22. UTILIZAREA STRUCTURILOR DE DATE ÎN CLONAREA INFORMATICĂ

22.1 Conceptul de ortogonalitate a elementelor

Indiferent de forma de existenţă a produselor şi serviciilor în domeniul informaticii, evidenţierea şi gestionarea procesului de clonare reprezintă o activitate importantă întrucât limitarea clonelor creează premise obţinerii de software performant şi de atragere de resurse financiare pentru dezvoltarea producţiei şi calificarea personalului. În acest scop, se impune dezvoltarea de instrumente software care să analizeze arhitecturi hardware, produse software şi baze de date existente într-o arie de utilizare precis delimitată şi să identifice clonele existente. În cazul în care clonele şi procesul de clonare depăşesc limitele definite printr-un cadru legal existent la un moment dat se procedează la efectuarea de corecţii pe toate planurile.

Astfel de produse software sunt realizate deja pentru stabilirea părţilor comune din texte redactate cu procesoarele de tip Word şi pentru identificarea clonelor din programe C++ precum şi depistarea elementelor de proporţionalitate care generează clone în tabele construite prin prelucrări de tip matriceal. Absenţa clonelor este specifică datelor ortogonale.

Două drepte sunt ortogonale dacă unghiul format la intersecţia acestora are cosinusul egal cu zero, cu alte cuvinte cele două drepte sunt perpendiculare. Un ansamblu de drepte este ortogonal dacă dreptele ce-l compun sunt perpendiculare două câte două.

Două plane sunt ortogonale dacă unghiul format la intersecţia lor are cosinusul egal cu valoarea zero, adică cele două plane sunt perpendiculare. Un ansamblu format din mai multe plane este ortogonal dacă planele care îl formează sunt ortogonale două câte două.

Doi vectori sunt ortogonali dacă produsul scalar al acestora este nul.

În continuare o dată este reprezentată uzual ca un număr (125 sau 0 sau –400.72 sau +25.3e-4) sau un şir de caractere (“maşina” sau “125” sau “Cluj-Napoca-2000”).

Extinzând, rezultă că datele D_1 şi D_2 sunt ortogonale **semantic**, în cazul în care conţinutul informaţional al acestora, sensul lor, diferă într-o manieră categorică şi **semiotic**, dacă acestea au o formalizare matematică total diferită.

Ortogonalitatea reprezintă diferenţa dintre două entităţi. Dacă datele sunt complet diferite, atunci ele sunt ortogonale. Forma pe care o iau indicatorii de ortogonalitate este strâns legată de tipul şi complexitatea datelor comparate şi specificul analizei.

Se consideră colectivitatea KOL = $\{ek_1, ek_2, \ldots, ek_{nek}\}$, având *nek* elementele omogene. Elementele colectivităţii sunt descrise prin intermediul a nca caracteristici ale colectivităţii, $Ch_1, Ch_2, \ldots, Ch_{nca}$.

Pentru un element ek_i, ce aparţine colectivităţii KOL, nivelurile caracteristicilor sunt evidenţiate de următorul tuplu:

$$ek_i = (VCh_{i1}, VCh_{i2}, \ldots, VCh_{i,nca}) \qquad (22.1)$$

unde Vch_{ik} este nivelul caracteristicii Ch_k pentru elementul ek_i.

Un element x este ortogonal pe colectivitatea KOL dacă şi numai dacă:

$$< x, ek_i > = 0 \qquad (22.2)$$

oricare ar fi ek_i, element al colectivităţii.

Două seturi de date sunt ortogonale dacă ele nu au nimic în comun. Elementele ek_i şi ek_j sunt ortogonale dacă valorile situate pe poziţiile similare sunt distincte. Identitatea a două seturi de date se defineşte în mod contrar ortogonalităţii. Elementele ek_i şi ek_j sunt identice dacă au acelaşi volum, iar valorile de pe poziţiile similare sunt egale.

Dacă se consideră I indicatorul asociat identităţii şi O indicatorul asociat ortogonalităţii elementelor, atunci:

$$I = 1 - O \qquad (22.3)$$

Ortogonalitatea seturilor de date joacă un rol important în dezvoltarea modelelor cu un nivel ridicat al calităţii. Identificarea unică a unui element dintr-o colectivitate are loc pe baza conceptului de *amprentă*.

Amprenta datelor reprezintă o succesiune de caracteristici obiectiv măsurabile. Caracteristicile individualizează datele şi determină ca în cazul în care nivelurile lor sunt diferite toate pentru date diferite sau sunt identice pentru date identice.

În cazul datelor reprezentate ca entităţi text, amprenta se construieşte prin efectuarea sondajului statistic.

Contribuţia originală la dezvoltarea domeniului D_i este rezultatul activităţii de creaţie. Ea vizează următoarele aspecte [Popa05]:

- introducerea şi definirea de concepte noi care completează sau îmbunătăţesc cadrul conceptual al domeniului;
- dezvoltarea de tehnici şi metode noi de analiză prin introducerea de noi instrumente, sisteme de evaluare, algoritmi de sistematizare şi prelucrare a datelor;
- punerea la punct, completarea sau introducerea de noi metodologii de descriere a ciclului de viaţă pentru entităţile dezvoltate în domeniul D_i;
- elaborarea de studii de caz cu un puternic impact asupra lucrului cu entităţi în domeniul D_i.

Este aproape imposibil să se construiască entităţi cu ortogonalitate de 100%. Plusul de valoare este adus pe baza elementelor existente la care se adaugă, se dezvoltă sau se corectează noi aspecte care prezintă importanţă în domeniu.

Noile entităţi cuprind o foarte mare parte din elementele deja construite. În analiza ortogonalităţii partea cea mai importantă o constituie identificarea şi cuantificarea elementelor cu un ridicat nivel de noutate.

Conceptul de *clonare informaţională* este strâns legat de cel al reutilizării.

Termenul *clonare* îşi are originile în cuvântul grecesc *klon*. Semnificaţia cuvântului este cea de reproducere a sistemelor, produselor sau informaţiei prin constituirea de entităţi separate, dar cu acelaşi conţinut. A clona o entitate ET înseamnă a derula procese prin care se generează noi entităţi care au acelaşi conţinut şi îndeplinesc aceleaşi funcţii.

