Conținut:

Capitolul 1 – Sistemeinformatice. Probleme şi perspective

Capitolul 2 – Etapele de dezvoltare a sistemelor de programe

Capitolul 3 – Paradigmele de dezvoltare a sistemelor software

Capitolul 4 – UML- Limbaj unificat de modelare

Capitolul 5 – Principii de proiectare orientată pe obiect

Capitolul 6 – Şabloane de proiectare software

Capitolul 7 – Proiectarea sistemelor software

Capitolul 8 – Testarea sistemelor software

Capitolul 9 – Estimarea costurilor unui proiect software

Capitolul 10 – Calitatea sistemelor software

Capitolul 11 – Evaluarea sistemelor software

Bibliografie:

1. Cornelia Novac Ududec, Ingineria sistemelor de programe - *Ingineria programării, Ediție adăugită şi revizuită*, Editura Alma Mater, Bacău, 2011;
2. Rotar Dan, Ingineria programelor, Editura Alma Mater, Bacău, 2007
3. Pilat Florin, s.a., Metode, tehnici şi instrumente în ingineria programării, EdituraTehnică, Bucureşti 1985
4. Vaduva Ilie, Baltac Vasile, Florescu Vasile, s.a., Ingineria programării. Vol I, II – Editura Academiei, Bucureşti, 1986

1. Sistemele informatice. Probleme şi perspective

1.1 Introducere

Următoarele exemple oferă o imagine mai completă a complexității la care a ajuns software-ul în zilele noastre:

- Sistemul de rezervare a biletelor pentru compania aeriană KLM conținea, în anul 1992, 2.000.000 de linii de cod în limbaj de asamblare;
- Sistemul de operare System V versiunea 4.0 (UNIX) a fost obținut prin compilarea a 3.700.000 linii de cod;
- Sistemele de programe pentru controlul navetelor spațiale NASA au circa 40 de milioane de linii de cod;
- Pentru realizarea sistemului de operare IBM OS360 au fost necesari 5000 de ani-muncă.om.

Se face o paralelă între ingineria software şi ingineria construcțiilor:

- cuşcă pentru câine, putem să mergem în grădină, să căutam lemne şi cuie, să luăm un ciocan şi să începem să lucrăm. Avem şanse destul de bune să reuşim, mai ales dacă suntem

îndemânatici. Dacă totuşi nu ne iese, putem încerca a doua zi din nou cu alte lemne şi alte cuie. Şi totuşi, dacă câinele nu încape în cuşcă, putem să ne cumpărăm alt câine.

- când dorim să construim o casă:
  - angajăm un arhitect care să ne facă un proiect, sau să cumpărăm un proiect standard de casă.
  - negociem cu o firmă de construcții: prețul, durata de realizare, calitatea finisajelor.
  - Nu ne permitem să riscăm economiile familiei pe o construcție care se va dărâma la o rafală de vânt. În plus, dacă membrilor familiei nu le place orientarea ferestrelor sau peisajul, nu îi putem schimba cu alții (în cel mai rău caz, ne schimbă ei pe noi).

Inginerii constructori întocmesc:

- planuri,
- construiesc machete,
- studiază proprietățile materialelor folosite
- fac rapoarte privind progresul operațiunilor.

"Constructorii" de aplicații software trebuie să procedeze la fel pentru ca dezvoltarea programelor să nu mai fie un proces impredictibil.

Un raport prezentat de către o companie de software, în care erau analizate diverse proiecte şi stadiile lor de finalizare, a constatat că:

- 2% din sistemele software contractate au funcționat de la predare;
- 3% din sistemele software au putut funcționa după câteva modificări;
- 29% au fost predate dar n-au funcționat niciodată;
- 19% au fost folosite dar au fost abandonate;
- 47% au fost plătite dar niciodată predate.

Prima definiție dată ingineriei software a fost formulată astfel (F. L. Bauer):

Ingineria software este stabilirea şi utilizarea de principii inginereşti solide pentru a obține în mod economic programe sigure şi care funcționează eficient pe maşini de calcul reale.

