Aspecte privind optimizarea în domeniul informaticii aplicate în economie

Cătălin Boja
Asistent universitar
Academia de Studii Economice București

Abstract. *The paper describes the main characteristics of the software optimization process. The quality characteristics of software applications are presented. Based on them, there are defined optimum criteria regarding minimization of processing volume and maximization of generality, problem to be solved and precision. There are described methods and techniques used to reach objectives that correspond with minimization of memory space and with minimization of processing volume through modifications of source code. There are taken into consideration the number of optimum criteria analyzed simultaneous and the characteristics of bicriteria optimization are defined. In order to analyze the effects of optimization process and to decide on optimal solution, a method for empirical analysis of results is defined.*

Key words: optimization; software; criteria; optimum; quality.

1. Optimizarea software

Optimizarea software este un domeniu foarte larg al ingineriei software și o etapă importantă în dezvoltarea produselor software.

Se consideră un număr dat de programe care rezolvă aceeași problemă. Dintre ele *programul optim* este acela care dă cea mai bună valoarea pentru un indicator numit criteriu de performanță.

A optimiza un program înseamnă a îmbunătăți performanțele acestuia, cu pretenția că s-a obținut pentru moment o valoare mai bună a criteriului de performanță. Soluția nu este unică întrucât de la o optimizare la alta se ameliorează performanța. În plus optimizarea software are caracter local, referindu-se la un program care se modifică sau la rezultatul comparării unui număr foarte mic de programe între ele, programe care reprezintă versiuni de rezolvare a unei probleme.

Optimizarea programelor înseamnă îmbunătățirea programului. Toate referirile se fac la o mulțime țintă de programe. Se evită referirile de forma: oricare ar fi programul sau pentru orice program, întrucât în activitatea de dezvoltare software se lucrează cu un număr mic de programe.

Optimizarea este precedată de realizarea aplicației informatice și aducerea ei la o formă funcțională lipsită de orice fel de erori. Procesul de optimizare se aplică unei aplicații software funcționale care rezolvă corect și complet problema pentru care a fost realizată.

Optimizarea nu înseamnă corectarea erorilor. Obiectivul acestei etape este constituit de îmbunătățirea caracteristicilor aplicației prin aducerea lor la un nivel optim.

Conceptul de optimizare a aplicațiilor informatice descrie procesul de îmbunătățirea a produsului software prin dezvoltarea de versiuni sau soluții cu un grad crescut pentru nivelul de calitate software. Principalele caracteristici ale procesului sunt:

- continuitatea, care se referă reluarea procesului prin definirea de noi obiective calitative; limitarea temporară a reluării procesului de optimizare este dată de constrângerile materiale, tehnologice și de timp care afectează procesul de dezvoltare sau de atingere a unui nivel calitativ considerat adecvat situației prezente;
- funcționalitatea indică faptul că punctul de start al procesului este constituit de existența unei aplicații software complet funcționale; pe baza acesteia sunt determinate nivelurile calitative de bază în raport cu care sunt evaluate rezultatele procesului de optimizare; de asemenea, soluțiile obținute reprezintă aplicații funcționale care să permită măsurarea nivelurilor criteriilor de optim prin aplicarea metricilor software asociate;
- caracterul empiric descrie modalitatea prin care sunt luate deciziile cu privire la alegerea și implementarea unei soluții optime din mulțimea de soluții posibile; la baza procesului decizional se află seturi de date obiective, obținute prin măsurarea directă a nivelurilor criteriilor de optim pentru fiecare soluție de îmbunătățire în parte.

În ansamblu, activitatea de optimizare include sau este strict legată de procesele de:

- evaluare a aplicației; imaginea obiectivă și completă a aplicației software de optimizat este obținută în urma măsurării directe, prin rularea cu seturi de date diferite și în condiții de lucru reale, a nivelurilor caracteristicilor de calitate analizate;
- testare a versiunilor aplicației; odată definite criteriile de îmbunătățit și soluțiile de atingere a acestor obiective se realizează variante diferite de module, secvențe, algoritmi, tipuri agregate de date care să fie implementate; evaluarea efectelor modificării aplicației se realizează empiric prin testare și măsurare a nivelurilor criteriilor de optim; evaluarea efectelor optimizării prin prisma punctelor de vedere teoretice conduce de cele mai multe ori la îndepărtarea de cea mai bună soluție;
- ierarhizare a versiunilor îmbunătățite; aplicația software reprezintă o construcție complexă cu toate că prin intermediul abordării modulare are asociată o imagine mult simplifcată; lipsa unei metode care să cuantifice efectele produse de optimizare afecteaza obiectivitatea și semnificația deciziei de alegere a celei mai bune versiuni; este necesară măsurarea tuturor factorilor ce influențează criteriul de optim și nivelul de calitate al aplicației;
- alegerea versiunii optime; în urma analizei comparate a metodelor de optimizare se stabilește calea de optimizare a aplicației astfel încât aceasta să fie îmbunătățită pe ansamblu.

Importanța procesului de optimizarea implică stabilirea poziției în ciclul de viață al produsului după ce s-a obținut software funcțional conform figurii 1.

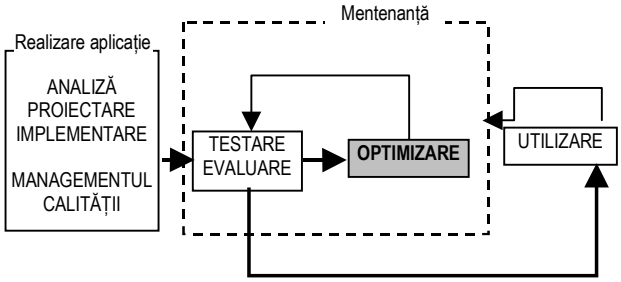

Figura 1. *Ciclul de viață a aplicației*

În etapa de realizare a aplicației, specialiștii ce asigură managementul calității software stabilesc pe baza analizei caracteristicile software considerate esențiale pentru viitorul produs software. Se obțin niveluri estimate și se definesc obiectivele de realizat. Întregul proces de producție urmărește atingerea obiectivului final, realizarea aplicației, precum și asigurarea nivelului de calitate propus.

2. Criterii de optim

Însușirile economice ale produselor software sunt legate în principal de capacitatea acestuia de a genera efecte directe asupra reducerii costurilor de producție în întreprinderi, în gestiunea resurselor, la reducerea ponderii forței de muncă umane în etapele de prelucrare și interpretare a datelor, respectiv a rezultatelor.

Latura economica vizează și modul de fundamentare a deciziilor, atunci când sunt luate în considerare soluții oferite prin implementarea de algoritmi determiniști sau euristici. Aceștia au capacitatea de selectare a variațiilor eficiente dintr-un număr extrem de mare de soluții considerate realizabile.

Însușirile tehnice privesc produsul software ca entitate cu comportament specific bazat pe consumuri de resurse. Produsul este caracterizat prin prisma structurii, funcționalității și a instrumentelor cu care este înzestrat. Din aceste puncte de vedere produsul program este ușor de utilizat, ușor de întreținut, mentenabil, tolerant la erori și elimină într-o proporție ridicată șansele apariției situațiilor care blochează execuția aplicației cu sau fără pierderea datelor, cu consum mic de resurse și nu în ultimul rând cu interfață user friendly.