De exemplu, pentru entităţile software, clonarea presupune obţinerea de noi produse, diferite din punct de vedere sintactic, dar identice din cel al funcţiilor îndeplinite. În acest caz, procesele de clonare presupun efectuarea transformărilor [Ivan03]:

- *înlocuirea cuvintelor cu sinonime*; există operaţii care se implementează prin mai multe modalităţi, utilizând cuvinte diferite; în acest caz, procesele de clonare se referă la funcţiile entităţii software;
- *eliminarea secvenţelor care nu joacă un rol critic*; secvenţele vizează implementarea operaţiilor de normalizare, sistematizare, pregătire apel de subrutine; pentru a asigura caracterul de generalitate pentru entitatea software dezvoltată, este necesar să se efectueze validări; pentru o entitate concretă nu este obligatoriu ca validările să fie implementate;
- *înlocuirea secvenţelor de cod cu secvenţe echivalente*; constă în schimbarea succesiunii paşilor concomitent cu detalierea sau generalizarea secvenţelor de cod asociate; procesele de clonare privesc natura prelucrărilor care, pentru aceleaşi intrări, conduc la obţinerea aceloraşi rezultate;
- *optimizarea subexpresiilor comune*; subexpresiile comune sunt incluse într-o structură căreia i se atribuie un nume generic; în entitatea software, subexpresiile comune sunt înlocuite cu noua structură; se modifică doar modul în care subexpresiile sunt referite;
- *eliminarea secvenţelor redundante*; nu afectează prelucrările specificate în algoritm, ci numai modul de reprezentare a programului sursă ca structură şi conţinut;
- *concatenarea structurilor repetitive*; pentru structurile repetitive având acelaşi număr de iteraţii sau aceeaşi condiţie de oprire sunt reunite expresiile care se execută în cadrul acestora; entitatea software rezultată este identică sub aspectul prelucrărilor şi funcţiilor implementate, dar diferită ca structură;
- *conversia formatului de stocare a datelor*; constă în modificarea structurilor utilizate în stocarea datelor; conţinutul structurilor este identic; în schimb, diferă modul de organizarea externă a datelor;
- *schimbarea structurilor de date utilizate*; constă în modificarea modului de organizare internă a datelor; datele şi prelucrările sunt identice, dar diferă modul de reprezentare a lor;
- *aducerea programelor la o formă comună*; presupune translatarea în plan matematic a caracteristicilor entităţilor software; scopul este de a efectua analize comparative pe entităţi software, prin asigurarea omogenităţii reprezentării acestora;
- *restructurarea programelor*; presupune inventarierea structurilor implementate în entitatea software şi stabilirea succesiunii lor; pentru secvenţe de structuri, se dezvoltă structuri echivalente ca prelucrări efectuate;
- *translatarea software*; constă în transformarea unei entităţi scrisă într-un limbaj de programare într-o entitate dezvoltată cu un alt limbaj.

Cel mai frecvent, procesele de clonare sunt derulate pe baza operaţiei de copiere a conţinutului. Copierea entităţilor reprezintă operaţia de

duplicare a conţinutului acestora. În funcţie de tipului suportului de stocare, există mai multe forme de copiere.

22.2 Indicatori ai diferenţelor măsurate dintre proceduri

Procedurile dintr-un program sunt asemănătoare din următoarele puncte de vedere:

- problema pe care o rezolvă;
- instrumentele folosite;
- funcţiile pe care le realizează;
- tehnicile de proiectare;
- performanţele.

Este preferabil sa se construiască proceduri şi biblioteci de proceduri în scopul reutilizării acestora pentru a creşte productivitatea programatorilor şi pentru a reduce volumul de munca vie încorporată în programele noi care se elaborează. În cazul în care pentru o aceeaşi tipologie de prelucrări se scriu mai multe proceduri, costurile aplicaţiei informatice cresc nejustificat.

Pentru a controla conţinutul textelor sursă din produsele software elaborate şi utilizate în vederea reducerii ponderii componentelor identice realizate accidental este necesar să se evalueze permanent ortogonalitatea procedurilor existente în aplicaţiile aflate în uz curent sau în curs de elaborare.

Astfel, se impune definirea unui sistem de indicatori cu scopul de a evalua asemănarea existenta între proceduri.

Se consideră două proceduri P_i şi P_j. Gradul de asemănare între cele două componente ale colectivităţii formată din produsele software elaborate, se determină ca o medie geometrică între indicatorii ce caracterizează asemănarea procedurilor în funcţie de criteriile considerate:

- dimensiune ca număr de instrucţiuni;
- frecvenţa de apariţie a caracterelor alfabetice;
- frecvenţele de apariţie a cuvintelor unui vocabular dat de către utilizator;
- frecvenţele de apariţie a cuvintelor care formează cele două proceduri;
- frecvenţele de apariţie a diferitelor tipuri de date;
- complexitatea secvenţelor program.

Primul indicator J_1 caracterizează asemănarea procedurilor P_i şi P_j din punctul de vedere al lungimii acestora şi se calculează ca raport între lungimea procedurii P_i şi lungimea procedurii P_j .

$$J_1 = \frac{\min(L_i, L_j)}{\max(L_i, L_j)} \qquad (22.4)$$

unde:

- L_i – lungimea în octeţi a procedurii P_i;
- L_j – lungimea în octeţi a procedurii P_j;

Aceasta se realizează pentru a determina un grad de asemănare relevant a cărui valoare să fie cuprinsă în intervalul [0; 1].

Al doilea indicator J_2 are în vedere surprinderea asemănării dintre cele două proceduri P_i şi P_j în funcţie de frecvenţele de apariţie a caracterelor alfabetice.

$$J_2 = \frac{\sum_{i=1}^{NTC} nca_i}{NTC} \qquad (22.5)$$

unde:
- *NTC* – reprezintă numărul total de caractere alfabetice (mari şi mici);
- nca_i – ia una din următoarele valori:
  - 0, dacă pentru caracterul *i* frecvenţele de apariţie în cele două proceduri sunt diferite;
  - 1, dacă pentru caracterul *i* frecvenţele de apariţie în cele două proceduri sunt identice.

Al treilea indicator J_3 reflectă gradul de asemănare a celor două proceduri în funcţie de un vocabular furnizat de către utilizator.

$$J_3 = \frac{\sum_{i=1}^{NCV} ncv_i}{NCV} \qquad (22.6)$$

unde:
- NCV – numărul de cuvinte al vocabularului pentru care frecvenţele de apariţie în cele două proceduri nu sunt egale şi nule pentru un cuvânt dat;
- ncv_i – ia una din următoarele valori:
  - 0, dacă frecvenţele pentru cuvântul *i* sunt diferite sau egale, dar nule;
  - 1, dacă frecvenţele pentru cuvântul *i* sunt identice şi diferite de zero.

