UNIVERSITATEA TEHNICĂ "GHEORGHE ASACHI" DIN IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
AN UNIVERSITAR 2016-2017

Managementul Proiectelor Software

ASIST.ING. CORINA CÎMPANU

Conținutul cursului

MANAGEMENTUL PROIECTELOR SOFTWARE

Cursul nr. 2

METODOLOGII DE DEZVOLTARE A PROGRAMELOR

Capitolul 2. Metodologii de dezvoltare a programelor

O metodologie de dezvoltare a programelor, numită şi ***proces de dezvoltare software***, reprezintă o structură impusă asupra dezvoltării unui produs software. Este planul de acțiune privind succesiunea celor patru faze ale ingineriei programării: analiza, proiectarea, implementarea şi testarea.

Unele companii folosesc propriile metodologii. Altele folosesc metodologii comerciale. Însă fără o metodologie adecvată, dezvoltarea produsului este haotică.

Alegerea unei metodologii de dezvoltare trebuie să țină seama de natura fiecărui proiect: mărimea echipei, bugetul, tehnologia şi instrumentele utilizate, termenele limită, procesele existente deja.

Capitolul 2. Metodologii de dezvoltare a programelor

- Metodologia "codează și repară"
- Metodologia secvențială (modelul cascadă)
- Metodologia ciclică/ iterativă (modelul spirală, modelul Timeboxing)
- Metodologia hibridă ecluză
- Modelul V
- Metode formale
- Programarea extremă
- Metodologia Open Source
- Metodologia de dezvoltare Offshore (Outsourcing)

Capitolul 2. Metodologii de dezvoltare a programelor **Metodologia "codează și repară"**

- Abordarea „codează şi repară" (engl. "code and fix") este cea mai rapidă şi puțin eficientă metodologie;
- Nu există reguli de organizare şi se ignoră etapele cheie de dezvoltare;
- Aceasta este metoda utilizată de obicei de companiile de la început de drum, cu puțini dezvoltatori;
- Recomandarea în această situație este introducerea treptată a unor modalități formale de dezvoltare şi verificare.

Capitolul 2. Metodologii de dezvoltare a programelor **Metodologia secvențială**

- Este o metodologie generică în care cele patru faze urmează una alteia într-o manieră serială.
- Echipele care se ocupă de fiecare fază pot fi diferite, iar la fiecare tranziție de fază poate fi necesară o decizie managerială.
- Se recomandă pentru probleme bine înțelese, pentru automatizarea unor sisteme manuale existente sau pentru proiecte de scurtă durată.

Metodologia secvenţială

Capitolul 2. Metodologii de dezvoltare a programelor **Metodologia secvențială**

Avantaje

- Metodologia secvențială este potrivită când complexitatea sistemului este mică iar cerințele sunt statice.
- Este simplă, logică şi uşor de urmat.
- Forțează menținerea unei discipline de lucru în care analiza şi proiectarea au loc înaintea implementării.
- Se evită presiunea scrierii codului înainte de a cunoaşte precis ce produs va trebui de fapt construit şi deci scade probabilitatea introducerii unor părți de cod inutile sau ineficiente în produsul final.
- Un mare număr de sisteme software din trecut au fost construite cu o metodologie secvențială.

Capitolul 2. Metodologii de dezvoltare a programelor **Metodologia secvențială**

Dezavantaje

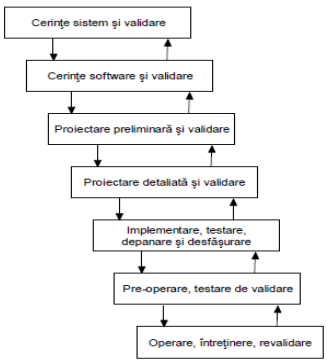
- Se acordă o foarte mare importanță fazei de analiză.
- Analiştii trebuie să definească toate detaliile aplicației încă de la început şi nu există un proces de corectare a erorilor după stabilirea cerințelor.
- Comunicarea dintre echipele desemnate pentru fazele de dezvoltare se limitează în principal la documentele realizate de o echipă şi trimise următoarei echipe.
- Într-un mediu economic dinamic, metodologia secvențială poate produce sisteme care, la vremea lansării, să fie deja învechite.