Caracteristicile sociale se manifestă prin efectele determinate de program în colectivități largi sau în grupuri restrânse de utilizatori, sau prin rolul avut în procesul de instruire sau de rezolvarea de cerințe ale utilizatorilor. Însușirile sociale vizează efortul utilizatorilor pentru a se familiariza cu un program, dar și măsura în care programul răspunde așteptărilor.

Din punct de vedere al producătorului calitatea software este un obiectiv. Din punct de vedere al utilizatorului calitatea este forma concreta de manifestare a produsului software. Producătorul stabilește și ierarhizează caracteristicile de calitate ce definesc programele ca produse software:

- independent de utilizatori: modularitate, mentenabilitate, structurabilitate, testabilitate;
- în funcție de nevoile acestora: fiabilitate, toleranța la erori, generalitate, utilizabilitate.

Nivelurile stabilite ca obiectiv pentru caracteristicile de calitate urmărite sunt rezultatul unui proces interactiv în care intervin resursele umane și tehnice disponibile ale echipei de dezvoltare software, cerințele utilizatorilor, resursele financiare pe care utilizatorul le alocă dezvoltării software, limitările tehnice impuse de generațiile software și hardware la un moment dat și restricțiile de personal ale utilizatorilor.

Calitatea software este un concept complex, iar perfecționarea sa are la bază comunicarea permanentă între producătorul de software și utilizatori, chiar dacă produsul program este proiectat să funcționeze independent de producător.

În contextul existenței unei canal de comunicație, sugestiile și reclamațiile utilizatorilor sunt sursele sigure pentru reproiectarea produselor program și pentru dezvoltarea de noi versiuni. Acestea elimină dezavantajele semnalate sau introduce noi opțiuni de prelucrare.

În fiecare dintre etapele descrise se urmărește obținerea caracteristicilor de calitate la niveluri cât mai apropiate de cele planificate.

Factorii de calitate avuți în vedere sunt:

- eficiență prin economia de timp și economia de resurse;
- funcționalitate prin completitudine, corectitudine, securitate, compatibilitate și interoperabilitate;
- mentenabilitate, ce permite corectarea deficiențelor și dezvoltarea viitoare a produsului;
- portabilitate, ce asigură independența hardware, independența software și reutilizarea aplicației în produse software diferite;
- fiabilitatea prin toleranță la erori și prin gradul în care produsul este operațional având în vedere defectarea sistemului hardware sau software;
- utilizabilitate ce descrie efortul depus de utilizator pentru a înțelege și utiliza produsul.

Pe baza acestora sunt definite criteriile de optim. Acestea devin obiective în cadrul procesului de optimizare deoarece descriu calitativ și cantitativ ceea ce se urmărește a se obține în ciclul de dezvoltare a produsului software:

Minimizarea volumului de prelucrări. Volumul de prelucrări influențează decisiv durata prelucrării datorită faptului că acesta se definește fie ca număr de cicluri mașină, fie ca număr de instrucțiuni executabile. Ciclul mașină și instrucțiunea executabilă se caracterizează prin durate, iar un număr mare de cicluri mașină sau număr mare de instrucțiuni executabile conduce la o durată mare de execuție a programului. A optimiza un program înseamnă a realiza o astfel de construcție care permite obținerea rezultatelor finale într-un timp cât mai scurt. Rezultă că a optimiza un program înseamnă a identifica acele căi a căror consecință este reducerea volumului de prelucrări până la un nivel cât mai scăzut, adică minimizarea volumului de prelucrări.

Atunci când se proiectează soluția pentru o problemă, algoritmii identificați diferă unii de ceilalți prin volumul datelor care se stochează în memorie, volumul rezultatelor intermediare, operații de prelucrare care se execută independent, modalități echivalente, folosirea recursivității și stocarea de rezultate în fișiere.

Volumul de prelucrări ca număr de instrucțiuni executabile se calculează diferențiat pentru secvențele liniare de instrucțiuni, pentru secvențele alternative și pentru secvențele repetitive. Aplicând formulele pentru calculul volumului celor trei tipuri de structuri de control se calculează volumul de prelucrări pentru orice program. Rezultatele sunt utilizate pentru a efectua analize comparate asupra efectelor obținute prin diverse modalități de optimizare.

Maximizarea preciziei. Precizia este un concept nou în raport cu limitările pe care le presupun reprezentările existente în sistemele de calcul. Se lucrează la nivel de bait, la nivel de cuvânt format din doi baiți, la nivel de dublu cuvânt, la nivel de precizie dublă și la nivel de zonă de memorie definită pe zece baiți. Pentru fiecare tip de zonă de memorie se definește un domeniu, se definește o reprezentare și se stabilesc efectele pe care le au operațiile de calcul cu operanzii definiți pe aceste zone de memorie asupra indicatorilor de condiție. Programatorul trebuie să efectueze studii privind domeniile operanzilor pentru a obține rezultate care sunt garantate ca fiind corecte și care nu generează întreruperi de necontrolat în cazul aplicațiilor distribuite. De exemplu, dacă se lucrează cu operanzi definiți pe un bait, aceștia au valori cuprinse între 0 și 255, adică {0, 1, 2, 3, 4, …, 254, 255}.

Înseamnă că programatorul trebuie să aibă grijă să facă astfel de prelucrări încât rezultatul final *RFIN* după evaluarea unei expresii să îndeplinească condițiile *0 d" RFIN d" 255*.

Programul trebuie să includă secvențe de verificare a apartenenței lui la această mulțime.

Pentru programe în care intervin iterații precizia trebuie stabilită astfel încât durata de obținere a soluției să fie corespunzătoare, adică să fie utilizată în fundamentarea deciziei.

În cazul în care rezultatul se obține mult mai târziu decât momentul fundamentării deciziei, rezultatul respectiv este inutilizabil.

Numărul de iterații trebuie să fie acceptabil, iar precizia, de asemenea, trebuie să fie cu un astfel de nivel încât să nu influențeze comportamentul efectiv al sistemului modelat.

În cazul modelelor euristice, precizia are în opoziție eroarea sau gradul de nefolosire de resurse. Impunerea unor condiții privind precizia trebuie să fie rezonabilă fără a determina încercări repetate care au același rezultat, adică imposibilitatea de a obține o îmbunătățire a utilizării resurselor.

Maximizarea preciziei are drept scop realizarea de aplicații informatice care să genereze rezultate finale și intermediare caracterizate de un nivel al preciziei ridicat. Anumite domenii de utilizare a aplicațiilor software impun restricții legate de gradul de precizie minimă a rezultatelor. Urmărirea acestui criteriu influențează negativ restul de criterii prin creșterea spațiului de memorie necesar, a complexității prelucrărilor și a volumului de prelucrări.