Următorul indicator, J_4, are în vedere criteriul de comparare a elementelor privind frecvenţele de apariţie a totalităţii cuvintelor care compun cele două proceduri.

$$J_4 = \frac{\sum_{i=1}^{NTCV} ncvf_i}{NTCV} \qquad (22.7)$$

unde:
- *NTCV* – numărul total de cuvinte al celor două elemente, cuvinte care sunt distincte;
- $ntcv_i$ – ia una din următoarele valori:
  - 0, dacă frecvenţele pentru cuvântul *i* sunt diferite sau cuvântul i nu se regăseşte în cealaltă procedură;
  - 1, dacă pentru cuvântul *i* frecvenţele sunt identice în cele două proceduri.

Al cincilea indicator, J_5, pune în evidenţă asemănarea a două elemente în funcţie de structura lor.

$$J_5 = \frac{NESC}{NMES} \qquad (22.8)$$

unde:
- *NESC* – numărul de elemente structurale comune celor două proceduri considerate, respectiv P_i şi P_j;
- *NMES* – numărul minim de elemente structurale a celor două proceduri P_i şi P_j.

Următorul indicator, J_6, determină gradul de asemănare a elementelor prin utilizarea unei matrice de precedenţă. Acest grad de asemănare se foloseşte pentru analiza gramaticală a elementelor.

$$J_6 = \frac{\min(NCI_i, NCI_j)}{\max(NCI_i, NCI_j)} \qquad (22.9)$$

unde:
- NCI_i – numărul de cuvinte interschimbabile din procedura P_i;
- NCI_j – numărul de cuvinte interschimbabile din procedura P_j;

Aceasta se realizează pentru a determina un grad de asemănare relevant a cărui valoare să fie cuprinsă în intervalul [0; 1].

Ultimul indicator, J_7, are în vedere determinarea gradului de asemănare a celor două elemente din punctul de vedere al complexităţii acestora.

Gradul de complexitate se determină conform relaţiei:

$$cmplx = n*ln(n) \qquad (22.10)$$

unde n reprezintă numărul de operatori ale elementului.

$$J_7 = \frac{\min(cmplx_i, cmplx_j)}{\max(cmplx_i, cmplx_j)} \qquad (22.11)$$

unde $cmplx_i$ şi $cmplx_j$ reprezintă complexităţile celor două proceduri P_i şi P_j.

Gradul de asemănare a celor două proceduri, J, reprezintă, de fapt, o sinteză a gradelor de asemănare calculate după cele şapte criterii amintite, realizând o omogenizare a acestor indicatori.

$$J = \sqrt[7]{\prod_{i=1}^{7} J_i} \qquad (22.12)$$

În final, pentru un produs software se determină indicatorul agregat J_n care măsoară gradul de asemănare a procedurilor ca medie geometrică între gradele de asemănare dintre toate componentele ce formează un produs software, componentele fiind luate două câte două.

$$J_n = \sqrt[C_n^2]{\prod_{i=1}^{k} J_i^{n_i}} \qquad (22.13)$$

unde:

- C_n^2 – numărul de combinaţii posibile de proceduri luate două câte două;
- J_i – valori indicatori calculate pentru fiecare două proceduri considerate;
- N_i – frecvenţa de apariţie a valorii respectivului indicator.

De exemplu, se consideră procedurile:

P_1 – destinată calculului sumei pătratelor elementelor unui masiv unidimensional al cărui text sursă este:

```
int sume(int x[],int n)
{
      int S;
      S=0;
      for(int j=0; j<n; j++)
            S=S+x[j]*x[j];
      return S;
}
```

P_2 – destinată calculului produsului scalar a elementelor a două masive unidimensionale al cărui text sursă este:

```
int prod_sc(int x[],int y[],int n)
{
      int S;
      S=0;
      for(int j=0;j<n;j++)
            S=S+x[j]*y[j];
      return S;
}
```

P_3 – alegerea minimului dintre trei elemente întregi al cărui text sursă este:

```
int min(int a,int b,int c)
{
      int min;
      min=a;
      if(min>b)
            min=b;
      if(min>c)
            min=c;
      return min;
}
```

P_4 – alegerea maximului dintre trei elemente întregi al cărui text sursă este:

```
int max(int a,int b,int c)
{
	int max;
	max=a;
	if(max<b)
		max=b;
	if(max<c)
		max=c;
	return max;
}
```

Textele sursă ale celor patru proceduri conduc la obţinerea frecvenţelor f_{ij} şi g_{ij} din tabelele nr. 22.1 şi nr. 22.2.

Tabelul nr. 22.1 *Frecvenţele de apariţie a cuvintelor cheie în procedurile P_1, P_2, P_3 şi P_4*

Cuvinte cheie	**Frecvenţe de apariţie**			
	P_1	**P_2**	**P_3**	**P_4**
int	5	6	5	5
}	1	1	1	1
}	1	1	1	1
(	2	2	3	3
)	2	2	3	3
=	3	3	3	3
+	1	1	0	0
-	0	0	0	0
[	3	4	0	0
]	3	4	0	0
;	6	6	5	5
++	1	1	0	0
return	1	1	1	1
--	0	0	0	0
for	1	1	0	0
<	0	0	0	2
>	0	0	2	0
if	0	0	2	2
,	1	2	2	2
*	1	1	0	0

Tabelul nr. 22.2 *Frecvenţele de apariţie a cuvintelor nespecifice din procedurile P_1, P_2, P_3 şi P_4*

Cuvinte nespecifice	**Frecvenţe de apariţie**			
	P_1	**P_2**	**P_3**	**P_4**
sume	1	0	0	0
x	3	2	0	0
n	2	2	0	0
S	5	5	0	0
0	2	2	0	0
prod_sc	0	1	0	0
y	0	2	0	0
min	0	0	8	0
a	0	0	2	2
b	0	0	3	3
c	0	0	3	3
max	0	0	0	8

Se studiază ortogonalitatea între P_1 şi P_2 şi se obţine:

$$I = I_1 * I_2 = 0,8 * 0,7916 = 0,6(3) \quad (22.14)$$

şi între P_3 şi P_4 şi se obţine:

$$I = 0,8(3) * 0,9 = 0,75 \quad (22.15)$$

Din aceste două comparaţii rezultă că gradele de ortogonalitate dintre P_1 şi P_2, respectiv dintre P_3 şi P_4 sunt foarte scăzute.

