ORTOGONALITATEA PROGRAMELOR C++ SOFTWARE MINING

Ion Ivan
ionivan@ase.ro

Daniel Milodin
daniel.milodin@ase.ro

Sorin-Nicolae Dumitru
sorin.dumitru@yahoo.com

Academia de Studii Economice Bucureşti

Rezumat: Se prezintă conceptul de ortogonalitate pentru entităţi text. Se construiesc indicatori de măsurare a ortogonalităţii. Se utilizează indicatori pentru măsurarea ortogonalităţii programelor scrise în limbajul C++. Se analizează influenţa măsurării ortogonalităţii asupra creşterii calităţii programelor C++. Rezultatele sunt bazate pe utilizarea de software destinat studiului ortogonalităţii produselor - program scrise în diferite limbaje de programare.

Cuvinte cheie: entitate text, ortogonalitate.

1. Ortogonalitatea entităţilor text

Alfabetul este o mulţime finită, formată din simbolurile a_1, a_2, a_3, a_4, ... a_{ns}, unde *ns* este numărul de simboluri.

Cu ajutorul simbolurilor alfabetului, se construiesc vocabularele.

Asocierea dintre o serie de simboluri ale alfabetului şi un element al lumii reale conduce la crearea de termeni reprezentativi şi la formarea de vocabulare.

Vocabularul conţine asocieri de simboluri care îndeplinesc condiţiile:

- sunt uşor de reprodus;
- sunt diferite unele de celelalte;
- au simetrii;
- sunt în corespondenţă biunivocă cu o mulţime de sunete, elemente din lumea reală;
- permite construirea de şiruri prin concatenare.

Vocabularul este construit cu ajutorul simbolurilor alfabetului, iar termenii conţinuţi de un vocabular sunt diferiţi.

Vocabularele sunt utilizate pentru a reprezenta informaţia în formă vizuală, fonică, tactilă sau mentală.

Sunetele reprezintă un vocabular distinct. Sunetul se transmite sub forma oscilaţiilor în mediul în care s-a produs fiind perceput de aparatul auditiv în intervale de frecvenţe.

De asemenea, limitând aria de reprezentativitate a vocabularului la un domeniu strict, acesta este construit pe baza termenilor de referinţă din respectivul domeniu. Spre exemplu, în ceea ce priveşte coloristica, se pleacă de la trei culori de bază roşu, verde şi albastru, prin combinarea lor obţinându-se alte culori care îmbogăţesc atât domeniul respectiv, cât şi vocabularul acestuia.

Domeniul muzical are, de asemenea, la bază un număr de note muzicale, iar prin compunerea de variaţiuni pe baza acestora rezultă noi game de note muzicale şi noi sunete de reprezentat.

În ceea ce priveşte vocabularul unui limbaj de programare, acesta este format din litere mari, litere mici, cifre, operatori şi separatori. Toate simbolurile conţinute de un limbaj de programare se regăsesc în cadrul caracterelor ASCII.

Cu ajutorul simbolurilor sunt reprezentate noţiuni sau obiecte. Literele sunt simbolurile cu ajutorul cărora se construiesc alfabetele. O condiţie esenţială pentru ca un simbol să aparţină unui alfabet este aceea de a respecta anumite reguli stricte, care-l poziţionează în mod unic faţă de simbolul anterior şi faţă de simbolul următor.

Între litere şi sunete există o corespondenţă. Prin intermediul literelor, această corespondenţă este percepută vizual, fonic sau tactil.

În cadrul limbajului, există simboluri care nu sunt incluse în alfabet. În această categorie intră separatorii şi simbolurile speciale. Prin convenţie, separatorul cu cea mai mare frecvenţă de apariţie, utilizat între grupurile de litere, este *„blank"*. În categoria simbolurilor speciale, intră cele utilizate în reprezentarea informaţiei şi a operaţiilor în diverse ramuri ale ştiinţei: matematică, fizică, chimie, informatică etc.