Maximizarea dimensiunii problemei de rezolvat. Orice problemă este caracterizată prin:

- date de intrare specificate prin structuri de date în care se stochează; dacă este vorba de fișiere se precizează numărul de articole și lungimea fiecărui articol; dacă se lucrează cu matrice se stabilesc numărul de linii și numărul de coloane, iar dacă se lucrează cu masive unidimensionale se stabilește numărul de componente ale acestora; dimensiunea problemei de rezolvat este dată de numărul de articole din fișier sau de numărul de linii și coloane al matricelor sau de numărul de componente ale masivului unidimensional; dimensiunea problemei de rezolvat se exprimă ca sumă de lungimi de zone de memorie ocupate de datele de intrare ale acesteia;
- pașii algoritmului de prelucrare care presupun evaluarea unor expresii în care apar ca operanzi zonele de memorie în care sunt stocate datele de intrare; algoritmii diferă unii de ceilalți prin rezultatele intermediare care coexistă pe parcursul a mai multor pași; se construiesc algoritmi care necesită un volum restrâns de rezultate intermediare simultane de-a lungul unei succesiuni de pași, care însă conduc la operații de prelucrare suplimentare; există algoritmi care, din contră, determină o viteză de prelucrare ridicată și o gestiune mai bună a iterațiilor, necesitând însă menținerea simultană a unui volum foarte mare de rezultate intermediare stocate în zone de memorie;
- rezultate finale caracterizate prin ocuparea unor zone de memorie a căror lungime depinde de intențiile designerilor de algoritmi de a reutiliza pași ai algoritmului și de a reutiliza rezultate finale ulterior;
- cerințe de mentenanță pentru soluția informatică aleasă concretizate prin definirea unor câmpuri neutilizate la un moment dat, dar care se transformă din rezervă în zone de memorie cu semnificație dedicată atunci când trebuie să se definească noi operanzi, noi rezultate finale și noi rezultate intermediare; mentenanța presupune o nouă abordare a dimensiunii problemei de rezolvat, întrucât este necesar ca de la început să se cunoască echilibrul ce trebuie stabilit în definirea unor limite între care se înscriu parametrii ce definesc datele de intrare ale problemei.

Maximizarea dimensiunii problemei de rezolvat urmărește realizarea unei aplicații software care să includă un set maximizat de funcții legate de obiectivul pentru care a fost dezvoltată. Criteriul stabilește gradul de generalitate a problemei prin includerea de factori de influență considerați, ipoteze de calcul și metode de rezolvare a problemei. Dacă se consideră aplicația Math pentru realizarea de operații aritmetice de bază, atunci maximizarea gradului de generalitate și implicit maximizarea dimensiunii problemei este atinsă prin implementarea unui număr cât mai ridicat de operații și tipuri de date cunoscute. Atingerea acestui obiectiv afectează negativ nivelul complexității aplicației prin creșterea gradului acesteia.

Maximizarea dimensiunii problemei de rezolvat este o tendință pe care programatorul trebuie să o urmărească simultan cu durata de rezolvare a problemei pentru care elaborează software.

Maximizarea generalității. Orice problemă supusă analizei pentru a elabora software trebuie abordată gradat. Se ia o formă mai simplă, după care apar dezvoltări. În acest timp, programatorul capătă experiența necesară abordării corecte a problemei și înțelegerii efortului pe care îl necesită trecerea de la un nivel de complexitate la un nivel de complexitate mai ridicat al problemei.

De exemplu, se consideră problema simplă pentru calculul mediei aritmetice simple. Textul sursă al procedurii este următorul.

Pentru a crește generalitatea se impune elaborarea unei proceduri pentru calculul mediei aritmetice ponderate, *mediap()*. Procedura *media()* nu calculează media aritmetică ponderată, iar procedura *mediap()* nu calculează media aritmetică simplă. Dacă se dorește elaborarea unei proceduri mai generale se procedează la concatenarea celor două.

În cazul în care creșterea generalității se realizează printr-o variabilă de control VB, care are valoarea 1 dacă media aritmetică este ponderată și valoarea 0 dacă media este simplă. Procedura este:

```
float mediap(float x[],int* f,int n,int vb){
      if(vb==0){          f = new int[n];
for(int i=0;i<n;i++)f[i] = 1;}
      return mediap(x,f,n);
}
```

Dacă se abordează de la început probleme în forma cea mai complexă cu grad de generalitate cel mai ridicat, există riscuri să nu poată fi înțeleasă, iar beneficiarii să aștepte foarte mult până la primirea primelor rezultate corecte. De regulă, costurile unui astfel de demers sunt cu mult mai mari decât dacă se parcurg mai multe stadii de la un program care soluționează o problemă mai simplă până la un program asociat unei probleme complexe.

3. Optimizarea spațiului de memorie

Orice aplicație informatică înseamnă:

- baze de date sau fișiere cu date inițiale;
- programe de prelucrare;
- fișiere cu date intermediare sau rezultate finale.

Dacă se notează:

LDate – lungimea bazei de date, exprimată în KB;

LFDI – lungimea fișierului cu date de intrare, exprimată în KB;

LPRG – lungimea programului, exprimată în KB;

LREZ – lungimea fișierului cu rezultate finale, exprimată în KB;

rezultă că lungimea aplicației informatice LAPL este dată de relația

$$LAPR = LDate + LFDI + LPRG + LREZ$$

Programul pentru soluționarea unei probleme conține operanzi care se constituie în definiri de variabile statice sau dinamice și operatori care sunt concretizați prin instrucțiuni executabile incluse în secvențele de programe, proceduri, funcții, module.

Atât zonele de memorie statică alocată operanzilor, LZS_1, LZS_2, …, LZS_{opr}, cât și zonele de memorie alocată dinamică de lungime $LDIM_1$, $LDIM_2$, …, $LDIM_{oprd}$, precum și memoria alocată codului executabil asociat fiecărei proceduri LEX_1, LEX_2, …, LEX_{proc} joacă un rol deosebit de important în asigurarea duratelor de execuție a unui program.

Zonele de memorie alocate static apar în economia aplicației informatice ca având lungimea dată de relația

$$LTS = \sum_{i=1}^{opr} LZS_i$$

Alocarea dinamică presupune lucru pe stivă și transformarea în rezervă care se atribuie urmărind existența unui necesar care să permită existența operanzilor ce ocupă cea mai mare parte din stivă, lungimea totală a zonei alocate dinamic fiind dată de relația

$$LTD = \max_{1 \le i \le oprd} \{LDIM_i\}$$

Structura programului asociat unei aplicații informatice este o arborescență, iar lansarea în execuție cu succes presupune traversarea tuturor nodurilor de la rădăcină până la frunză.

Frunzei îi corespunde finalitatea execuției, concretizată prin afișarea unor rezultate finale.

Drumurile de la rădăcina RAD la frunzele FRZ_1, FRZ_2, …, FRZ_{nrfz} au lungimile, respectiv, $LDRUM_1$, $LDRUM_2$, …, $LDRUM_{nrfrz}$. Memoria ocupată depinde de modulele din program care se încarcă, fiecărui drum fiindu-i asociat un necesar de memorie $LDRM_1$, $LDRM_2$, …, $LDRM_{nrfrz}$. Ca și în cazul memorie alocate dinamic, și aici necesarul de memorie este dat de lungimea cea mai mare a drumului de la rădăcină la frunză, adică:

$$LTDRUM = \max_{1 \le i \le nrfrz} \{LDRUM_i\}$$

Rezultă că necesarul de memorie pentru a executa un program este:

$$LPRG = LTS + LTD + LTDRUM$$

În acest context, necesarul de memorie pentru aplicația informatică este:

$$LAPR = LDate + LFDI + LTS + \max_{1 \le i \le oprd} \{LDIM_i\} + \max_{1 \le i \le nrfrz} \{LDRUM_i\} + LREZ$$

Algoritmii implementați pentru soluționarea unei probleme diferă unii de ceilalți prin faptul că impun niveluri diferite ale memorie utilizate.