Se studiază ortogonalitatea între P_1 şi P_3 şi se obţine:

$$I = I_1 * I_2 = 0,68(3) * 0,6916 = 0,4769 \quad (22.16)$$

şi între P_2 şi P_4 şi se obţine:

$$I = 0,75 * 0,6875 = 0,515 \quad (22.17)$$

Din aceste două comparaţii rezultă că gradele de ortogonalitate dintre P_1 şi P_3, respectiv dintre P_2 şi P_4 sunt ridicate.

22.3 Structura software pentru analiza calitativă a noilor elemente incluse în baze de date

Aplicaţiile informatice complexe includ module program intercorelate şi fişiere independente sau interdependente. În cazul folosirii unor SGBD-uri aplicaţiile informatice includ texte sursă de bază, structuri generate şi date care fac obiectul prelucrărilor precum şi date necesare accelerării proceselor de selecţie şi de regăsire.

Este important să se gestioneze redundanţa în bazele de date mai ales atunci când se impune ca textele stocate să fie diferite între ele.

De exemplu, se consideră mulţimea ofertelor pentru obţinerea de fonduri de finanţare şi se impune ca între oferte sa existe diferenţe semnificative, în sensul nedepunerii aceleiaşi oferte pentru două licitaţii sau în sensul depunerii de oferte asemănătoare în cadrul aceluiaşi program.

De asemenea, tezele de doctorat, lucrările de licenţă, articolele şi cărţile prezentate de autori pentru concursuri trebuie să fie diferite unele de celelalte.

În acest sens, este necesară construirea unui produs software care măsoară gradul de ortogonalitate în cadrul unui fişier sau a unei baze de date precum şi dintre fişiere, respectiv, baze de date.

Determinarea printr-o aplicaţie software a valorilor caracteristicilor de calitate şi indicatorilor de ortogonalitate a entităţilor proiect presupun derularea următoarelor activităţi:

- definirea obiectivului aplicaţiei prin stabilirea rezultatelor necesare fundamentării actului decizional;
- stabilirea inputurilor prin studiul sistemului caracteristicilor de calitate şi a modelelor asociate metricilor de ortogonalitate a proiectelor;
- construirea arhitecturii sistemului prin analiza inputurilor şi a corelaţiilor dintre acestea;
- culegerea, normalizarea şi sistematizarea datelor conform cerinţelor metricilor;
- implementarea sistemului de metrici;
- proiectarea interfeţei utilizator în asistarea procesului de stabilire a ortogonalităţii bazei de proiecte;
- testarea sistemului de metrici, urmărindu-se cu precădere comportamentul produsului software în cazurile extreme.

În [Popa02] este prezentată arhitectura şi funcţiile produsului *Cloning Analysis Software – CAS*. Această aplicaţie implementează metricile ale caracteristicilor de bază pentru texte şi date organizate în masive bidimensionale.

Aplicaţia solicită utilizatorului date privind:

- *fişierele de lucru* – trebuie să se specifice numărul şi numele fişierelor care se analizează din punctul de vedere al ortogonalităţii;
- *vocabularul utilizator* – trebuie să se furnizeze dimensiunea şi conţinutul unui vocabular utilizat în modele asociate metricilor; vocabularul utilizator este stocat pe disc într-un fişier;
- *devizele de cheltuieli* – există trei modalităţi în care devizele de cheltuieli asociate proiectelor sunt încărcate în aplicaţie:
  - completarea formularului în care trebuie să se specifice: structura devizului ca număr de linii şi coloane, denumirile categoriilor de cheltuieli şi valorile asociate;
  - completarea formularelor asociate devizelor cu structură predefinită; devizul are o structură impusă, iar utilizatorul trebuie să furnizeze valorile asociate categoriilor de cheltuieli;
  - încărcarea devizelor asociate ofertelor dintr-o bază de proiecte; au o structură impusă, iar valorile sunt completate de ofertanţi; aceste devize sunt stocate în fişiere.
- *fişiere din anexe* – proiectele TIC se concretizează în aplicaţii informatice; se analizează ortogonalitatea aplicaţiilor dezvoltate

prin analiza codului sursă; sunt definite şi calculate metrici ale programelor sursă.

În continuare, se prezintă o serie de metode definite în clasa de obiecte *OrtoMetric* dezvoltate în vedere implementării metricilor de evaluare a similarităţii entităţilor text. Clasa *OrtoMetric* are următoarea definiţie:

```
class OrtoMetric{
public:
      unsigned char NrFis;
      char **F;
      OrtoMetric(unsigned char);
      //[1] Grad de asemanare al fisierelor dupa lungimea lor
      double OrtoLength();
      //[2] Grad de asemanare al fisierelor dupa caracterele
      // alfabetice
      double OrtoCAlfa();
      //[3] Grad de asemanare al fisierelor dupa vocabular definite
      // utilizator
      double OrtoUserVoc();
      //[4] Grad de asemanare al fisierelor dupa vocabularul acestora
      double OrtoFisVoc();
      //[5] Grad de asemanare al fisierelor dupa matricea de
      // precedenta a cuvintelor
      double OrtoMatVoc();
};
```

Metodele implementate utilizează structuri de date externe, respectiv fişiere, pentru determinarea indicatorilor intermediari şi structurarea variabilelor de intrare în modelele asociate metricilor.

Constructorul clasei OrtoMetric are următorul conţinut:

```
OrtoMetric::OrtoMetric(unsigned char n){
      NrFis=n;
      F=new char*[NrFis];
      for(int i=0; i<NrFis; i++)
      {
            char DenFis[30];
            cout<<"Denumirea fisierului "<<i+1<<":";
            cin>>DenFis;
            F[i]=new char[strlen(DenFis)];
            strcpy(F[i],DenFis);
      }
}
```

Se foloseşte o structură de date dinamică pentru stocarea denumirilor de fişiere supuse analizei de similaritate pe baza indicatorilor implementaţi.

Determinarea similarităţii textelor din punctul de vedere al lungimii fişierelor în care acestea sunt stocate este implementată cu ajutorul metodei *double OrtoMetric::OrtoLength()*.

Fişierul *GrPartLg.txt* memorează gradele de asemănare între fişierele lotului introdus prin analiza în pereche. Indicatorul final se determină ca medie geometrică a gradelor de asemănare parţiale.

