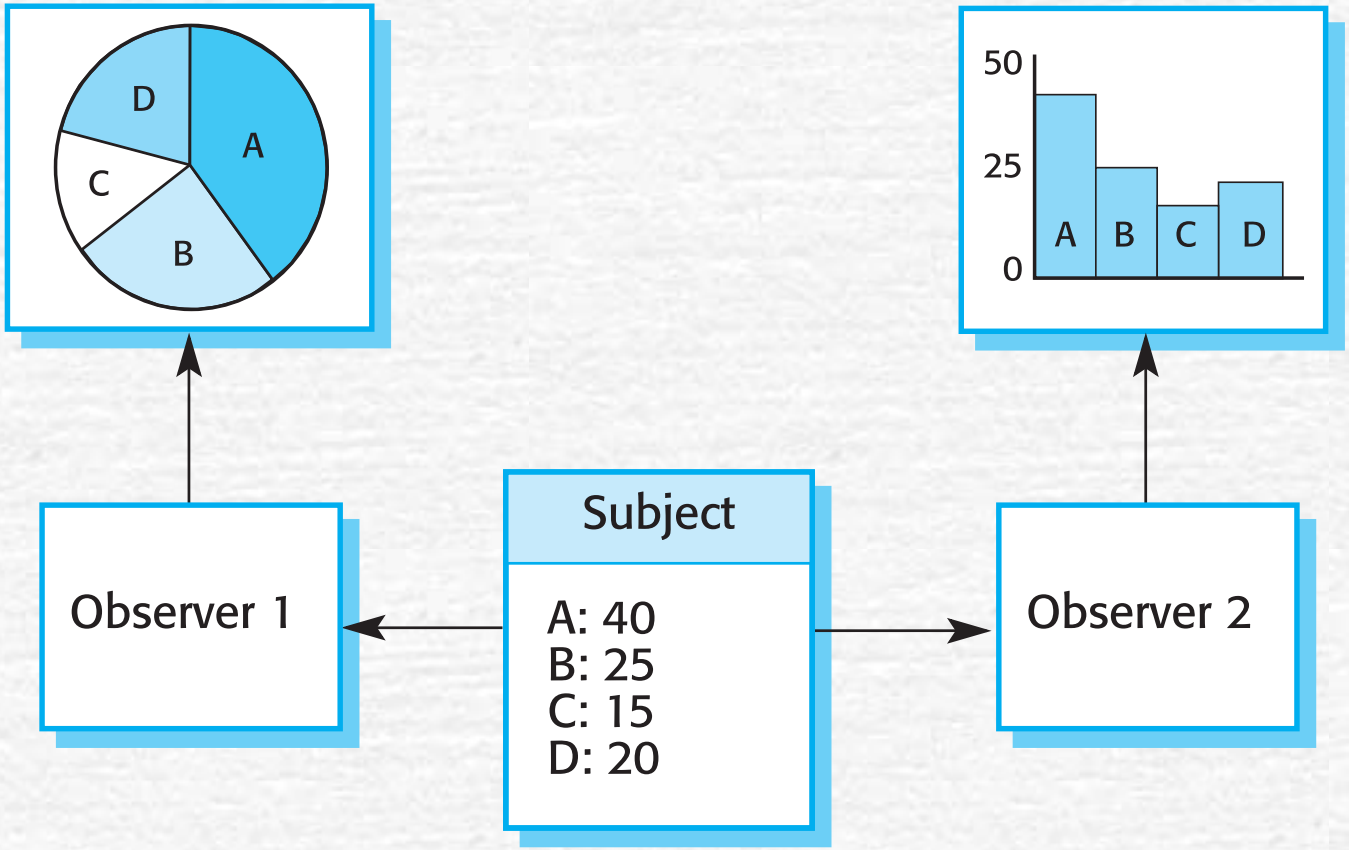
Posibilităţi de Reutilizare

- *Application Product Lines*
  - Un tip de aplicaţie este generalizat în jurul unei arhitecturi comune astfel încât să poată fi adaptată în diverse forme pentru a satisface diferiţi clienţi
- *COTS Integration*
  - Sistemele sunt dezvoltate prin integrarea sistemelor de aplicaţii existente
- *Configurable Vertical Applications*
  - Este proiectat un sistem generic astfel încât să poată fi reconfigurat pentru cerinţele diverşilor clienţi
- *Program Libraries*
  - Clase şi biblioteci de funcţii sunt crete spre a fi reutilizate
- *Program Generators*
  - Pot genera sisteme sau fragmente de sisteme într-un anumit domeniu
- *Aspect Oriented Software Development*
  - *Asigură injectarea unor funcţionalităţi în cod în momentul compilării*

Design Patterns

- Nume
  - Un nume sugestiv pentru identificarea tiparului
- Descriere tipar
- Descriere soluţie
  - Nu un proiect concret ci o machetă pentru soluţia de proiectare care poate fi instanţiată în diverse forme
- Consecinţe
  - Rezultatul şi părţile negative ale aplicării tiparului