Capitolul 2. Metodologii de dezvoltare a programelor **Metodologia secvențială – Metodologia "Waterfall"**

Metodologia cascadă (engl. "waterfall") a fost propusă de Winston W. Royce (1970). Procesul „curge" de la etapă la etapă, ca apa într-o cascadă. După fiecare etapă există un pas de validare. În descrierea inițială există o întoarcere, un pas înapoi interactiv între fiecare două etape succesive.

Dezavantaje

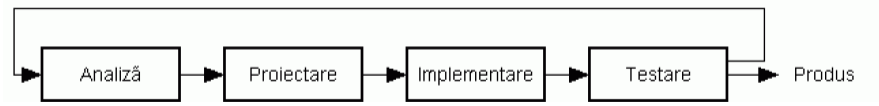
- Clientul obține o viziune practică asupra produsului doar în momentul terminării procesului de dezvoltare.
- De asemenea, modelul nu are suficientă putere descriptivă, în sensul că nu integrează activități ca managementul riscului sau managementul configurației.



Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia ciclică/ iterativă

- Se bazează pe ideea perfecționării incrementale a metodologiei secvențiale.
- Fazele sunt dispuse în cicluri care contribuie succesiv la realizarea sistemului final.
- În fiecare fază se consumă un timp mai scurt, după care urmează mai multe iterații prin toate fazele.
- Pe măsură ce procesul înaintează, sunt generate din ce în ce mai multe detalii.
- În final, după câteva cicluri, sistemul este complet şi gata de lansare.
- Procesul poate continua pentru lansarea mai multor versiuni ale produsului.
- Este recomandată pentru proiecte unde timpul este esențial, unde apar riscuri asociate cu o durată mare a proiectului sau unde cerințele nu pot fi cunoscute exact de la început.

Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia ciclică/ iterativă



Avantaje

- Permite feedback-ul de la fiecare echipă în ceea ce priveşte corectitudinea cerințelor.
- Există etape în care pot fi corectate eventualele greşeli.
- Clientul poate studia rezultatul şi poate oferi informații importante mai ales în faza dinaintea lansării produsului.
- Dezvoltarea se poate adapta mai bine progresului tehnologic: pe măsură ce apar noi soluții, ele pot fi încorporate în produs.
- Pot avea loc livrări regulate sau rapide.
- Permite prioritizarea cerințelor.

Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia ciclică/ iterativă

Dezavantaje

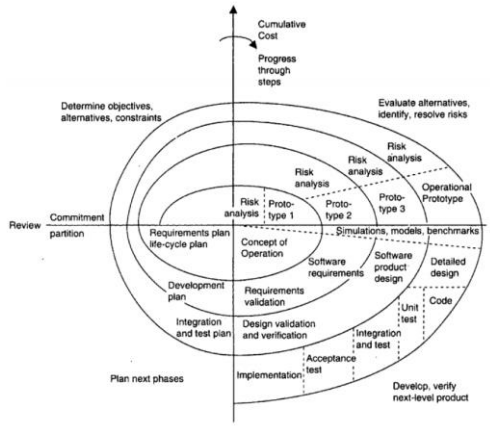
- De vreme ce nu există constrângeri asupra echipei de analiză să facă lucrurile cum trebuie de prima dată, acest fapt poate duce la scăderea responsabilității.
- Echipa de proiectare nu are o viziune globală asupra produsului şi deci nu poate realiza o arhitectură completă.
- Fiecare iterație necesită un timp suplimentar de planificare.
- Ciclurile pot continua fără o condiție clară de terminare.
- Echipa de implementare poate fi pusă în situația nedorită în care cerințele şi arhitectura sistemului sunt în permanentă schimbare iar renunțarea la părți de cod deja scrise determină creşterea costurilor.

Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia iterativă – Metodologia spirală

Metodologia spirală, propusă de Barry Boehm (1988), consideră că fiecare etapă a procesului de dezvoltare conține o serie de activități comune:

- pregătirea: se identifică obiectivele, alternativele, constrângerile;
- managementul riscului: analiza şi rezolvarea situațiilor de risc;
- activități de dezvoltare specifice pasului curent, de exemplu analiza cerințelor sau scrierea de cod;
- planificarea următorului stadiu: termenele limită, resursele necesare, revizuirea stării curente a proiectului.