Este deosebit de important să se efectueze o analiză detaliată pentru a găsi algoritmii ce minimizează necesarul de memorie pentru stocarea informațiilor și pentru efectuarea prelucrărilor.

Metode utilizate pentru minimizarea spațiului ocupat de datele de intrare sunt bazate pe:

- implementarea tipurilor de date corespunzătoare; alegerea tipului de dată corespunzător se face pe baza unei analize a mulțimii valorilor de intrare și a situațiile prelucrate de aplicația software; de exemplu, implementarea unei variabile care ia valori în mulțimea {0,1} este definită ca fiind de tip *char* ocupând doar un octet;
- analiza și utilizarea structurilor de date adecvate; în cele mai multe cazuri se folosește mai mult spațiu decât este necesar datorită ușurinței implementării lucrului cu anumite structuri de date și datorită lipsei unei analize care să preceadă implementarea și care să indice cel mai potrivit mod de a stoca datele astfel încât spațiul ocupat să fie minim;
- reutilizarea spațiului de memorie prin reinițializări de variabile diferite referite sub același nume;
- alegerea modului de stocare a datelor de intrare; de exemplu, stocarea informațiilor referitoare la mulțimea studenților dintr-o facultatea se realizează prin intermediul fișierelor relative; utilizarea acestei metode prezintă avantaje pentru operațiile de citire a datelor despre un student, însă este ineficientă din punctul de vedere al minimizării spațiului de memorie pentru că multe zone de date nu sunt folosite; dacă numărul matricol al studenților are valoarea maximă 10.000 și dacă poziționarea datelor se face după această valoare distinctă vor exista în fișierul respectiv tot atâtea zone de memorie de dimensiune egală cu spațiul ocupat de datele unui singur student; în cazul care mulțimea studenților activi este cu mult mai mică decât valoarea maximă a numărului matricol vor exista numeroase zone din fișier care nu sunt utilizate;
- compactarea zonei de memorie; se utilizează metode și algoritmi de compresie a datelor; cu toate că se obține un spațiu de memorie ocupat minimizat, procesul are efecte negative asupra volumului de prelucrări din aplicație pentru că presupune o creștere a efortului de prelucrare prin repetarea operațiilor de compresie/decompresie a datelor la fiecare citire/scriere;

- suprapunere de informații pe aceeași zonă de date; limbajul de programare C are implementată facilitatea de a defini structuri de date de tip *union* în care membrii sunt definiți pe aceeași zonă de date; la nivel de date de intrare se analizează setul de valori și se definesc zone din baza de date ce conțin aceeași valoarea pentru mai multe câmpuri.

O altă abordare a problemei (Ivan, Boja, 2006) presupune stocarea datelor sub formă de șiruri de caractere s_{ij} și identificarea de combinații de simboluri ASCII și secvențe în format binar care să minimizeze spațiul ocupat de acestea.

În varianta 1 se consideră un caracter delimitator de sfârșit de șir de caractere „\0" în C++, C# sau în limbaje de asamblare, α; șirul s_{ij} la care se adaugă acest caracter delimitator de sfârșit de șir devine s'_{ij}, $s_{ij} \| \alpha = s'_{ij}$, iar lungimea șirului de caractere devine $\lg(s'_{ij}) = \lg(s_{ij}) + 1$.

Se concatenează șirurile s'_{ij} care descriu elementele c_i din colectivitate. Lungimea șirului pentru a descrie un element din colectivitate este dată de indicatorul $L^i_{art} = \sum_{j=1}^{M} \lg(s_{ij}) + M$.

*Varianta 2 p*resupune efectuarea de conversii sau compresii care au menirea de a reduce lungimea bazei de date.

Cantitățile exprimate în baza de date ca șiruri de caractere sunt supuse conversiei prin reprezentarea întreagă binară sau virgulă mobilă. De exemplu valoarea 1.000, care ocupă 4 octeți prin memorarea sub formă de șir de caractere, este stocată în format binar necesitând un spațiu de 2 baiți. Rezultă lungimea articolului formată din:

- câmpuri cu delimitator (comp1, comp2 și comp5);
- câmpuri cu lungime standard impusă prin conversie pentru comp3 și comp4.

Prin *varianta 3* soluția propusă în variantele anterioare este îmbunătățită prin definirea unei metode care să nu includă utilizarea de delimitatori. Ipoteza de lucru și implementare a acestei variante impune o serie de condiții restrictive care să asigure baza modelului de date utilizat.

Se consideră șablonul Art ce combină într-un singur articol toate datele necesare prelucrării unei singure entități. Forma acestuia este:

articol {tip_1 $câmp_1$; tip_2 $câmp_2$; … tip_s $câmp_s$;}

Pentru a stoca datele astfel încât spațiul ocupat de acestea să fie minimizat, se implementează o modalitate de aranjare a câmpurilor astfel încât oricare două câmpuri adiacente $câmp_i$ și $câmp_{i+1}$ să nu fie de același tip, $tip_i \neq tip_{i+1}$ cu *i = 1..s-1*. Situația permite eliminarea delimitatorilor de câmp, pentru că trecerea de la un tip de dată la altul este anunțată de modificarea formatului intern.

În *varianta 4* se consideră un vocabular V_j care descrie mulțimea valorilor acordare elementelor colectivității C_j.

Mulțimea V_j este descrisă de elementele $V_j = \{v_{j1}, v_{j2}, \ldots, v_{jh}\}$, unde v_{j1}, v_{j2} sunt cuvinte din vocabularul V_j și $\lg(v_{ji})$ descrie lungimea cuvântului v_{ij}.

Oricare șir de caractere s_{ij} reprezentând valoarea componentei *j* a articolului *i* din baza de date este inclusă în vocabularul colectivității, V_j.

Ipoteza care stă la baza implementării acestei variante de lucru presupune existența unui număr foarte mare de date și un vocabular cu un număr limitat de cuvinte. Fiecare cuvânt din vocabular ocupă o anumită poziție. În descrierea articolului se utilizează poziția cuvântului în articol, înlocuind șirul de caractere al valorii printr-un număr ce reprezintă poziția acesteia în vocabularul componentei.

Varianta 5 corespunde efectuării de operații de normalizare și de utilizare a lucrului pe șiruri de baiți. Normalizarea bazei de date se obține prin:

- pentru câmpurile numerice se definește baza de normalizare, B_{norm}, ca fiind valoarea minimă a seriei de date; normalizarea seriei de date sd_i cu i = 1..dim_{serie}, se obține prin înlocuirea valorilor cu forma sd'_i, dată de relația: $sd'_i = sd_i - B_{norm}$;
- pentru câmpurile reprezentate de șiruri de caractere se definește vocabularul valorilor distincte, *Vsd*; normalizarea seriei de date se obține prin înlocuirea fiecărei valori cu poziția acesteia în cadrul vocabularului seriei.