Codul sursă al metode *OrtoLength()* este:

```
double OrtoMetric::OrtoLength(){
      unsigned char i,j;
      FILE *f,*g,*h;
```

```
      float gr;
      h=fopen("GrPartLg.txt","w+");
      for(i=0; i<NrFis; i++){
                  for(j=0; j<NrFis; j++){
                        f=fopen(F[i],"r");
                        g=fopen(F[j],"r");
                        if(f&&g){
                              fseek(f,0,SEEK_END);
                              fseek(g,0,SEEK_END);

                              if(ftell(f)<ftell(g)){
                                    gr=(float)ftell(f)/ftell(g);
                                    fprintf(h,"%5.3f ",gr);
                              }
                              else{
                                    gr=(float)ftell(g)/ftell(f);
                                    fprintf(h,"%5.3f ",gr);
                              }
                        }
                        else
                              if(!f)
                                    cout<<"Fisierul "<<F[i]<<" nu se
deschide!"<<endl;
                              else
                                    cout<<"Fisierul "<<F[j]<<" nu se
deschide!"<<endl;
                        fclose(f);
                        fclose(g);
                  }
                  fprintf(h,"\n");
      }

      //DETERMINAREA GRADULUI DE ASEMANARE A LOTULUI DE FISIERE
      fseek(h,0,SEEK_SET);
      long double produs=1;
      unsigned int NrComp=0;
      for(i=0; i<NrFis; i++){
            for(j=0; j<NrFis; j++){
                  fscanf(h,"%f",&gr);
                  if(i<j){
                        if(gr){
                              produs*=gr;
                              NrComp++;
                        }
                  }
            }
      }
      fclose(h);
      return pow(produs,1./NrComp);
}
```

Indicatorul de asemănare al fişierelor text din punctul de vedere al caracterelor alfabetice conţinute este implementat cu ajutorul metodei *double OrtoMetric::OrtoCAlfa()* al cărei cod sursă C++ este:

```
double OrtoMetric::OrtoCAlfa(){
      unsigned char i,j,k;
      FILE *f,*g,*h;
      float gr;
      h=fopen("GrPartAlfa.txt","w+");
```

```
      for(i=0; i<NrFis; i++){
            for(j=0; j<NrFis; j++){
                  f=fopen(F[i],"r");
                  g=fopen(F[j],"r");
                  if(f&&g){
                        unsigned     int     contor1m[26],contor2m[26],
contor1M[26],contor2M[26];
                        char c;
                        for(k=0; k<26; k++){
                              contor1m[k]=0;
                              contor2m[k]=0;
                              contor1M[k]=0;
                              contor2M[k]=0;
                        }

                        //citire caractere alfabetice din primul fisier
                        fscanf(f,"%c",&c);
                        while(!feof(f)){
                              if(c==13)
                                    fscanf(f,"\n");
                              else
                              {
                                    if(c>=65&&c<=90)
                                          contor1M[c-'A']++;
                                    else
                                    {
                                          if(c>=97&&c<=122)
                                                contor1m[c-'a']++;
                                    }
                              }
                              fscanf(f,"%c",&c);
                        }

                        //citire caractere alfabetice din al doilea
                        // fisier
                        fscanf(g,"%c",&c);
                        while(!feof(g))
                        {
                              if(c==13)
                                    fscanf(g,"\n");
                              else
                              {
                                    if(c>=65&&c<=90)
                                          contor2M[c-'A']++;
                                    else
                                    {
                                          if(c>=97&&c<=122)
                                                contor2m[c-'a']++;
                                    }
                              }
                              fscanf(g,"%c",&c);
                        }

                        int nca=0;
                        for(k=0; k<26; k++){
                              if(contor1m[k]==contor2m[k])
                                    nca++;
                              if(contor1M[k]==contor2M[k])
                                    nca++;
                        }
```

```
                        float gr;
                        gr=(float)nca/52;
                        fprintf(h,"%5.3f ",gr);
                  }
                  else
                        if(!f)
                              cout<<"Fisierul    "<<F[i]<<"    nu    se
deschide!"<<endl;
                        else
                              cout<<"Fisierul    "<<F[j]<<"    nu    se
deschide!"<<endl;
                  fclose(f);
                  fclose(g);
            }
            fprintf(h,"\n");
      }

      //DETERMINAREA GRADULUI DE ASEMANARE A LOTULUI DE FISIERE
      fseek(h,0,SEEK_SET);
      long double produs=1;
      unsigned int NrComp=0;
      for(i=0; i<NrFis; i++){
            for(j=0; j<NrFis; j++){
                  fscanf(h,"%f",&gr);
                  if(i<j){
                        if(gr){
                              produs*=gr;
                              NrComp++;
                        }
                  }
            }
      }
      fclose(h);
      return pow(produs,1./NrComp);
}
```

Fişierul *GrPartAlfa.txt* este utilizat pentru stocarea gradelor de asemănare parţiale. Fişierele sunt analizate în pereche pentru determinarea gradului de asemănare parţial, iar indicatorul sintetic se obţine prin determinarea mediei geometrice a indicatorilor parţiali.