Procesul începe în centrul spiralei. Fiecare ciclu terminat reprezintă o etapă. Pe măsură ce se parcurge spirala, produsul se maturizează. Cu fiecare ciclu, sistemul se apropie de soluția finală.



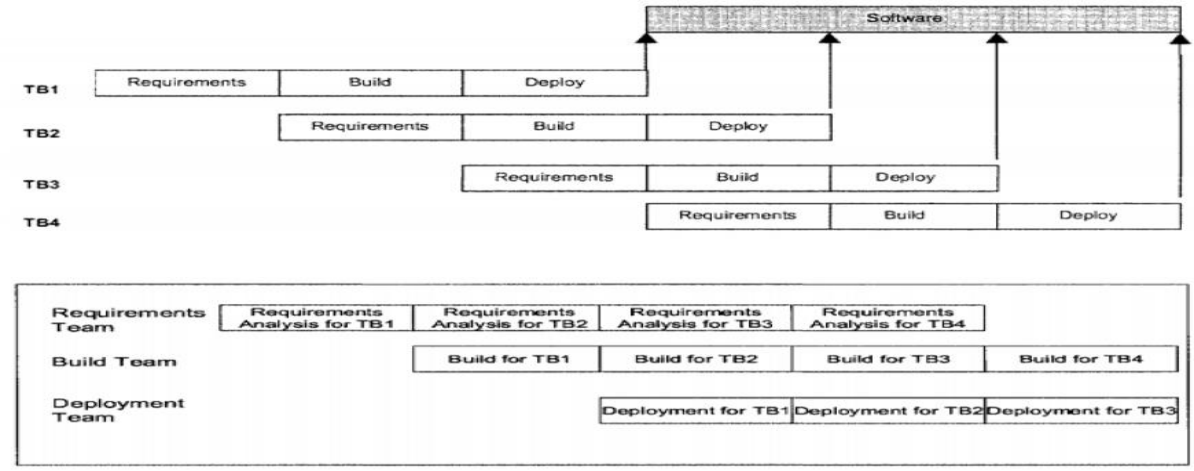
Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia iterativă – Metodologia Timeboxing

- Încearcă accelerarea procesului de dezvoltare prin aplicarea paralelismului între diferite iterații. Unitatea de bază este o *casetă de timp* (engl. "time box"), cu durată fixă.
- De aceea, un factor cheie în alegerea cerințelor este ce poate fi realizat într-o casetă de timp.
- Acest fapt contrastează cu metodologiile iterative clasice în care mai întâi se hotărăşte funcționalitatea şi aba apoi timpul de livrare.
- Fiecare casetă de timp este împărțită într-o succesiune de etape, ca în modelul cascadă.
- Duratele acestor etape trebuie să fie de asemenea aproximativ egale.
- Trebuie să existe o echipă dedicată pentru fiecare etapă.
- Când o echipă termină etapa *i* din caseta de timp *k*, îi predă livrabilul echipei dedicate fazei *i*+1 şi apoi începe lucrul la aceeaşi etapă din caseta de timp *k*+1.
- Dacă o casetă de timp are *T* zile şi *n* etape, prima livrare are loc după *T* zile, iar următoarele la câte *T* / *n* zile distanță.
- Costul suplimentar pentru scurtarea timpului de livrare stă în utilizarea resurselor.

Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia iterativă – Metodologia Timeboxing

- De exemplu, dacă *T* este 9 săptămâni iar *n* este 3, prima livrare va fi făcută după 9 săptămâni, a doua la 12 săptămâni, a treia la 15 ş.a.m.d. Execuția liniară a iterațiilor ar conduce la termene de livrare de 9, 18, respectiv 27 de săptămâni.
- De exemplu, dacă pentru un proiect analiza şi proiectarea durează 2 săptămâni cu 2 persoane, implementarea 2 săptămâni cu 4 persoane iar testarea 2 săptămâni cu 3 persoane, pentru execuția serială a iterațiilor sunt necesare 4 persoane. Pentru metodologia timeboxing, sunt necesare 2+4+3 = 9 persoane.

Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia iterativă – Metodologia Timeboxing



Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia iterativă – Metodologia Timeboxing

Avantajul principal este

- scurtarea ciclului de livrare.

Dezavantajele constau în:

- creşterea dimensiunii echipei,
- complexitatea managementului şi costul ridicat.