Din setul de caractere ASCII, sunt extrase următoarele alfabete:

- alfabetul format din literele mici: $SLm = a|b|c|...|z;$
- alfabetul format din literele mari: $SLM = A|B|C|...|Z;$
- alfabetul format din cifrele arabe: $SCA = 0|1|2|...|9;$
- alfabetul format din mulțimea separatorilor: $SMS = \#|\ |;|,|.|?|...|]$.

Cu ajutorul alfabetelor definite, sunt descrise cuvintele limbajelor de programare.

Cuvântul reprezintă un grup de simboluri din alfabetul A, concatenate și aranjate în ordinea $c_i = a_1^i a_2^i ... a_{n_i}^i$ ușor de pronunțat, unde n_i este numărul de simboluri din cuvântul c_i. Se asociază o semnificație grupului de simboluri în funcție de elementele extrase și relația de ordine dintre ele. Un cuvânt c_i se caracterizează prin lungime, $L_C(c_i)$, exprimată ca număr de elemente ce intră în alcătuirea sa.

În funcție de modul de construire, se disting următoarele clase de cuvinte:

- cuvinte simple – formează lexicul de bază al unui limbaj;
- cuvinte compuse – se formează din agregarea cuvintelor simple prin intermediul operațiilor: prefixare, sufixare, concatenare.

Vocabularul V_A este o mulțime formată din cuvinte diferite:

$V_A = \{ c_0, c_1, c_2, c_3, c_4, ..., c_n \};$

Lungimea vocabularului V_A poate fi notată cu $Lgv(V_A)$, care indică numărul de cuvinte ce intră în alcătuirea vocabularului.

Între cuvintele vocabularelor se stabilesc asocieri. Astfel, se construiesc dicționare prin intermediul cărora se evidențiază semnificația textului în două limbi diferite.

Vocabularul textului este constituit din mulțimea cuvintelor diferite, care apar într-un text.

Vocabularul textului, VT, este inclus în vocabularul V_A. Uneori, VT este identic cu V_A.

De exemplu, pentru:

alfabetul $A = \{ a, b, c, d, e \},$

vocabularul $V_A = \{ abc, bbd, ddd, aaaaaaaa, bbbb, eeeee, abced \}$

se construiește textul:

T = < *abc + abc + abc + aaaaaaaa + aaaaaaaa + aaaaaaaa +eeeee+eeeee* >

Vocabularul textului T, VT, este alcătuit din cuvintele:

abc
aaaaaaaa
eeeee

Lungimea vocabularului VT este de 3 cuvinte.

Vocabularul VT este inclus în vocabularul V_A.

Regulile de construire a vocabularului constau în:

- pronunția nativă – succesiuni de sunete asociate cu obiecte, ființe, procese, fenomene din lumea reală;
- preluarea de subșiruri din cuvintele vocabularului;
- preluarea din alte limbi (neologisme): *sponsor, software, hot-dog* etc.

Textul T reprezintă o succesiune de cuvinte din vocabularul V_A delimitate prin separatori.

$T = \{ c_{k1}, c_{k2}, c_{k3}, ..., c_{kn} \};$

unde $k_1, k_2, ... k_n \varepsilon \{1,n\}$.

Lungimea textului este exprimată în următoarele moduri:

- $LG_{CAR}(T)$: număr de caractere;
- $LG_{SYMB}(T)$: număr de simboluri;
- $LG_{WORD}(T)$: număr de cuvinte;
- $LG_{PH}(T)$: număr de propoziții;
- $LG_{STM}(T)$: număr de fraze.

Şablonul este o formalizare a regulilor de construire a textelor. Textul este împărțit în subșiruri de cuvinte, iar șablonul impune ca fiecare subșir să aparțină unui subvocabular.

Un exemplu de șablon este structura unui cod sursă, ce conține secțiuni standard, care se repetă de la un cod sursă la altul. Ceea ce diferă este conținutul informațional, reprezentat prin entitatea text respectivă.

Șablonul privește și un mod de formatare a construcțiilor sintactice. De exemplu, tipul variabilelor, declararea unei proceduri, poziția blocului de declarare a variabilelor etc.

De asemenea, există posibilitatea ca șablonul unei entități text să conțină informație statică, netransmisibilă de la o entitate la alta, și date dinamice, care sunt particulare fiecărei entități.