În IEEE Standard Glossary of Software Engineering Technology (1983) ingineria software este definită după cum urmează:

Ingineria software, ingineria sistemelor de programe (ingineria programării), reprezintă abordarea sistematică a dezvoltării, funcționării, întreținerii, şi retragerii din funcțiune a programelor.

A doua definiție adaugă însă referiri la perioade importante din viața unui program, ce urmează creării şi funcționării sale, şi anume întreținerea şi retragerea din funcțiune.

Se consideră că ingineria software are următoarele caracteristici importante:

- este aplicabilă în producerea de sisteme de programe mari;
- este o ştiință inginerească;
- scopul final este îndeplinirea cerințelor clientului.

Statitstici:

- în SUA se cheltuiesc anual 250 de miliarde de dolari pentru producția de software, dar 33% dintre proiectele informatice eşuează, adică 80 de miliarde de dolari se pierd.
- 83% dintre proiectele informatice au probleme.

- că proiectarea şi realizarea aplicațiilor informatice nu pot fi făcute oricum ci trebuie să respecte normele şi metodologiile standardizate, în primul rând PMI (Project Managemant Institute) şi apoi pe cele specifice aplicațiilor software.

Un program este ***fiabil*** dacă funcționează si continuă să funcționeze fără întreruperi şi erori un interval de timp. = rezistența la condițiile de funcționare.

Un sistem de operare trebuie să fie fiabil pentru că este obligatoriu să funcționeze o perioadă suficient de lungă de timp fără să clacheze, indiferent de programele care rulează pe el, chiar dacă nu totdeauna la performanțe optime.

Programul este ***sigur*** dacă funcționează corect, fără operații nedorite. Un program pentru un automat bancar trebuie să fie sigur, pentru a efectua tranzacțiile în mod absolut corect, chiar dacă funcționarea sa poate fi întreruptă din când în când. Atunci când funcționează însă, trebuie să funcționeze foarte bine.

Un program are o ***eroare*** dacă nu se comportă corect. Se presupune că dezvoltatorul ştia ce ar fi trebuit să execute programul, iar comportamentul greşit nu este intenționat.

Ingineria software are ca scop obținerea de sisteme funcționale chiar şi atunci când teoriile şi instrumentele disponibile nu oferă răspuns la toate provocările ce apar. Inginerii fac ca lucrurile să meargă, ținând seama de restricțiile organizației în care lucrează şi de constrângerile financiare.

Problema fundamentală a ingineriei software este ***satisfacerea cerințelor utilizatorului***. Aceasta trebuie realizată nu punctual, nu imediat, ci într-un mod flexibil şi pe termen lung.

Ingineria software se ocupă de toate etapele dezvoltării programelor, de la achiziția cerințelor de la client până la întreținerea şi retragerea din folosință a produsului livrat. Pe lângă cerințele funcționale, clientul doreşte (de obicei) ca produsul final să fie realizat cu costuri de producție cât mai mici. De asemenea, este de dorit ca aceasta să aibă performanțe cât mai bune (uneori direct evaluabile), un cost de întreținere cât mai mic, să fie livrat la timp, şi să fie sigur.

Rezumând, atributele cheie ale unui produs software se referă la:

- ***Mentenabilitate****,* posibilitatea de a putea fi *întreținut*: Un produs cu un ciclu de viață lung este supus deseori modificărilor, de aceea el trebuie foarte bine documentat;
- ***Fiabilitate***: produsul trebuie să se comporte după cerințele utilizatorului şi să nu „cadă” mai mult decât e prevăzut în specificațiile sale;

- ***Eficiență***: produsul nu trebuie să folosească în pierdere resursele sistemului ca memoria sau timpul de procesare;
- ***Interfața*** potrivită pentru utilizator: interfața trebuie să țină seama de capacitatea şi cunoştințele utilizatorului.

Optimizarea tuturor acestor atribute e dificilă deoarece unele se exclud pe altele (de exemplu, o mai bună interfață pentru utilizator poate micşora eficiența produsului).

În cazurile în care eficiența este critică, acest lucru trebuie specificat explicit încă din faza de preluare a cerințelor utilizatorului, precum şi compromisurile pe care ea le implică privind ceilalți factori.