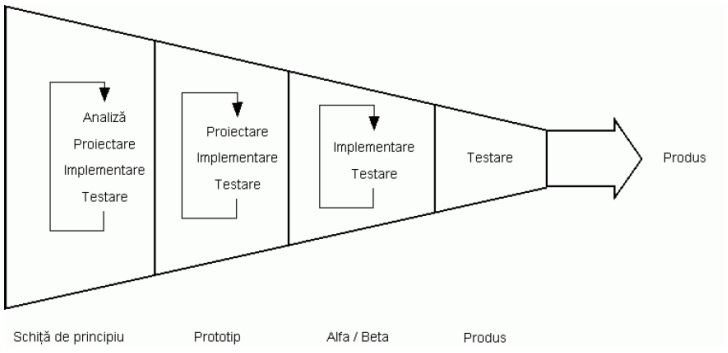
Metodologia este recomandată atunci când sunt necesare termene de livrare scurte şi există flexibilitate în gruparea funcționalităților, astfel încât casetele de timp să aibă lungimi egale.

Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia hibridă ecluză

Metodologia ecluză (engl. "watersluice"), propusă de Ronald LeRoi Burback (1998), combină progresul constant al metodologiei secvențiale cu incrementele iterative ale metodologiei ciclice.

- În prima etapă, *schița de principiu*, echipele lucrează iterativ la toate fazele problemei. Echipa de implementare se concentrează asupra sarcinilor critice. Unul din scopurile acestei etape este de a se convinge echipele că proiectul este fezabil, că o soluție poate fi găsită şi pusă în practică.
- În cea de a doua etapă, de *prototip*, cerințele sunt înghețate. După ce sarcinile critice au fost terminate, echipa de implementare se poate concentra pe extinderea acestora. Unul din scopurile acestei etape este de a convinge persoanele din afara echipelor (părțile interesate) că o soluție este posibilă.
- În versiunile *alfa şi beta*, arhitectura este înghețată, iar accentul cade pe implementare şi asigurarea calității. Rolul acestei etape este de a convinge clienții că aplicația va fi un produs adevărat. Înainte de lansarea *produsului*, implementarea este înghețată şi accentul cade pe testare. Scopul este realizarea unui produs competitiv.

Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia hibridă ecluză

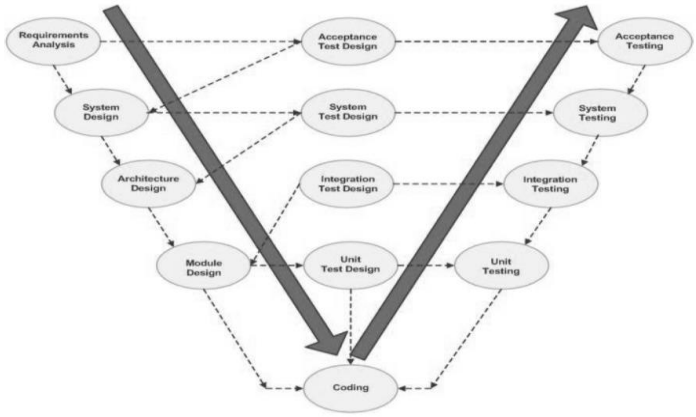


Capitolul 2. Metodologii de dezvoltare a programelor **Modelul V**

Modelul V (germ. "V-Modell") evidențiază testarea pe tot parcursul ciclului de dezvoltare. A fost creat pentru reglementarea dezvoltării produselor software în administrația federală germană.

- Modelul poate fi considerat o extensie a metodologiei cascadă, însă pune accentul pe relația dintre fiecare fază de dezvoltare şi faza de testare asociată.
- Trecerea la faza următoare se face numai după ce toate livrabilele din faza curentă au trecut testele de verificare şi validare.

Capitolul 2. Metodologii de dezvoltare a programelor **Modelul V**



Capitolul 2. Metodologii de dezvoltare a programelor
Metode formale

Acest model de dezvoltare apelează la formalismul şi rigoarea matematicii pentru construirea specificațiilor. Metodele formale sunt potrivite în special pentru sisteme cu cerințe critice.

Avantaje:

- precizia obținută prin specificarea formală;
- păstrarea corectitudinii în timpul transformării specificațiilor în cod executabil;
- oferă posibilitatea generării automate de cod;
- verificarea consistenței şi testarea prin metode similare demonstrării automate de teoreme.