Varianta 6 presupune transformarea alfabetului. Pe un bait se reprezintă alfabetul ASCII de 256 simboluri. În realitate, vocabularele au la bază alfabete cu un număr mult mai restrâns de simboluri. De aceea, este reprezentativ să se ia în considerare operarea pe vocabulare cu alfabete restrânse.

Pentru nume și prenume se iau în considerare literele mari și separatorul spațiu. Alfabetul acesta are 27 simboluri + 1 simbol separator, necesitând 5 biți pentru reprezentare, deoarece $2^4 < 27 < 2^5$.

Astfel numele unei persoane care ocupă L_{NT} = 30 baiți, în noul context va ocupa nu 30 * 8 biți, ci 30 * 5 biți, adică 19 baiți.

Variantele descrise presupun dezvoltarea de rutine diferite care să țină seama de particularitățile fiecărei metode de stocare și care să permită refacerea datelor în memoria internă. Fiecare dintre aceste rutine presupune realizarea unui efort, măsurat în cicluri mașină, asociat procesului de prelucrare și care depinde direct de complexitatea metodei.

Implementarea uneia dintre variante presupune o analiză comparată a eficienței fiecărei metode în funcție de tipologia datelor.

Reducerea spațiului de memorare a datelor externe se obține prin utilizarea metodelor de compresie a datelor. În (Ivan, Verniș, 1998) sunt prezentați algoritmi de compresie și indicatori de evaluare a performanței proceselor de compresie/decompresie.

Compresia revine la a pune în corespondență șiruri de biți pentru alfabete astfel încât să se obțină reducerea lungimii fișierelor.

De asemenea, compresia presupune efectuarea unor transformări asupra șirurilor de simboluri astfel încât șirul rezultat să fie cu o lungime mult mai mică decât lungimea șirului inițial.

4. Optimizarea pe text sursă

În domeniul aplicațiilor software conceptul de calitate este definit prin prisma setului de caracteristici de calitate software. Numeroasele sisteme de caracteristici de calitate software, ISO 90126, ICI, propun categorii de caracteristici ce permit derularea unui proces complex de analiză a calității. La nivel global, prin prisma importanței acordate în procesul de producție, se identifică două seturi de criterii ce descriu dimensiunea aplicației prin intermediul spațiului de memorie:

- extern, ce este ocupat de datele de intrare, datele de ieșire și codul sursă;
- intern, reprezentând necesarul de memorie pentru ca aplicația să ruleze ciclul de prelucrare a datelor de intrare.

și timpul de prelucrare a datelor și afișare a rezultatelor, criteriu direct influențat de viteza de prelucrare.

Optimizarea calității software revine la a îmbunătăți o secvență program în așa fel încât să fie atins un anumit criteriu de optim, ce definește direct nivelul de calitate al aplicației.

Metode de minimizare a volumului de prelucrări urmăresc optimizarea:

- codului sursă;
- algoritmilor implementați; se urmărește găsirea de algoritmi de complexități reduse care să conducă la doritele dorite, dar care să genereze cât mai puține prelucrări; alegerea algoritmului se face în funcție de complexitatea acestuia, indicatorul fiind notat cu *O( )*; de exemplu, dacă se dorește căutarea unei valori într-o mulțime este mult mai eficient să se implementeze operația pe arbori binari de căutare echilibrați și atunci complexitatea este $O(log_2N)$; dacă operația este implementată pe liste, atunci complexitatea algoritmului este *O(N)*, fapt care va genera mult mai multe prelucrări.

Optimizarea pe cod sursă, (Ivan, Boja, 2006) include:

- eliminarea subexpresiilor comune;
- eliminarea instrucțiunilor fără sens și care nu influențează cu nimic determinarea rezultatului final;
- compunerea instrucțiunilor care modifică valoarea unei singure variabile; situația aceasta este prezentă atunci când o variabilă este determinată prin evaluarea unei expresii matematice complexe; descompunerea prelucrărilor în mai multe instrucțiuni de atribuire contribuie la creșterea gradului de înțelegere a codului sursă, însă implică creșterea numărului de cicluri mașină în primul rând datorită salvării repetate a valorii în memorie;
- înlocuirea expresiilor de referire complexe cu unele mai simple;
- eliminarea secvențelor în care apar instrucțiuni cu efecte opuse;
- eliminarea invarianților;
- eliminarea codului mort prin transformarea secvenței;
- reducerea expresiilor indiciale;
- regruparea structurilor de control.

Ideea de a optimiza se concretizează prin modificări în programe care reduc volumul de prelucrări. Volumul de prelucrări, *V*, dat ca număr de iterații se calculează pentru secvența:

```
S = 0;
for(int i=0; i<n; i++ )
      S+= x[i];
```

Volumul de prelucrări V este V = 3n+1 iterații.

Îmbunătățirea secvențelor conduce la micșorarea volumului de instrucțiuni. De exemplu, pentru generarea matricei unitate:

```
for(int i=0;i<n;i++)
      for(int j=0;j<m;j++)
            a[i][j] = 0;
for(i=0;i<n;i++)
      a[i][i] = 1;
```

care are un volum $V_1 = 2n^2 + 3n$.

Secvența îmbunătățită este

```
for(int i=0;i<n;i++)
      for(int j=0;j<m;j++)
            a[i][j] = (i = =j)
```

cu un volum $V_2 = n + n^2$.

Comparând secvențele S_1 și S_2 rezultă că secvența S_2 este mai performantă. Sunt situații în care se urmărește scăderea gradului de complexitate.

Pentru eliminarea subexpresiilor:

$$e = \frac{a\times a + b\times b + c\times c\times c}{a\times a + b\times b + c\times c\times c + d\times d - 1}$$

complexitatea este: $C_1 = 18 \times \log_2 18 + 17 \times \log_2 17$.

Dacă înlocuim elementele subexpresiilor cu $exp = a \times a + b \times b + c \times c \times c$ și $e = \frac{exp}{exp + d\times d - 1}$ atunci complexitatea întregii secvențe este $C_2 = 14 \times \log_2 14 + 12 \times \log_2 12$

Inegalitatea $C_1 > C_2$ conduce la concluzia că prin eliminarea subexpresiilor se obține scăderea complexității.

Există particularități de la limbaj la limbaj de programare în legătură cu posibilitățile de a optimiza.

Cunoașterea unui limbaj de asamblare rezolvă multe dintre problemele de optimizare care diferă de la un limbaj evoluat la celălalt prin găsirea numitorului comun.

Optimizarea algoritmului și a modelului matematic de rezolvare a problemei are ca rezultat reducerea numărului de prelucrări și, implicit, minimizarea numărului de cicluri mașină. De asemenea, modul în care expresia matematică este transpusă în limbajul de programare utilizat are ca efect îmbunătățirea eficienței aplicației, prin reducerea duratei prelucrărilor sau în caz contrar generarea unei soluții slabe.

Optimizarea expresiilor aritmetice prin fragmentare reprezintă o modalitate de a implementa prelucrări matematice care să conducă la un volum cât mai mic de operații prin identificare și reutilizarea valorii unor subexpresii comune.