Trebuie spus că ingineria software nu rezolvă toate problemele care apar atunci când se scriu programe dar, în momentul de față, ea ne poate spune sigur ce să *nu* facem.

1.2 Probleme ale software-ului

Una dintre cele mai întâlnite probleme ale sistemelor software este "proiectarea greşită" (în engl. ”bad design”). Pentru a rezolva problema unui design greşit trebuie mai întâi definită această problemă şi analizate cauzele ei.

Definiția unei ”proiectări greşite”

O piesă software care satisface cerințele impuse, legate de funcționalitate, are un "design greşit", dacă îndeplineşte cel puțin una din condițiile de mai jos:

1. Este dificil de modificat, pentru că orice modificare implică modificări în multe alte părți ale sistemului, această caracteristică numindu-se **rigiditate;**
2. Dacă se face o modificare, alte părți ale sistemului nu mai funcționează; această caracteristică se numeşte **fragilitate**;
3. Este dificil de refolosit în altă aplicație pentru că nu poate fi detaşată de aplicația curentă. Această caracteristică se numeşte **imobilitate**.

Cauze ale unui ”design greşit"

Una din cauzele **rigidității**, **fragilității** şi **imobilității** unei proiectări este **interdependența** modulelor implicate în acel design.

Un design este **rigid** dacă nu poate fi modificat cu uşurință.

Rigiditate este cauzată de faptul că o singură modificare într-un modul produce o cascadă de modificări în modulele dependente de acesta, datorită interdependenței.

Când numărul de modificări (care survin ca urmare a primei modificări) nu poate fi prevăzut, atunci impactul modificării nu poate fi estimat, şi implicit nici costul modificării.

Fragilitatea este tendința unui program de a nu mai funcționa atunci când se face o modificare.

În acest caz, survin o serie de noi probleme; cel mai adesea, acestea sunt de natură diferită decât a modificării inițiale. Rezolvarea acestora conduce la alte probleme. O mare fragilitate conduce la un software de o calitate scăzută.

Un design este **imobil** atunci când un modul din program, care se doreşte a fi detaşat şi folosit într-o altă aplicație, este dependent de detaliile din restul aplicației. Adesea, costul pentru separarea modulului dorit de restul aplicației este mult mai mare decât dacă se realizează un nou modul, acesta fiind unul dintre motivele pentru care se renunță la re-folosirea / re-utilizarea piesei de software.

Un sistem software este un sistem dinamic, care de-a lungul ciclului de viață se modifică inevitabil.

Modificarea cerințelor/specificațiilor pentru un sistem software poate duce la dezvoltarea unui design greşit.

Specificarea cerințelor se realizează într-un document scris care trebuie să poată fi citit şi înțeles atât de beneficiar cât şi de proiectant. Altfel, beneficiarul nu poate să-l avizeze.

Motive pentru care cerințele se schimbă:

- Beneficiarul (clientul) vine cu noi cerințe, adăugării / modificări de funcții;
- Analistul sau persoana care se ocupă de proiect (şi a formulat inițial cerințele aplicației) a interpretat greşit cererile clientului;

- În timpul procesului de discuții cu beneficiarul, unele din cerințele inițiale ale clientului pot fi "uitate". În faza de feedback de la client, aceste cerințe sunt reconsiderate. Ca urmare, apar modificări ale programului;
- alte cauze: o lege nouă, o nouă politică de aprovizionare a firmei, o reorganizare sau o fuziune a firmei pentru care este dezvoltată aplicația, etc.
- Cu cât durata de viață a proiectului este mai lungă, cu atât este mai vulnerabil la modificări de acest tip.

Atât sistemele cu un design bun cât şi cele cu design greşit sunt supuse modificărilor. Diferența dintre ele este că proiectarea bună este stabilă când sunt supuse modificării.

1.3 Satisfacerea cerințelor utilizatorilor şi costul software-ului

Nici un produs, deci nici produsele software, nu vor fi solicitate şi utilizate dacă ele nu răspund unor nevoi ale utilizatorilor.

Cu cât sunt mai bine acoperite cerințele şi cu cât produsul va răspunde mai bine acestor solicitări, cu atât cererea pentru sistemul (produsul) respectiv va fi mai mare.