Dezavantaje:

- specificarea formală este de obicei o barieră de comunicare între client şi analist;
- necesită personal foarte calificat, deci mai scump;
- folosirea impecabilă a tehnicilor specificării formale nu implică neapărat obținerea de programe sigure, deoarece anumite aspecte critice pot fi omise din specificații.

Capitolul 2. Metodologii de dezvoltare a programelor
Programarea extremă

Programare extremă (engl. "eXtreme Programming") este o metodologie „agilă", potrivită dezvoltării rapide de aplicații, propusă de Kent Beck (1996). Dintre caracteristicile XP, amintim următoarele:

- Echipa de dezvoltare nu are o structură ierarhică. Fiecare contribuie la proiect folosind maximul din cunoştințele sale;
- Scrierea de cod este activitatea cea mai importantă;
- Proiectul este în mintea tuturor programatorilor din echipă, nu în documentații, modele sau rapoarte;
- La orice moment, un reprezentant al clientului este disponibil pentru clarificarea cerințelor;
- Codul se scrie cât mai simplu;
- Se scrie mai întâi cod de test;
- Dacă apare necesitatea rescrierii sau eliminării codului, aceasta se face fără milă;
- Modificările aduse codului sunt integrate continuu, de câteva ori pe zi;
- Se programează în perechi. Echipele se schimbă la sfârşitul unei iterații (1-2 săptămâni);
- Se lucrează 40 de ore pe săptămână, fără lucru suplimentar.

Capitolul 2. Metodologii de dezvoltare a programelor **Programarea extremă - AGILE**

Manifest pentru dezvoltarea agilă de software (2001)

Descoperim continuu metode mai bune pentru dezvoltarea de software prin aplicarea acestora şi prin ajutorul acordat altora pentru a le aplica. Astfel, am ajuns să prețuim:

- *Oamenii şi interacțiunile* mai mult decât procesele şi instrumentele
- *Software-ul funcțional* mai mult decât documentația cuprinzătoare
- *Colaborarea cu clienții* mai mult decât negocierea contractelor
- *Adaptarea la schimbare* mai mult decât respectarea unui plan
- Mai exact, deşi elementele din dreapta sunt valoroase, le prețuim mai mult pe cele din stânga.

Capitolul 2. Metodologii de dezvoltare a programelor **Programarea extremă - AGILE**

Principiile Manifestului agil

1. Prima noastră prioritate este aceea de a satisface clienții prin livrări neîntârziate şi continue de software valoros.
2. Acceptăm schimbarea cerințelor, chiar şi mai târziu în procesul de dezvoltare. Procesele agile valorifică schimbarea în avantajul competițional al clientului.
3. Livrăm frecvent software funcțional, de la câteva săptămâni până la câteva luni, preferând intervalele de timp mai scurte.
4. Oamenii de afaceri şi dezvoltatorii trebuie să lucreze zilnic împreună pe parcursul proiectului.
5. Construim proiecte în jurul persoanelor motivate. Le oferim mediul şi sprijinul de care au nevoie şi avem încredere în ei că vor finaliza lucrarea.
6. Metoda cea mai eficientă şi eficace pentru comunicarea informațiilor către şi în cadrul echipei de dezvoltare o reprezintă conversația față în față.
7. Software-ul funcțional este principala măsură a progresului.
8. Procesele agile promovează dezvoltarea durabilă. Sponsorii, dezvoltatorii şi utilizatorii trebuie să fie capabili să mențină un ritm constant pe timp nedefinit.
9. Atenția continuă pentru superioritate tehnică şi proiectare bună sporeşte agilitatea.
10. Simplitatea – arta maximizării cantității de muncă neefectuată – este esențială.
11. Cele mai bune arhitecturi, cerințe şi proiectări provin din cadrul echipelor care se organizează singure.
12. La intervale regulate, echipa reflectează asupra modului în care poate deveni mai eficace şi apoi îşi modifică în mod corespunzător comportamentul.