Se consideră expresia

$$E = (a^2 + b^2 + c^2 + d^2) \times (a^2 + b^2 + c^2 + d^2 - 1) \times (a^2 + b^2 + c^2 + d^2 + g^2) \times (a^2 + b^2 + c^2 + d^2 + g^3) \times (a^2 + b^2 + c^2 + d^2 + g^4)$$

În varianta V_1 expresia E se implementează în forma inițială și se procedează la compilarea ei neoptimizată.

Utilizând informațiile referitoare la numărul de cicluri mașină aferent fiecărei instrucțiuni, precum și un program utilitar de măsurare a volumului de prelucrări, rezultă un program dezvoltat în limbajul de programare C pentru care sunt necesare $NC(V_1) = 88$ cicluri mașină.

Dacă se notează $e_1 = a^2 + b^2 + c^2 + d^2$ și $e_5 = g^2$ și se dezvoltă expresia E folosindu-le se obține construcția corespunzătoare variantei V_2:

$$E = e_1 \times (e_1 - 1) \times (e_1 + e_5) \times (e_1 + e_5 \times g) \times (e_1 + e_5 \times e_5)$$

ceea ce conduce la obținerea unei secvențe a cărei execuție necesită $NC(V_2) = 37$ cicluri mașină.

Prin comparare rezultă că varianta a doua conduce la minimizarea volumului de prleucrări.

Necesitatea efectuării de calcule simbolice este dată de obținerea de forme simplificate ale expresiilor aritmetice. Dacă se consideră expresia $Exp = (a^2 – b^2)/(a + b)$, atunci aceasta necesită un număr de cicluri mașină $NC(Exp) = 52$.

Dacă are loc simplificarea prin calcule simbolice obținem $Exp = a – b$, căreia îi corespunde o secvență de program în C a cărei execuție conduce la $NC(Exp) = 2$ cicluri mașină. Pentru a evidenția efectele operațiilor de optimizare implementate în procesor, la prima rulare a exemplului anterior s-a obținut un efort de prelucrare egal cu 18 cicluri mașină. Prin rularea repetată a aceluiași exemplu s-a ajuns la un nivel constant de 2 cicluri mașină.

Din exemplele anterioare se observă că efectuarea de calcule simbolice conduce la optimizarea prelucrării și obținerea unui nivel scăzut de cicluri mașină în condițiile în care simplifică relațiile prin evitarea efectuării unor calcule suplimentare.

5. Optimizare bicriterială

Luarea în considerare a două criterii reprezintă o etapă importantă pentru optimizarea multicriterială a software.

Cele două criterii alese permit măsurarea riguroasă a efectelor ca modificări ale lungimii zonelor de memorie, respectiv reducerea numărului de cicluri mașină.

Dacă se consideră metodele de reducere a volumului de date MD_1, MD_2, …, MD_n și metode de reducere a volumului de prelucrări MP_1, MP_2, …, MP_m rezultă $m \times n$ variante de combinare a efectelor, pentru fiecare pereche (MD_i, MP_j) măsurându-se costul total CT_{ij}, respectiv costul total corectat, CT'_{ij}.

Dacă pentru aceeași pereche (MD_i, MP_j) se obține relația $CT_{ij} = CT'_{ij}$ rezultă situația stabilă în care costul economic coincide cu costul perceput de utilizatori.

Trebuie realizate măsurări sistematice pentru a identifica situațiile în care metodele nu conduc spre optimizare, rezultând eliminări de linii și de coloane din matricea formată din elementele (MD_i, MP_j). Astfel se îmbunătățește procesul de optimizare a produsului software prin reducerea la minimum a numărului de perechi, (MD_i, MP_j), pentru care se impune realizarea practică a soluției și evaluarea acesteia.

În vederea evaluării efectelor optimizării bicriteriale se analizează corelația dintre volumul de date și volumul de prelucrări.

Crearea bazei de date în formă naturală presupune introducerea de la tastatură a datelor. Volumul de prelucrări, efortul, depind de operatori și se împarte durata în:

- *durata specifică operatorilor*, în care se analizează timpul necesar executării rutinelor de prelucrare a datelor și aducerea lor în formatul specificat; de asemenea, se cuantifică durata necesară obținerii rezultatelor intermediare și finale;
- *durata de scriere pe suport*, care are o pondere însemnată din timpul total de execuție al aplicației și este influențată de caracteristicile componentelor hardware; dacă se analizează o aplicație distribuită atunci se iau în calcul și parametrii transmiterii datelor în rețea.

În continuare, problematica optimizării este legată strict de ceea ce se prelucrează prin program și fără a lua în considerare ceea ce consumă operatorul. Apar și probleme de optimizare, care fac obiectul altei abordări. Se presupune că operatorii care introduc date au randament constante, iar datele introduse sunt complete și corecte. Se construiește programul PP. Datele se introduce o singură dată în baza de date naturală BDN și se utilizează de N ori într-un interval de timp dat, ÄT.

Varianta 0: se creează BDN și se efectuează prelucrările pentru a obține rezultatele. Programul analizat are cinci secvențe asociate operațiilor:

Secv1 – introducerea date operator, V_1;

Secv2 – validare și scriere date pe suport, V_2;

Secv3 – citire date de pe suport, V_3;

Secv4 – prelucrare date, V_4;

Secv5 – afișare rezultate, V_5;

unde V_1, V_2, V_3, V_4 și V_5 reprezintă volumul de prelucrări ca număr de cicluri mașină necesare execuției secvențelor.

Volumul total de prelucrări este $V_{(0)}$ cicluri mașină, iar baza de date ocupă zona de memorie de lungime L_{BDN} octeți.

Varianta 1: se creează baza de date BDN și se produc modificări în program pentru a optimiza programul ca număr de cicluri mașină.

Programul conține:

Secv1 – introducerea date operator, V_1;

Secv2 – validări și scriere date, V_2;

Secv3 – citire date de pe suport, V_3;

$(Secv4)_{modificată}$ – prelucrare date, V'_4;

Secv5 – afișare rezultate, V_5.

Varianta aceasta conține un volum de prelucrări egal cu $V_{(1)}$ cicluri mașină, iar $V_{(0)} > V_{(1)}$. Contribuția la reducerea ciclurilor mașină revine secvenței $Secv4_{modificată}$.

Zona ocupată de datele de intrare rămâne neschimbată, L_{BDN}.

$$V_{(0)} = V_1 + V_2 + k \times (V_3 + V_4 + V_5)$$

unde *k* reprezintă numărul de rulări ale programului pentru obținerea de rapoarte.

$$V_{(1)} = V_1 + V_2 + k \times (V_3 + V'_4 + V_5).$$

Varianta 2: se execută compresia de date și decompresia la fiecare execuție a programului. Se introduc secvențele:

Secv2-0 – pentru compresie înainte de a fi scrise pe suportul de memorare; se obține un spațiu de memorie ocupat de dimensiune redusă;

Secv3-0 – pentru decompresia datelor; după citirea datelor, acestea sunt decompresate pentru a fi prelucrate;

Secv5-0 – pentru compresia datelor înainte de a rescrie baza de date; odată ce au fost obținutele rezultatele dorite, iar sesiunea de lucru se închide, datele obținute sunt compresate și scrise în memorie.