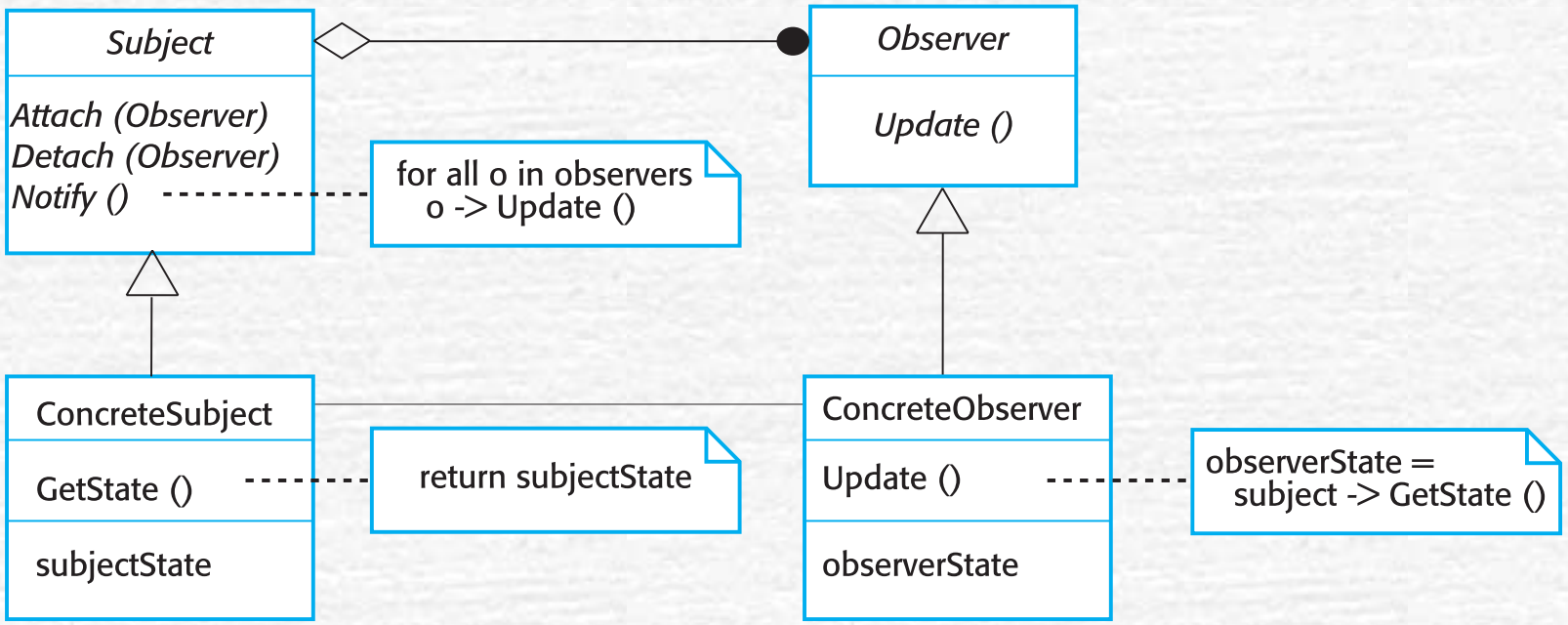
Exemplu: Tiparul Observer



Tiparul Observer

- Nume
  - Observer
- Descriere
  - Separă afişarea stării obiectului de obiectul însăşi
- Descrierea problemei
  - Utilizat când afişări diverse ale stării sunt necesare pentru un obiect
- Descrierea soluţiei
  - Este dată în continuare folosind UML
- Consecinţe
  - Optimizări pentru creşterea performanţei sunt nepractice

Tiparul Observer. Descrierea soluţiei



Cadre pentru Aplicaţii

- Cadrele (frameworks) sunt proiecte de sub-sisteme create din colecţii de clase abstracte şi concrete şi interfeţele dintre ele
- Sub-sistemul este implementat prin adăugarea componentelor pentru a umple părţi din proiect şi prin instanţierea claselor abstracte din cadru
- Cadrele sunt entităţi de mărimi moderate ce pot fi utilizate în diverse situaţii