Statisticile arată că în țările puternic industrializate, ponderea ocupată de costul software-ului în produsul național brut este în continuă creştere.

Costul este influențat, şi determinat totodată, de:

- productivitatea de programare (10-20 instr / zi),
- predicția timpului în care se va realiza produsul final,
- costul hardware-ului în raport cu cel al software-ului,
- utilizarea generatoarelor de programe, etc.

Costul software-ului este dat în principal de salarii.

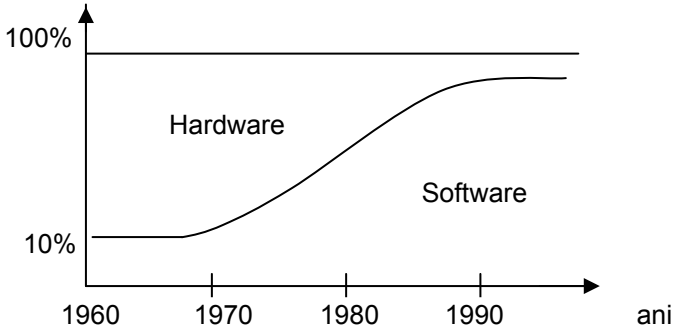

Fig.1.1. Relația între costul hardware-ului şi software-ului în timp

Rezumând, se poate spune că software-ul este scump, în primul rând din cauza productivității scăzute a programatorilor.

Astfel se naşte, în mod natural, întrebarea : Cum poate fi scăzut costul? Este interesant de văzut care parte din dezvoltarea unui produs software costă mai mult. Fig. 1.2 ilustrează acest lucru.

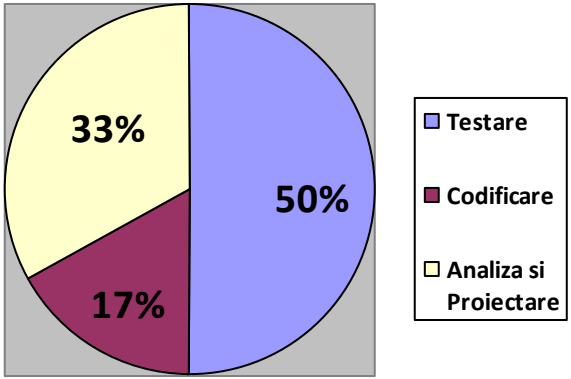

Fig. 1.2 Costul relativ în diferite stadii de dezvoltare ale software-ului

Dacă totuşi erorile sunt o problemă majoră, atunci, când apar ele?

Fig.1.3 arată numărul relativ de erori comise în diferite stadii de evoluție a software-ului.

- Dar problema este puțin mai dificilă şi constă în a stabili cât costă depistarea unei erori.
- Se ştie că o eroare nedescoperită costă mai mult decât ar fi costat ea dacă ar fi fost descoperită şi înlăturată la timp.

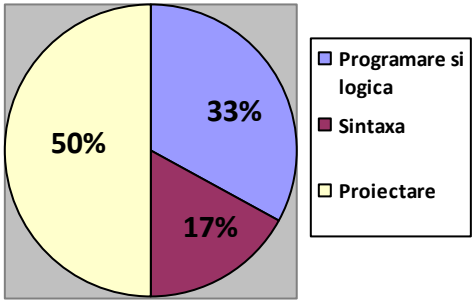

Fig. 1.3. Numărul relativ de erori de-a lungul stadiilor de evoluție ale software-ului

- Erorile de sintaxă sunt descoperite automat de către compilatoare, la prima compilare, şi pot fi corectate cu uşurință.

- Din contră, erorile de proiectare pot fi detectate de abia în faza de testare, ceea ce implică o activitate, câteodată destul de laborioasă, de reproiectare.