Metoda *double OrtoMetric::OrtoUserVoc()* implementează indicatorul de asemănare al textelor în raport de un vocabular definit de utilizator. Conţinutul metodei este:

```
double OrtoMetric::OrtoUserVoc(){
      unsigned char i,j,k;
      FILE *f,*g,*h;
      FILE *f1,*g1,*h1;
      float gr;
      struct FA{
            char cuv[30];
            unsigned int fr;
      };

      int opt=1;
      FA str;
      h1=fopen("UserVoc.dat","wb+");
      while(opt){
            cout<<"Introduceti cuvant vocabular:";
            cin>>str.cuv;
```

```
            str.fr=0;
            fwrite(&str,sizeof(FA),1,h1);
            cout<<"Continuati?(0/1) ";
            cin>>opt;
      }

      h=fopen("GrPartUserVoc.txt","w+");
      for(i=0; i<NrFis; i++){
            for(j=0; j<NrFis; j++){
                  f=fopen(F[i],"r");
                  g=fopen(F[j],"r");
                  if(f&&g){
                        f1=fopen("FrecvF.dat","wb+");
                        g1=fopen("FrecvG.dat","wb+");
                        fseek(h1,0,SEEK_SET);
                        fread(&str,sizeof(str),1,h1);
                        int nrc=0;
                        while(!feof(h1)){
                              nrc++;
                              fwrite(&str,sizeof(str),1,f1);
                              fwrite(&str,sizeof(str),1,g1);
                              fread(&str,sizeof(str),1,h1);
                        }

                        //DETERMINARE FRECVENTE DE APARITIE PENTRU
                         // PRIMUL FISIER
                        char CuvF[30];
                        char size=sizeof(FA);
                        fseek(f1,0,SEEK_END);
                        long int dim=ftell(f1)/size;

                        fscanf(f,"%s",CuvF);
                        while(!feof(f)){
                              fseek(f1,0,SEEK_SET);

                              for(k=1;k<=dim;k++){
                                    fread(&str,sizeof(FA),1,f1);
                                    if(strcmp(CuvF,str.cuv)==0){
                                          str.fr++;

                                          fseek(f1,-size,SEEK_CUR);
                                          fwrite(&str,sizeof(FA),1,f1);
                                    }
                              }
                              fscanf(f,"%s",CuvF);
                        }

                        //DETERMINARE FRECVENTE DE APARITIE PENTRU
                         //AL DOILEA FISIER
                        fseek(g1,0,SEEK_END);
                        dim=ftell(g1)/size;

                        fscanf(g,"%s",CuvF);
                        while(!feof(g)){
                              fseek(g1,0,SEEK_SET);
                              for(k=1;k<=dim;k++){
                                    fread(&str,sizeof(FA),1,g1);
                                    if(strcmp(CuvF,str.cuv)==0){
                                          str.fr++;
                                          char size=sizeof(FA);
                                          fseek(g1,-size,SEEK_CUR);
```

```
                                    fwrite(&str,sizeof(FA),1,g1);
                              }
                        }
                        fscanf(g,"%s",CuvF);
                  }

                  int nfa=0;
                  fseek(f1,0,SEEK_SET);
                  fseek(g1,0,SEEK_SET);
                  FA strf,strg;

                  for(k=1;k<=dim;k++){
                        fread(&strf,sizeof(FA),1,f1);
                        fread(&strg,sizeof(FA),1,g1);

                        if(strf.fr==strg.fr){
                              if(strf.fr) nfa++;
                              else nrc--;
                        }
                  }

                  float gr;
                  gr=(float)nfa/nrc;
                  fprintf(h,"%5.3f ",gr);

                  fclose(f1);
                  fclose(g1);
            }
            else
                  if(!f)
                        cout<<"Fisierul    "<<F[i]<<"    nu    se
deschide!"<<endl;
                  else
                        cout<<"Fisierul    "<<F[j]<<"    nu    se
deschide!"<<endl;
            fclose(f);
            fclose(g);
      }
      fprintf(h,"\n");
}
fclose(h1);

//DETERMINAREA GRADULUI DE ASEMANARE A LOTULUI DE FISIERE
fseek(h,0,SEEK_SET);
long double produs=1;
unsigned int NrComp=0;
for(i=0; i<NrFis; i++){
      for(j=0; j<NrFis; j++){
            fscanf(h,"%f",&gr);
            if(i<j){
                  if(gr){
                        produs*=gr;
                        NrComp++;
                  }
            }
      }
}
fclose(h);
return pow(produs,1./NrComp);

}
```

Analiza asemănării fişierelor din lotul de fişiere este realizată şi din punctul de vedere al vocabularelor acestora. Metoda care implementează această metrică de asemănare este *double OrtoMetric::OrtoFisVoc()*.

```
double OrtoMetric::OrtoFisVoc(){
      unsigned char i,j,k;
      FILE *f,*g,*h;
      FILE *f1,*g1;
      float gr;
      struct FA{
            char cuv[30];
            unsigned int fr;
      };

      h=fopen("GrPartFisVoc.txt","w+");
      for(i=0; i<NrFis; i++){
            for(j=0; j<NrFis; j++){
                  f=fopen(F[i],"r");
                  g=fopen(F[j],"r");
                  if(f&&g){
                        f1=fopen("FrecvF.dat","wb+");
                        g1=fopen("FrecvG.dat","wb+");
                        FA str;

                        //DETERMINARE FRECVENTE DE APARITIE PENTRU
                        // PRIMUL FISIER
                        char CuvF[30];
                        char size=sizeof(FA);
                        long int dim;

                        fscanf(f,"%s",CuvF);
                        while(!feof(f)){
                              fseek(f1,0,SEEK_END);
                              dim=ftell(f1)/size;

                              fseek(f1,0,SEEK_SET);
                              char vb=0;
                              for(k=1;k<=dim&&(!vb);k++){
                                    fread(&str,sizeof(FA),1,f1);
                                    if(strcmp(CuvF,str.cuv)==0){
                                          str.fr++;

                                          fseek(f1,-size,SEEK_CUR);
                                          fwrite(&str,sizeof(FA),1,f1);
                                          vb=1;
                                    }
                              }
                              if(!vb){
                                    str.fr=1;
                                    strcpy(str.cuv,CuvF);
                                    fseek(f1,0,SEEK_END);
                                    fwrite(&str,sizeof(str),1,f1);
                              }
                              fscanf(f,"%s",CuvF);
                        }

                        //DETERMINARE FRECVENTE DE APARITIE PENTRU
                        //AL DOILEA FISIER
                        fscanf(g,"%s",CuvF);
```

```
                        while(!feof(g)){
                              fseek(g1,0,SEEK_END);
                              dim=ftell(g1)/size;

                              fseek(g1,0,SEEK_SET);
                              char vb=0;
                              for(k=1;k<=dim&&(!vb);k++){
                                    fread(&str,sizeof(FA),1,g1);
                                    if(strcmp(CuvF,str.cuv)==0){
                                          str.fr++;
                                          char size=sizeof(FA);
                                          fseek(g1,-size,SEEK_CUR);
                                          fwrite(&str,sizeof(FA),1,g1);
                                          vb=1;
                                    }
                              }
                              if(!vb){
                                    str.fr=1;
                                    strcpy(str.cuv,CuvF);
                                    fseek(g1,0,SEEK_END);
                                    fwrite(&str,sizeof(str),1,g1);
                              }
                              fscanf(g,"%s",CuvF);
                        }

                        unsigned int nci=0,ncvf=0;
                        fseek(f1,0,SEEK_END);
                        dim=ftell(f1)/size;
                        fseek(g1,0,SEEK_END);
                        long int dim1=ftell(g1)/size;
                        FA strf,strg;
                        fseek(f1,0,SEEK_SET);

                        for(k=1;k<=dim;k++){
                              fread(&strf,sizeof(FA),1,f1);

                              fseek(g1,0,SEEK_SET);
                              for(int k1=1;k1<=dim1;k1++){
                                    fread(&strg,sizeof(FA),1,g1);

                                    if(strcmp(strf.cuv,strg.cuv)==0){
                                          nci++;
                                          if(strf.fr==strg.fr) ncvf++;
                                    }
                              }
                        }

                        float gr;
                        gr=(float)ncvf/(dim+dim1-nci);
                        fprintf(h,"%5.3f ",gr);

                        fclose(f1);
                        fclose(g1);
                  }
                  else
                        if(!f)
                              cout<<"Fisierul    "<<F[i]<<"    nu    se
deschide!"<<endl;
                        else
                              cout<<"Fisierul    "<<F[j]<<"    nu    se
deschide!"<<endl;
```

```
                fclose(f);
                fclose(g);
            }
            fprintf(h,"\n");
      }

      //DETERMINAREA GRADULUI DE ASEMANARE A LOTULUI DE FISIERE
      fseek(h,0,SEEK_SET);
      long double produs=1;
      unsigned int NrComp=0;
      for(i=0; i<NrFis; i++){
            for(j=0; j<NrFis; j++){
                fscanf(h,"%f",&gr);
                if(i<j){
                      if(gr){
                            produs*=gr;
                            NrComp++;
                      }
                }
            }
      }
      fclose(h);
      return pow(produs,1./NrComp);

}
```