Varianta 2 conduce la $V_{(2)}$ cicluri mașină:

$$V_{(2)} = (V_1 + V_{2\text{-}0} + V_2) + k \times (V_{3\text{-}0} + V_3 + V_4 + V_{5\text{-}0} + V_5).$$

Se reduce lungimea zonei de memorie ocupată de BD la $L_{BDCompresat} < L_{BDN}$, iar volumul de prelucrări este $V_{(2)} > V_{(1)}$.

Dacă $V_2 > V_0 > V_1$ rezultă că efectul minimizării secvenței de prelucrare este depășit cu mult de efectul compresiei de date, obținând un volum de prelucrări aferent lucrului cu baza de date naturală.

Rezultă că efortul optimizării prelucrărilor este atenuat de compresia/decompresia datelor, fără a depăși volumul prelucrării cu baza de date naturală.

În cazul în care se continuă optimizarea secvențelor de prelucrare obținându-se $V_{(3)}$ este important să se obțină inegalitatea $V_{(0)} > V_{(2)} > V_{(3)}$, adică optimizarea prelucrărilor să compenseze efectul compresiei/decompresiei.

Se observă că minimizarea spațiului de memorie e însoțită de introducerea de secvențe de prelucrare și deci de creștere a volumului de prelucrări. Din alt punct de vedere minimizarea prelucrărilor are ca efect scăderea volumului de cicluri mașină.

Efectele sunt în cele mai multe cazuri contradictorii. Cele două criterii nu sunt complementare, iar procesele ce conduc la minimizarea nivelurilor lor se influențează reciproc bazându-se pe măsuri ce scad nivelul unei caracteristici și, implicit, determină creșterea valorii pentru cealaltă caracteristică.

În cazul în care se dorește minimizarea volumului de prelucrări prin controlul redundanței, crește spațiul de memorie rezervat de aplicație în vederea efectuării prelucrărilor.

Existența unui unic loc unde se află o informație impune repetarea operației de citire a acelui articol ori de câte ori este nevoie de informație. Minimizarea redundanței conduce la creșterea volumului de operații de citire.

Pentru minimizarea prelucrărilor, anumite informații trebuie să fie memorate în mai multe locuri, ceea ce conduce la creșterea lungimii zonei de memorare. Mai mult, inițializările presupun efectuarea de operații în toate punctele, nu într-un singur punct, fapt ce generează creșterea volumului de prelucrări.

6. Analiza empirică a efectelor optimizării

O problemă P se construiește în numeroase variante în funcție de limbajul de programare utilizat, tehnici de programare, structuri de date folosite, mod de abordare: distribuit, rețea, standalone.

Pentru fiecare variantă intervin elemente de optimizare. După aceea se pune problema alegerii variantei celei mai potrivite, concept care se identifică în mare măsură cu procesul de optimizare (Ivan, Boja, 2005b).

Există nenumărate modele de analiză, proiectare și implementare a unui produs software. Marea majoritate urmăresc realizarea unei variante funcționale care este îmbunătățită într-un proces repetitiv până când se ating nivelurile stabilite ca obiective finale sau până când raportul calitate/cost este optim.

Fiecare dintre variantele produsului, cu excepția prototipului inițial, se obține în urma testării și evaluării variantei precedente, care este supusă modificărilor la niveluri diferite, cod sursă, framework, algoritmi, pentru a fi optimizată. Îmbunătățirea calității variantelor produsului program este un proces bazat pe optimizare entropică.

Efectele optimizării sunt materializate prin creșterea calității produsului program. Există situații în care se alege implementarea soluției economice în defavoarea soluției

performante, însă aceste decizii privesc părți specifice ale aplicației și sunt luate în considerare dacă nu afectează descrierea produsului în ansamblul său.

Optimizarea și metodele de realizare a acesteia capătă sensuri diferite în realizarea variantelor de program în funcție de obiectivele urmărite. Din perspectiva entității produsului software, optimizarea conduce la obținerea variantei finale, care este și cea mai bună din punct de vedere al producătorului.

Realizarea variantelor de program presupune obținerea de produse software funcționale intermediare care să reprezinte baza de evaluare a efectelor optimizării multicriteriale sau unicriteriale prin măsurarea nivelurilor caracteristicilor analizate. Obținerea nivelurilor agregate permite derularea analizei comparate a variantelor de program care să conducă la alegerea soluției optime.

Se consideră lista formată din șapte variante ale produsului program *Prog*, $Prog_1$, $Prog_2$, $Prog_3$, $Prog_4$, $Prog_5$, $Prog_6$, $Prog_7$ ce implementează în moduri variate metode identice necesare rezolvării problemei *Prob*. Obiectivul analizei constă în determinarea soluției optime, reprezentată de una dintre variante care să rezolve problema *Prob*, astfel încât să fie obținute cele mai bune niveluri pentru caracteristicile de calitate software considerate. Lista de produse software se consideră omogenă prin prisma:

- setului de date de intrare ce are caracteristici comune pentru toate programele;
- problemei de rezolvat;
- platformei hardware și software pe care rulează aplicațiile analizate.

Diferențele dintre programe au la bază modul de implementare în surse al metodei de rezolvare. Algoritmul de prelucrare a datelor diferă prin modalitatea de reprezentare a modelului de date și prin complexitatea metodei de obținere a rezultatelor.

Se consideră setul de caracteristici de calitate software *SQCS*: C_1 – durata unei tranzacții; C_2 – durata de prelucrare; C_3 – grad de încărcare a setului de date de intrare; C_4 – volum de prelucrări.

Setul de caracteristici variază în funcție de obiectivul final al analizei și reprezintă un instrument pus la dispoziția cercetătorului. Prin intermediul caracteristicilor și al metricilor software execuția programului analizat este descrisă de valori numerice ce reprezintă niveluri reale, măsurate. Validarea procesului de evaluare a produsului software evidențiază semnificația măsurătorilor și atestă faptul că niveluri obținute pentru aplicații software diferite sunt comparabile.

Obținerea unui indicator agregat construit pe baza acestor niveluri direct măsurate pornește de la identificarea coeficienților de importanță asociați caracteristicilor analizate. Obținerea variantei optime de program care să rezolve problema *Prob* analizează influențele tuturor caracteristicilor software luate în discuție. În acest sens, coeficientul de importanță ic_k, asociat caracteristicii C_k, descrie ponderea acesteia în caracterizarea produsului software ca fiind optim.

Pe baza analizelor anterioare și prin studiul dependenței dintre caracteristicile C_1, C_2, C_3, C_4 și viteza de obținere a rezultatului se determină valorile coeficienților de importanță.

Fiecare caracteristică software analizată are asociată o funcție de optimizare care să conducă la alegerea nivelului eficient. Identificarea acestei funcții are importanță în compararea directă a nivelurilor caracteristicii pentru două sau mai multe variante de program.