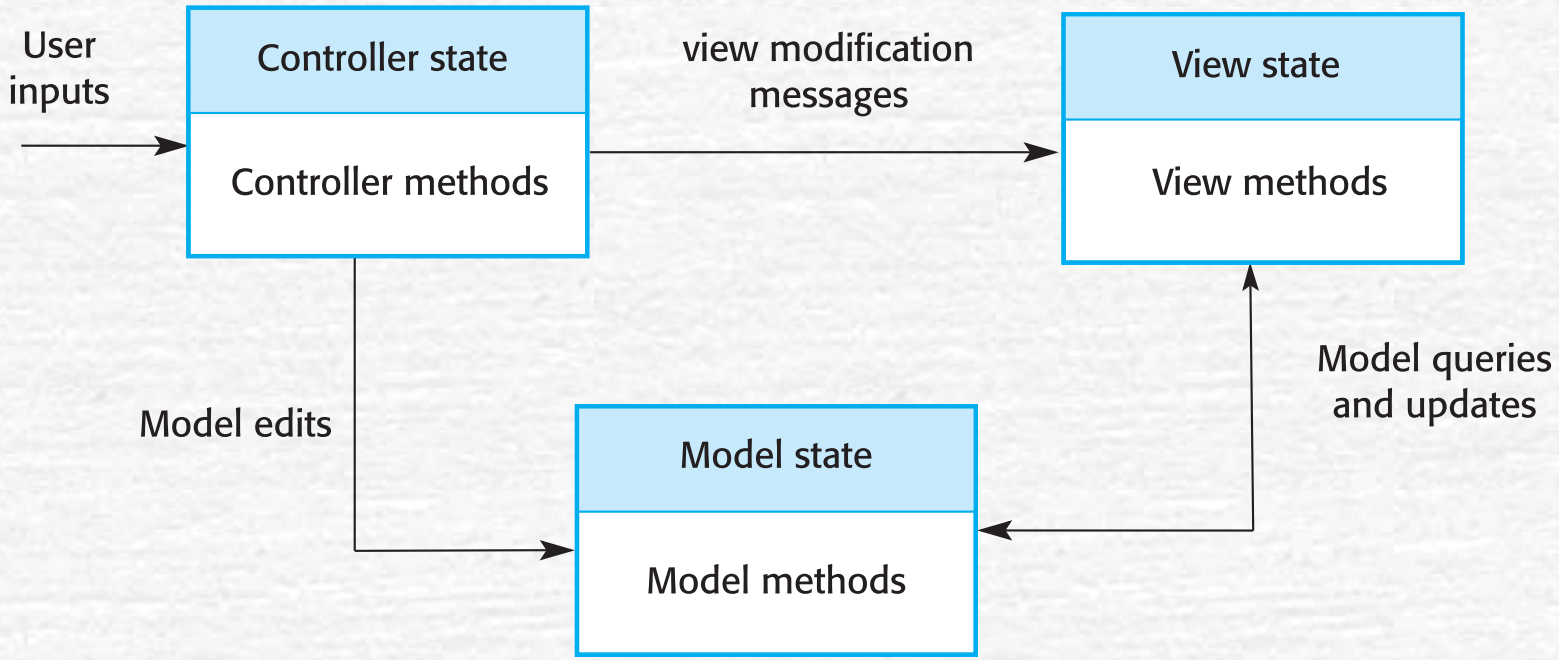
Clasele unui Cadru

- Cadre pentru infrastructura sistemului
  - Sprijină dezvoltarea de infrastructuri pentru sisteme precum cele de comunicaţii, interfeţe utilizator sau compilatoare
- Cadre pentru integrare middleware
  - Standarde şi clase care sprijină comunicarea între componente şi schimbul de informaţie
- Cadre pentru aplicaţii Enterprise
  - Sprijină dezvoltarea unor tipuri de aplicaţii specifice precum cele de telecomunicaţii sau sistemele financiare

Exemplu: Model-View Controller

- Cadru de infrastructură pentru proiectarea sistemelor cu interfaţă grafică
- Permite prezentări multiple ale unui obiect şi separarea interacţiunilor între aceste prezentări
- Cadrul MVC implică instanţierea mai multor design pattern-uri

Model-View Controller



Concluzii

- O abordare iterativă a dezvoltării software conduce la o livrare rapidă a aplicaţiei
- Metodele Agile sunt metode iterative care urmăresc reducerea supra-încărcării la dezvoltare permiţând producerea rapidă de software
- Programarea XP include practici precum: testarea sistematică, îmbunătăţirea continuă şi implicarea beneficiarului
- Puterea abordării testării în XP stă în testele executabile dezvoltate înainte de scrierea codului

Concluzii

- Mediile RAD includ: limbaje de programare pentru baze de date, generatoare de formulare şi legături către aplicaţii de birou
- Un prototip dispensabil este utilizat pentru a explora cerinţele şi opţiunile de proiectare
- Implementarea unui prototip dispensabil porneşte cu cerinţele cel mai puţin înţelese. În contrast, la dezvoltarea incrementală se porneşte cu cerinţele cele mai clare

Concluzii

- Avantajele reutilizării sunt: costuri scăzute, accelerarea dezvoltării software şi micşorarea riscului
- *Design Patterns* sunt abstracţii de nivel înalt care descriu soluţii de proiectare de succes
- Generatoarele de cod sunt un sprijin important pentru reutilizarea codului, cuprinzând seturi de concepte reutilizabile
- *Application Frameworks* sunt colecţii de obiecte abstracte şi concrete proiectate pentru a fi reutilizate prin specializare