Metoda double OrtoMetric::OrtoMatVoc() implementează metrica de asemănare a fişierelor ce conţin texte privind poziţia cuvintelor în cadrul acestora. Codul sursă C++ al metodei este descris în continuare:

```
double OrtoMetric::OrtoMatVoc(){
      unsigned char i,j,k,l;
      FILE *f,*g,*h;
      FILE *f1,*g1,*f2,*g2;
      float gr;
      struct FA{
            char cuv[30];
            unsigned int *fr;
      };

      h=fopen("GrPartMatVoc.txt","w+");
      for(i=0; i<NrFis; i++){
            for(j=0; j<NrFis; j++){
                f=fopen(F[i],"r");
                g=fopen(F[j],"r");
                if(f&&g){
                      f1=fopen("VocF.txt","w+");
                      g1=fopen("VocG.txt","w+");

                      f2=fopen("CuvMatF.txt","w+");
                      g2=fopen("CuvMatG.txt","w+");
                      FA str;

                      //CONSTRUIRE VOCABULAR PRIMUL FISIER
                      char CuvF[30],CuvD[30];
                      unsigned int nrcF=0;
                      char size=sizeof(CuvF);

                      fscanf(f,"%s",CuvF);
                      while(!feof(f)){
```

```
            fseek(f1,0,SEEK_SET);
            char vb=0;
            fscanf(f1,"%s",str.cuv);
            while(!feof(f1)){

                  if(strcmp(CuvF,str.cuv)==0)   vb=1;
                  fscanf(f1,"%s",str.cuv);
            }
            if(!vb){
                  fprintf(f1,"%s ",CuvF);
                  nrcF++;
            }
            fscanf(f,"%s",CuvF);
      }

      str.fr=new unsigned int[nrcF];
      fseek(f1,0,SEEK_SET);

      fscanf(f1,"%s",str.cuv);
      unsigned int nri=0;
      while(!feof(f1)){
            nri++;
            for(l=0;l<nrcF;l++) str.fr[l]=0;

            fseek(f,0,SEEK_SET);
            fscanf(f,"%s",CuvF);
            while(!feof(f)){
            if(strcmp(str.cuv,CuvF)==0){
            fscanf(f,"%s",CuvF);
            if(!feof(f)){
                  fseek(f1,0,SEEK_SET);
                  for(unsigned int l=0;l<nrcF;l++){
                        fscanf(f1,"%s",CuvD);
                  if(strcmp(CuvF,CuvD)==0)
                        str.fr[l]++;
                  }
            }
            }
            else fscanf(f,"%s",CuvF);
            }

            //SCRIEREA IN FISIERUL TEXT A LINIEI k CU
             // FRECVENTELE PERECHILOR
            fprintf(f2,"%s ",str.cuv);
            for(int unsigned l=0;l<nrcF;l++)
                  fprintf(f2,"%i ",str.fr[l]);
            fprintf(f2,"\n");

            fseek(f1,0,SEEK_SET);
            for(l=0;l<=nri;l++)
                   fscanf(f1,"%s",str.cuv);
      }

      //CONSTRUIRE VOCABULAR AL DOILEA FISIER
      unsigned int nrcG=0;

      fscanf(g,"%s",CuvF);
      while(!feof(g)){

            fseek(g1,0,SEEK_SET);
```

```
                        char vb=0;
                        fscanf(g1,"%s",str.cuv);
                        while(!feof(g1)){
                              if(strcmp(CuvF,str.cuv)==0)   vb=1;
                              fscanf(g1,"%s",str.cuv);
                        }
                        if(!vb){
                              fprintf(g1,"%s ",CuvF);
                              nrcG++;
                        }
                        fscanf(g,"%s",CuvF);
                  }

                  str.fr=new unsigned int[nrcG];
                  fseek(g1,0,SEEK_SET);

                  fscanf(g1,"%s",str.cuv);
                  nri=0;
                  while(!feof(g1)){
                        nri++;
                        for(l=0;l<nrcF;l++) str.fr[l]=0;

                        fseek(g,0,SEEK_SET);
                        fscanf(g,"%s",CuvF);
                        while(!feof(g)){
                        if(strcmp(str.cuv,CuvF)==0){
                        fscanf(g,"%s",CuvF);
                        if(!feof(g)){
                              fseek(g1,0,SEEK_SET);
                              for(unsigned int l=0;l<nrcF;l++){
                              fscanf(g1,"%s",CuvD);
                              if(strcmp(CuvF,CuvD)==0)
                                    str.fr[l]++;
                              }
                        }
                        }
                              else fscanf(g,"%s",CuvF);
                        }

                        //SCRIEREA IN FISIERUL TEXT A LINIEI k CU
                         // FRECVENTELE PERECHILOR
                        fprintf(g2,"%s ",str.cuv);
                        for(unsigned int l=0;l<nrcG;l++)
                              fprintf(g2,"%i ",str.fr[l]);
                        fprintf(g2,"\n");

                        fseek(g1,0,SEEK_SET);
                        for(l=0;l<=nri;l++)
                            fscanf(g1,"%s",str.cuv);
                  }


                  //DETERMINARE COMPONENTE GRAD DE ASEMANARE
                   // PENTRU FISIERELE F SI G
                  unsigned int nfm=0,ntcc=0;
                  FA str1;
                  str.fr=new unsigned int[nrcF];
                  str1.fr=new unsigned int[nrcG];
                  unsigned int frc;

                  for(k=0;k<nrcF;k++){
```

```
fseek(f2,0,SEEK_SET);
for(l=0;l<k;l++){
      fscanf(f2,"%s",CuvF);
      for(unsigned int m=0;m<nrcG;m++){
            fscanf(f2,"%i",&frc);
            str1.fr[m]=frc;
      }
}
fscanf(f2,"%s",CuvF);
strcpy(str.cuv,CuvF);
for(l=0;l<nrcF;l++){
      fscanf(f2,"%i",&frc);
      str.fr[l]=frc;
}

fseek(g2,0,SEEK_SET);
fscanf(g2,"%s",CuvD);
char vb=0;
while(!feof(g2)&&!vb){

      if(strcmp(CuvF,CuvD)==0){
            ntcc++;
            strcpy(str1.cuv,CuvD);
            for(l=0;l<nrcG;l++){
                  fscanf(g2,"%i",&frc);
                  str1.fr[l]=frc;
            }
            vb=1;
      }
      else{
            for(l=0;l<nrcG;l++){
                  fscanf(g2,"%i",&frc);
                  str1.fr[l]=frc;
            }
      }
      fscanf(g2,"%s",CuvD);
}

if(vb){
      unsigned int pozF=0;

      fseek(f2,0,SEEK_SET);
      fscanf(f2,"%s",CuvF);
      while(!feof(f2)){
            fseek(g2,0,SEEK_SET);
            fscanf(g2,"%s",CuvD);
            unsigned int pozG=0;
            while(!feof(g2)){
            if(strcmp(CuvF,CuvD)==0){
           if(str.fr[pozF]==str1.fr[pozG])
                nfm++;
                  }
                  for(l=0;l<nrcG;l++){
                  fscanf(g2,"%i",&frc);
                        str1.fr[l]=frc;
                  }
                  fscanf(g2,"%s",CuvD);
                  pozG++;
            }
            for(l=0;l<nrcG;l++){
                  fscanf(f2,"%i",&frc);
```

```
                                        str.fr[l]=frc;
                                    }
                                    fscanf(f2,"%s",CuvF);
                                    pozF++;
                                }
                            }
                        }

                        //DETERMINAREA GRADULUI DE ASEMANARE
                        float gr;
                        gr=(float)nfm/(ntcc*ntcc);
                        fprintf(h,"%5.3f ",gr);

                        fclose(f1);
                        fclose(g1);

                        fclose(f2);
                        fclose(g2);
                    }
                    else
                        if(!f)
                            cout<<"Fisierul    "<<F[i]<<"    nu    se
deschide!"<<endl;
                        else
                            cout<<"Fisierul    "<<F[j]<<"    nu    se
deschide!"<<endl;
                    fclose(f);
                    fclose(g);
                }
                fprintf(h,"\n");
            }

            //DETERMINAREA GRADULUI DE ASEMANARE A LOTULUI DE FISIERE
            fseek(h,0,SEEK_SET);
            long double produs=1;
            unsigned int NrComp=0;
            for(i=0; i<NrFis; i++){
                for(j=0; j<NrFis; j++){
                    fscanf(h,"%f",&gr);
                    if(i<j){
                        produs*=gr;
                        NrComp++;
                    }
                }
            }
            fclose(h);
            return pow(produs,1./NrComp);
}
```