Reprezentând criterii diferite se impune normalizarea valorilor în vederea determinării unui indicator agregat care să descrie varianta de produs software printr-o valoarea direct comparabilă cu valorile agregate asociate restului de variante. Dacă criteriul C_i are funcție de optimizare de tip *Minim*, valoarea normalizată se determină folosind prima relație, în caz contrar se utilizează cea de-a doua expresie:

$$VNC_i^j = \frac{VC_i^{\max} - VC_i^j}{VC_i^{\max} - VC_i^{\min}} \quad (1)$$

$$VNC_i^j = \frac{VC_i^j - VC_i^{\min}}{VC_i^{\max} - VC_i^{\min}} \quad (2)$$

în care:

VNC_i^j – valoarea normalizată a criteriului C_i pentru programul $Prog_j$;

$VC_i^{\max}$ – valoarea maxim înregistrată de criteriul C_i pentru variantele testate;

$VC_i^{\min}$ – valoarea minim înregistrată de criteriul C_i pentru variantele testate;

VC_i^j – valoarea înregistrată de criteriul C_i pentru varianta curentă de program $Prog_j$.

Valoare indicatorului agregat IA este determinată prin ponderarea valorilor normalizate cu valoarea coeficientului de importanță asociat fiecărui criteriu prin relația

$$IA^j = \sum_{i=1}^{4} VNC_i^j \times p_i .$$

Alegerea variantei optime de program este condiționată de o valoare mare a indicatorului agregat.

În cazul în care există două sau mai multe soluții optime, se alege varianta pentru care caracteristica cu coeficientul de importanță cel mai mare are cel mai bun nivel înregistrat.

Rezultatele măsurătorilor sunt utilizate ulterior la evaluarea influenței pe care o are dependența dintre caracteristicile de calitate software asupra obținerii unei soluții optime. Acest lucru se realizează prin analiza suplimentară a relației dintre caracteristici și prin construirea de modele de regresie care să includă acele caracteristici între care există o relație puternică.

Determinarea unui model matematic prin intermediul căruia se determină valoarea indicatorului agregat are la

bază analiza dependenței dintre caracteristicile software de optimizat. Pe baza analizei empirice a unui set omogen de produse program care reprezintă variante de implementare a metodei de prelucrare a datelor sunt identificate ponderile de importanță pentru fiecare criteriu în parte.

Absența studiului empiric și orientarea pur teoretică a cercetării conduc la obținerea de rezultate care sunt lipsite de semnificație și care nu sunt validate de practică.

Optimizarea reprezintă un proces continuu, care se finalizează odată cu decizia de a înlocui aplicația software cu o alta prin reluarea activităților de analiză, proiectare și implementare.

Dezvoltarea permanentă și cu un ritm mult prea rapid a societății informaționale prin lansarea de noi tehnologii și instrumente de realizare a produselor software susține caracterul de continuitate al optimizării, oferind mijloacele necesare atingerii criteriilor de optim.

Abordarea empirică fundamentează reguli practice de dezvoltare software. Utilizarea acestor reguli crește eficiența produselor software.

7. Concluzii

Rezultatele au fost publicate în volumul de lucrări al unor conferințe și comunicări științifice, precum și în paginile unor reviste de specialitate, iar în acest articol se prezintă într-o formă unitară rezultatele cercetărilor întreprinse de autori.

La fiecare zonă se face referire la articolele publicate.

Optimizarea pentru domeniul programării paralele reprezintă o arie de cercetare distinctă, iar rezultatele cele mai convingătoare au la bază algoritmi genetici și metode ale inteligenței artificiale în care definirea rețelelor neuronale și învățarea lor demonstrează caracterul deschis al demersului definit generic *optimizarea aplicațiilor informatice*.

Aplicațiile informatice sunt construcții unitare în care maximizarea gradului de satisfacție pe care îl înregistrează utilizatorul se obține numai maximizând nivelul de corectitudine și nivelul de completitudine al datelor de intrare, maximizând indicatorul agregat asociat calității produsului software și obținând rezultate finale, de asemenea, complete, corecte și absolut necesare utilizatorului.

Optimizarea aplicației informatice revine în primul rând la a agrega trei criterii de performanță ce privesc datele de intrare, software și rezultate finale, într-o singură expresie analitică și, în al doilea rând, a maximiza printr-un proces iterativ de transformare simultană a datelor de intrare, a software și a rezultatelor, indicatorul agregat al calității aplicației.

Bibliografie

Boja, C. Software Multicriterial Optimization, *The Proceedings of the Seventh International Conference of Informatics in Economy,* May 2005, Academy of Economic Studies, Bucharest, *Romania,* Inforec Printing House, ISBN 973-8360-04-8, pp. 1068-1074

Fog, A. (1999). *How to optimize for the Pentium family of microprocessors,* Copyright by Agner Fog

ISO/IEC 9126 International Standard - *Information Technology - Software product evaluation - Quality characteristics and guidelines for their use*, 1991, Geneve, Switzerland

Ivan, I., Boja, C. (2004). *Metode Statistice în Analiza Software*, Editura ASE, București, ISBN 973-594-498-7

Ivan, I., Boja, C. (2006). *Practica optimizării aplicațiilor informatice,* Editura ASE, în curs de apariție

Ivan, I., Boja, C. (2005a). *Collaborative systems components empirical software optimization effect assessment*, The proceedings of The International Workshop on COLLABORATIVE SUPPORT SYSTEMS IN BUSINESS AND EDUCATION, October 28 – 29, 2005, Cluj-Napoca, Romania, Risoprint Printing House, Cluj-Napoca, pp 152 – 189, ISBN 973-651-008-9, Romania

Ivan, I., Boja, C. (2005b). „Empirical Software Optimization", *Revista Informatică Economică,* ISSN 1453 – 1305, vol. IX, nr. 2/2005, Editura Inforec, București, 2005, pp. 43-50

Ivan, I., Verniș, D. (1998). *Compresia de date*, Editura CISON, București

Ivan, I., Boja, C., Felician, A., Web Applications Optimization, *Proceedings of the InfoBUSINESS'2005, „Innovative*

*Applications of Information Technologies in Business and Management" , I*nternational Symposium, October 14-15, 2005, „Al.I.Cuza" University of Iași, Faculty of Economics and Business Administration, Center of Research and Education in Information Systems for Management (CeSINCON), Iasi, Volume edited by Andone I. I., PIM Publishing House, ISBN 973-716-189-0, Iasi, Romania, 2005, pp. 21-30

Ivan, I., Boja, C. „Global Software Optimization"*, Information Systems & Operations Management – ISOM* no. 3 Workshop, Aprilie 20 – 21, 2005c, Romanian American University Master of Economic Informatics, Academy of Economic Studies Master BRIE, Bucharest, ProUniversalis Printing House, ISBN 973-87166-8-3, pp. 205-214

Ivan, I., Coman, S., Balog, Al., Arhire, R. „Tehnici de evaluare a efectelor optimizării de programe", *Revista de statistică*, nr. 3, 7, 1983

Ivan I., Codreanu C. (1996). *Optimizarea programelor assembler,* Raport de cercetare, București

Linux organisation – Applications*, http://www.linux.org/apps/index.html, 2006*

Schaefer M. (1973). A mathematical theory of global program optimization*, Prentice-Hall,*

Vervaet, E., De Cock, M. (2002). *Optimize your Java application's performance: A practical guide for squeezing every drop of performance out of your Java apps, Java Developerworks,* http://www106.ibm.com/developerworks/java/library/j-javaopt/