Capitolul 2. Metodologii de dezvoltare a programelor **Programarea extremă - AGILE**

Carta drepturilor dezvoltatorului:

- Ai dreptul să ştii ceea ce se cere, prin cerințe clare, cu declarații clare de prioritate;
- Ai dreptul să spui cât îți va lua să implementezi fiecare cerință, şi să îți revizuieşti estimările în funcție de experiență;
- Ai dreptul să îți accepți responsabilitățile, în loc ca acestea să-ți fieatribuite;
- Ai dreptul să faci treabă de calitate în orice moment;
- Ai dreptul la liniște, distracție şi la muncă productivă şi plăcută.

Carta drepturilor clientului:

- Ai dreptul la un plan general, să ştii ce poate fi făcut, când, şi la ce preț;
- Ai dreptul să vezi progresul într-un sistem care rulează şi care se dovedeşte că funcționează trecând teste repetabile pe care le specifici tu;
- Ai dreptul să te răzgândeşti, să înlocuieşti funcționalități şi să schimbi prioritățile;
- Ai dreptul să fii informat de schimbările în estimări, suficient de devreme pentru a putea reduce cerințele astfel ca munca să se termine la data prestabilită. Poți chiar să te opreşti la un moment dat şi să rămâi cu un sistem folositor care să reflecte investiția până la acea dată.

Capitolul 2. Metodologii de dezvoltare a programelor **Metodologia Open Source**

Conceptul de dezvoltare *open-source* (sursă la vedere) este o abordare recentă, apărută ca urmare a dezvoltării mijloacelor de comunicare prin internet: email, grupuri de discuție, site-uri dedicate dezvoltării colaborative de software. Exemple clasice de produse dezvoltate în această manieră sunt sistemul de operare Linux şi browser-ul Netscape 5. Codul sursă este transmis utilizatorului final într-o manieră non-proprietară (fără patent), pe baza unei licențe *open-source* (de exemplu GNU).

O iterație tipică:

Inițiatorul realizează proiectarea şi implementarea, individual sau în echipă;

Ceilalți dezvoltatori sau comunitatea de utilizatori realizează testarea şi eventual depanarea;

- Noile funcționalități dorite şi erorile depistate sunt trimise inițiatorului proiectului;
- Se lansează o nouă versiune cu defectele corectate şi se analizeazănoile cerințe;
- Se distribuie o listă de sarcini către comunitatea de utilizatori, căutându-se membri voluntari care să execute sarcinile de pe listă.

Capitolul 2. Metodologii de dezvoltare a programelor **Metodologia Open Source**

Avantaje:

- Preț mult mai mic decât al produselor proprietare;
- Ideile vin de la o întreagă comunitate de utilizatori şi programatori;
- Accesul la surse poate ajuta detectarea şi repararea defectelor.

Dezavantaje:

- Lipsa garanțiilor de calitate;
- Risc privind cazurile de încălcare a proprietății intelectuale, dacă programatorii folosesc surse proprietare, greu de verificat;
- Unele licențe prevăd ca produsele derivate să fie tot *open-source*,ceea ce obligă companiile care utilizează aceste produse să îşi publice propriul cod.

Capitolul 2. Metodologii de dezvoltare a programelor **Metodologia de dezvoltare Offshore**

Companiile utilizează *outsourcing*-ul pentru funcțiile care pot fi dezvoltate mai ieftin în altă țară. Un proiect tipic are următoarele etape:

- Inițierea proiectului (local);
- Analiza cerințelor (local);
- Proiectarea de nivel înalt (local);
- Proiectarea detaliată (offshore);
- Implementarea (offshore);
- Testarea (offshore sau local);
- Livrarea (local).

Capitolul 2. Metodologii de dezvoltare a programelor
Metodologia de dezvoltare Offshore

Motive pentru outsourcing:

- Reducerea costurilor;
- Creşterea eficienței;
- Concentrarea asupra obiectivelor critice ale proiectului;
- Accesarea flexibilă a unor resurse care altfel nu ar fi accesibile, deexemplu personal înalt calificat;
- Dimensiunea mare a proiectului.

Dezavantaje:

- Control managerial mai redus;
- Probleme de confidențialitate şi securitate;
- Risc în cazul falimentului companiei *offshore*;
- Probleme de limbă, fus orar.

Capitolul 2. Metodologii de dezvoltare a programelor

Concluzii

- Simpla adoptare a unei metodologii fără o analiză temeinică acerințelor şi contextului de lucru nu este recomandată.
- Metodologia este o tactică ce trebuie considerată numai după determinarea strategiei generale a companiei.