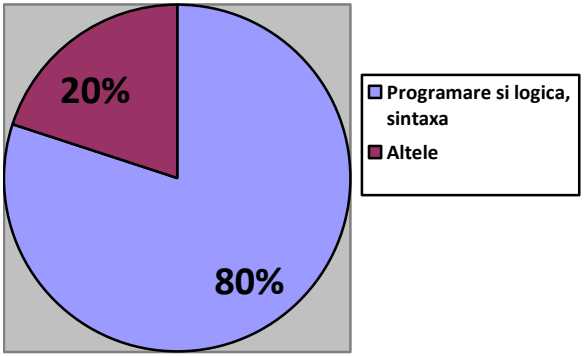

Fig.1.4 Costul relativ pentru depistarea diferitelor tipuri de erori software

1.4 Performanța, portabilitatea şi mentenanța software-ului

Performanța unui program este numită adesea *eficiență*.

Acestă terminologie datează din vremea când viteza hardware-ului era destul de mică iar costul calculatorului relativ mare, ceea ce însemna că efortul trebuia îndreptat spre utilizarea cât mai judicioasă a memoriei şi procesorului central, printr-un program cât mai eficient care conducea în final, la scurtarea timpului de execuție şi deci la o performanță mai bună a sistemului.

În momentul de față, prin performanță se subînțelege:

- un răspuns al sistemului într-o perioadă de timp rezonabilă;
- obținerea unui semnal de control la ieşire (într-un timp rezonabil);

Timpul scurt de execuție şi utilizarea unei memorii mici sunt două cerințe mutual contradictorii.

Mentenanța este termenul folosit pentru orice activitate de întreținere a unei "piese" software, după ce ea a fost dată în exploatare. Sunt două tipuri de mentenanță:

a). *corectivă* - prin care se înlătură erorile apărute în timpul exploatării;

b). *adaptivă* - care apare ca urmare a schimbărilor intervenite în solicitările utilizatorilor, în sistemul de operare sau în limbajele de programare.

O idee despre cât reprezintă activitățile de mentenanță raportate la întregul produs software se poate ilustra în fig. 1.5.

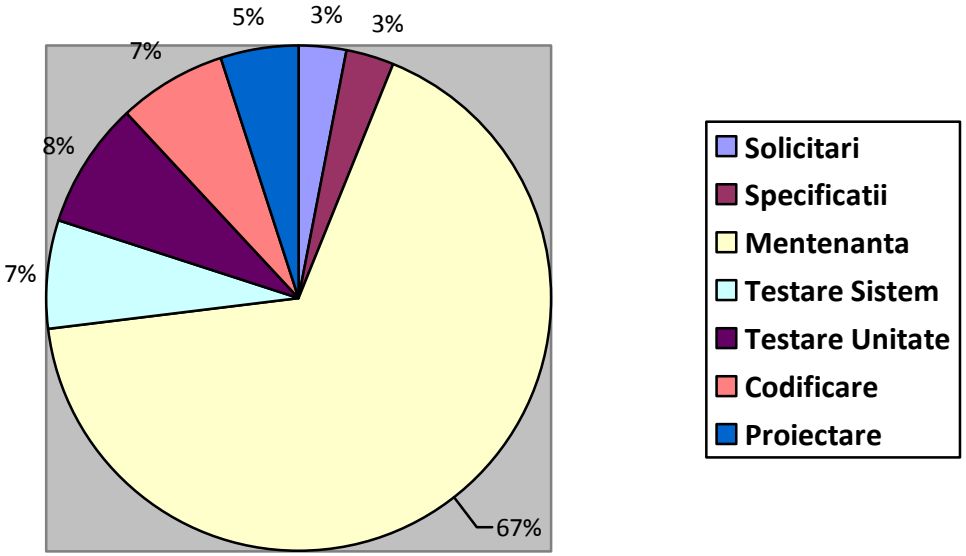

Fig.1.5 Proporția activităților în cadrul realizării unui produs software

1.5 Fiabilitatea

În termeni generali fiabilitatea software reprezintă capacitatea sistemului de a răspunde cerințelor utilizatorului (de a-şi executa misiunea), în conformitate cu specificațiile sale de proiectare.

În mod curent, testarea este principala tehnică care dă certitudinea că software-ul lucrează corect.

Dar problema se complică atunci când trebuie stabilit timpul în care este testată o piesă software pentru a avea certitudinea că este corectă.