Pentru reprezentare internă a textului entităţilor software, se propun mai multe variante, [Sand05]:

- *reprezentarea cu ajutorul a doi vectori* – pentru fiecare entitate se asociază un vector VD, cu *nvd* elemente, care memorează toate cuvintele distincte din şi un vector VF**,** cu *nvd* elemente, care reţine pe fiecare poziţie k, k = 1, 2, …, nvd, numărul de apariţii ale fiecărui cuvânt din VD;
- *reprezentare prin utilizarea de pointeri* – se utilizează o listă simplu înlănţuită care are ca înregistrare în nod perechea (vd_i,

vf_i); unde vd_i este un element din vectorul VD, iar vf_i este un element din vectorul VF;

- *reprezentarea prin arbore de căutare* – cheia din arbore este dată de un cuvânt; ca informaţie utilă, nodul are ataşat numărul de apariţii ale cuvântului în titlul proiectului;
- *reprezentarea prin tabele gestionate de un SGBD* – se defineşte o tabelă destinată memorării textelor entităţilor software, în care cheia primară este un cod unic de identificare asociat fiecărei entităţi; aceasta devine cheie externă într-o tabelă, care conţine coloanele: *nr_proiect*, *cuvânt* şi *apariţii*; în această tabelă se stochează cuvintele, împreună cu numărul de apariţii.

Analiza ortogonalităţii programelor sursă din anexele proiectelor din baza de proiecte presupune construirea unei matrice ale indicatorilor de ortogonalitate agregaţi pentru perechile de programe sursă (PRG_i, PRG_j). Indicatorul agregat de ortogonalitate asociat perechii (PRG_i, PRG_j) se obţine prin determinarea ortogonalităţii următoarelor metrici primare, [Popa02]:

- *lungimea programelor* – presupune determinarea mărimii pe disc a fişierelor în care sunt stocate codurile sursă ale aplicaţiilor din baza de proiecte;
- *frecvenţele de apariţie a caracterelor alfabetice* – constă în încărcarea elementelor masivelor unidimensionale cu numărul de apariţii al caracterelor utilizate în construirea programelor;
- *vocabularul utilizator* – utilizatorul introduce cuvinte, rezervate sau definite, pe care le urmăreşte în programele analizate din punctul de vedere al frecvenţelor de apariţie;
- *vocabularul programelor* – se identifică cuvintele diferite din textele sursă şi se determină frecvenţa lor de apariţie;
- *vocabularul comun* – constă în identificarea cuvintelor comune din vocabularele textelor sursă şi determinarea frecvenţelor de apariţie;
- *structura entităţilor* – constă în împărţirea în paragrafe a programelor sursă şi determinarea de frecvenţe de apariţie pentru fiecare paragraf;
- *variabilele definite* – pe baza sintaxei limbajului se identifică variabilele utilizate şi se determină frecvenţele de apariţie;
- *matrice de precedenţă a variabilelor* – presupune specificarea ordinii de folosire a variabilelor printr-o structură de tip masiv bidimensional; compararea valorilor din matricea de precedenţă conduce la determinarea indicatorului de ortogonalitate corespunzător acestui criteriu;
- *poziţia variabilelor* – presupune identificarea variabilelor şi reţinerea poziţiilor acestora în funcţie de instrucţiunile asociate în utilizare.

Modul în care este proiectat permite includerea de noi indicatori şi implementarea de noi mecanisme de analiză comparata a datelor.