1.6 Cerințe pentru ingineria sistemelor de programe

Aşa cum a reieşit din cele expuse până acum există câteva cerințe obligatorii pentru ca un produs informatic, un sistem software să fie utilizat şi anume:

- satisfacerea cât mai completă a cerințelor utilizatorului;
- cost de producție cât mai scăzut;
- performanță ridicată;
- portabilitate;
- cost de întreținere scăzut (mentenanță bună);

- fiabilitate ridicată;

- livrare la timp.

Cele mai multe dintre acestea, sunt în conflict, adică nu pot fi satisfăcute simultan la maximum.

În consecință, în funcție de domeniul de aplicație şi de cerințele problemei se va face o ierarhie a cerințelor acordând prioritate maximă celor care sunt critice pentru sistem.

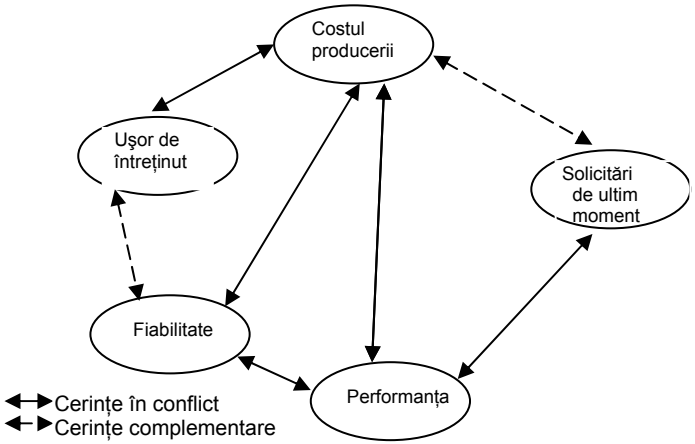

Fig. 1.6 Cerințe complementare şi în conflict în ingineria software

1.7 Teoremă pentru ingineria software

Dacă M(P) este măsura problemei P,
C(P) = costul scrierii programului P
Pt 2 probleme P şi Q, pt care M(P) > M(Q) atinci C(P) > C(Q).

Dacă combinăm cele 2 probleme, P+ Q , atunci

M(P+Q) > M(P) + M(Q).

C(P+Q) > C(P) + C(Q).

Rezultă că este mai uşor de creat două programe mici decât unul mare care cumulează funcțiile ambelor programe.

Pe măsură ce complexitatea creşte, pot apare şi mai multe erori, generate şi de interacțiunea dintre programe.

Fenomenul menționat este caracteristic tuturor domeniilor în care se rezolvă probleme.

Astfel pentru cazul proiectării programelor se poate obține o curbă a erorilor de felul celei prezentate în fig. 1.7.

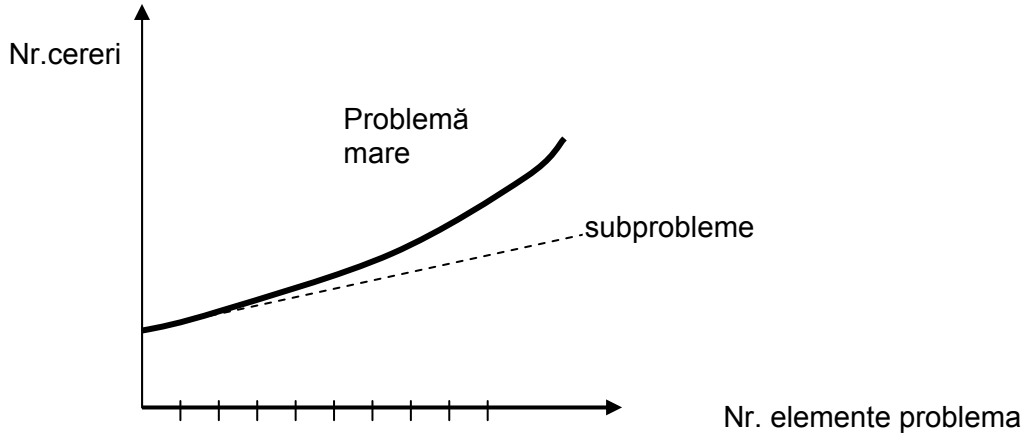

Fig.1.7. Curba erorilor la descompunerea problemei în subprobleme

Acest efect este determinat de limitările ființei umane în prelucrarea informațiilor. Se pare că suntem capabili la un moment dat să prelucrăm simultan şi complet informații referitoare numai la aproximativ şapte obiecte, entități sau concepte distincte.

Peste 7± 2 entități distincte ce trebuie folosite simultan, numărul de erori comise creşte mult mai repede.

în cazul problemelor mari pentru a învinge complexitatea cu un cost mai redus trebuie să se recurgă la descompunerea problemei în module mai mici şi cvasiindependente.

Făcând o substituție în relația de mai sus se obține:

$$C(P) > C(P') + C(P'')$$

numită teorema fundamentală a ingineriei programării, în care P' şi P'' sunt două subprobleme independente prin a căror rezolvare se soluționează la un cost mai scăzut problema P.

Legendă: 1- Cost realizare module
2- Cost realizare interfețe între module.
3- Cost total

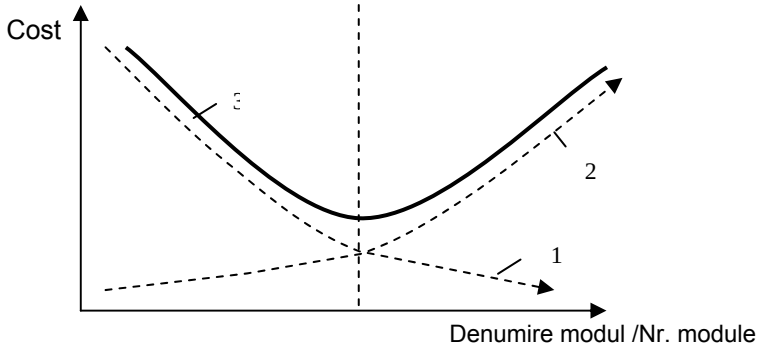

Fig.1.8. Relațiile cost-dimensiune modul şi cost-număr module

Existentă un cost optim de realizare care determină o dimensiune optimă de modul şi un număr optim de module în care proiectul poate fi descompus.

Evident acest optim nu poate fi precizat cu exactitate, ci trebuie căutat în fiecare caz în parte.

Soluțiile la problemele software sunt :

- dezvoltarea sistematică în toate stadiile de dezvoltare a unei piese software ;
- asistența calculatorului pentru dezvoltarea software-ului (limbaje de generația a patra, medii pentru dezvoltare software, proiectare asistată pentru inginerie software – instrumente software (CASE), UML, etc.) ;
- concentrarea efortului pentru depistarea cât mai exactă a dorințelor utilizatorilor ;
- utilizarea specificării formale pentru cerințele sistemului ;
- demonstrația făcută în fața beneficiarilor printr-o versiune timpurie a sistemului (prototipizare) ;
- utilizarea de limbaje de programare noi ;
- creşterea efortului pentru asigurarea că software-ul nu are erori .

Una dintre ideile dominante în abordarea dezvoltării software-ului este dezvoltarea pe baza ciclului de viață sau modelul « cascadă ».

În acest context se împarte dezvoltarea unui produs informatic în paşi independenți, aflați într-o anumită succesiune.

Paşii succesivi sunt:

stabilirea cerințelor

↓

specificarea

↓

proiectarea

↓

implementarea

↓

testarea

↓

operarea şi întreținerea

Specialiştii au idei diferite despre ce trebuie să conțină exact fiecare pas în parte, dar principiile modelului ciclului de viața sunt :

1. Există o serie succesivă de paşi ;
2. Fiecare pas este bine definit ;
3. Fiecare pas creează un produs definit (adesea doar o foaie de hârtie) ;
4. Corectitudinea fiecărui pas trebuie verificată cu grijă.

Utilizarea explicită a unei discipline de conducere a proiectului este factorul cheie în obținerea unei calități software ridicate.

1.9. Documentația sistemelor de programe

O parte extrem de importantă a oricărui produs software este documentația, de care va depinde ulterior utilizarea, întreținerea sistemului şi eventual extensibilitatea.

În mod normal, documentația este elaborată în cu două scopuri.

Primul este de a descrie pachetul software şi modul lui de utilizare.

Această documentație este cunoscută sub numele de **documentație de utilizare** şi este destinată utilizatorului sistemului, având un caracter mai puțin tehnic.

Cel de al doilea scop urmărit de documentație este de a descrie sistemul astfel încât acesta să fie întreținut pe durata celorlalte perioade de viață.

Documentația de acest tip este cunoscută sub numele de **documentație de sistem** şi are în mod inevitabil un caracter mult mai tehnic decât documentația de utilizare.

Astfel documentarea sistemului începe cu dezvoltarea specificațiilor inițiale şi continuă pe toată durata de viață a produsului software.

Documentația va consta în final din toate documentele elaborate în faza de dezvoltare a sistemului, inclusiv specificațiile conform cărora a fost verificat sistemul, diagramele de flux de date şi de relații între entități, dicționarul de date şi schemele de sistem, care reflectă structura sa modulară.

De o mare importanță sunt versiunile sursă ale tuturor programelor din sistem, care trebuie prezentate într-un format lizibil. De aceea, specialiştii încurajează limbajele de nivel înalt, bine proiectate, includerea în programe a comentariilor şi proiectarea modulară, care permite prezentarea fiecărui modul în parte ca un întreg corect.

Faptul că documentarea sistemului trebuie să fie un proces permanent duce la apariția unor conflicte între principiile ingineriei software şi natura umană.

1.10. Dreptul de proprietate şi garanții

Din păcate, problemele referitoare la dreptul de proprietate asupra produselor software nu se încadrează foarte clar în legile existente privind protecția drepturilor de autor şi a brevetelor.

Deşi aceste legi au fost elaborate tocmai pentru a permite fabricantului unui "produs" să-şi facă public produsul, păstrându-şi totodată drepturile de proprietate asupra lui, particularitățile produselor software au pus adesea instanțele în mare dificultate în eforturile lor de aplica şi în acest caz legile generale privind proprietatea intelectuală.

Inițial, drepturile de autor au fost elaborate pentru a proteja operele literare. În acest caz, valoarea nu stă în ideile exprimate, ci mai degrabă în modul în care sunt redate aceste idei. Valoarea unei opere literare rezidă din stil, formă şi nu neapărat din subiect.

În cazul unui produs software, ideea se exprimă în algoritm.

Spre deosebire însă de o poezie sau un roman, valoarea unui produs software nu constă în maniera particulară în care este exprimat algoritmul, ci chiar din algoritm.

În multe cazuri, costurile fazei de dezvoltare a unui pachet software se concentrează tocmai în descoperirea unor algoritmi, şi mai puțin în reprezentarea acestora.

În domeniul drepturilor de proprietate asupra pachetelor software, utilizarea brevetelor se loveşte de mai multe obstacole, dintre care cel mai important este principiul general care statuează că nimeni nu poate fi proprietarul unor fenomene naturale, cum ar fi legile fizicii sau formulele matematice.

În general, instanțele au decis că algoritmii intră şi ei în această categorie, astfel că şi în acest caz legea lasă neprotejată cea mai valoroasă componentă a programului – algoritmul. În plus, obținerea unui brevet este un proces costisitor şi îndelungat, a cărui durată poate fi de ordinul anilor. În acest timp, un produs software se uzează moral, iar până la obținerea brevetului de invenție solicitantul nu are nici un drept de exclusivitate.

Legile dreptului de autor şi cele legate de brevete au rolul de a încuraja atât creşterea numărului de invenții, cât şi schimbul liber de idei, în beneficiul societății.

Atunci când drepturile le sunt protejate, creatorii şi inventatorii sunt mai tentați să îşi aducă realizările la cunoştința publicului.

Adesea, producătorii de software precizează pentru produsele lor un set de garanții prin care îşi stabilesc limitele de responsabilitate.

Acestea iau forme ca: “În nici o situație compania X nu îşi asumă răspunderea pentru pagubele de orice fel provocate prin utilizarea acestui software” sau “Compania Y nu garantează că acest software corespunde exact cerințelor dumneavoastră”.