Operatorii sunt simboluri utilizate pentru precizarea operațiilor care trebuie executate asupra operanzilor.

Operanzii se regăsesc în constante, variabile, nume de funcții, expresii.

Expresiile sunt entități construite cu ajutorul operanzilor și operatorilor, respectând sintaxa și semantica limbajului. Cea mai simplă expresie este cea formată dintr-un singur operand.

Separatorul este un simbol ce nu aparține alfabetului A, care are rolul de a delimita cuvintele ce alcătuiesc o secvență de cuvinte. Dacă se definesc separatorii $s_1, s_2, ..., s_r$, oricare dintre ei se utilizează pentru a delimita cuvintele dintr-o înșiruire de cuvinte. Dacă un separator este utilizat consecutiv, se obține o secvență de separatori.

Cuvintele cheie sunt construcțiile pe baza cărora se definește și se delimitează un domeniu în mulțimea de activități umane. Ele se concretizează prin concepte definite în cadrul domeniului.

Se procedează la definirea domeniului pentru care se efectuează testarea. Există texte pentru prezentarea domeniului. Cuvintele cheie din textul asociat domeniului apar sub formă de cuvinte cheie și în cazul întrebărilor și variantelor de răspuns din bateriile de texte.

În cadrul unui limbaj de programare, cuvintele cheie sunt împărțite pe categorii de aplicabilitate: cuvinte folosite pentru definirea tipului de variabile, cuvinte folosite pentru definirea tipului de constante, cuvinte folosite pentru a realiza instrucțiunile asociate structurilor repetitive sau structuri secvențiale, cuvinte folosite pentru apelarea de funcții predefinite, cuvinte care apelează biblioteci de funcții.

Cuvintele utilizator sunt construcțiile definite de către programator; se concretizează prin concepte definite în funcție de necesitatea folosirii acestora; de exemplu, se folosesc cuvinte standard, aparținând unui stil de realizare a codului, denumit în limba engleză *„coding style"*, acestea alcătuind un vocabular.

O ***instrucțiune executabilă*** realizează acțiuni precum: execută metode sau proceduri, controlează fluxul execuției codului. O instrucțiune este, în unele cazuri, scrisă pe mai multe linii prin utilizarea caracterelor de continuare a liniei specifice fiecărui limbaj.

Vocabularul limbajului C++ este format din:

- literele mici *{a,b,c,....z}*;
- literele mari *{A,B,C,.....,Z}*;
- operatori:
  - **Aritmetici**
    - operatorii unari de păstrare / schimbare a semnului: + -
    - operatorii binari multiplicativi - de înmulțire, împărțire, modulo: * / %
    - operatorii binari aditivi - de adunare și scădere: + -
  - **Relaționali**
    - "<" "<=" ">" ">=" "==" "!="

- **Logici**
  - *"!"* **negație** logică
  - *"&&"* **si** logic
  - "||" **sau** logic
- **La nivel de bit:**
  - *"~"* complementare
  - *">>"* deplasare la dreapta
  - *"<<"* deplasare la stânga
  - *"&* si
  - "^" sau exclusiv
  - "|" sau
  - separatorii reprezentați în mulțimea S

    $S = \{$ `}` , `{` , ` `(tab)` , ` `(space)` , `.` , `;` , `(` , `)` , `_` $\}$
  - cuvintele cheie:
  - pentru domeniul informatic *{procedură, variabilă globală, structură repetitivă}*
  - specifice limbajului C++ *{char, int, long, float, unsigned, const, double, function, printf, scanf, stdin, stdout, putchar, for, while, return, case, main, include, if, abs}*
  - cuvintele definite de utilizator;

Tabelul 1. Cuvinte utilizator și semnificația acestora pentru programator

Cuvânt utilizator	Semnificație pentru programator
Inc	*Incrementare*
Dec	*Decrementare*
Matrice1	*Numele unei matrice definite de programator*
A	*Vectorul de numere*
B	*Vectorul de corespondență*

  - instrucțiuni (executabile, neexecutabile)

Există mai multe ***categorii de instrucțiuni*** executabile:

- **simple**:
  - instrucțiuni de intrare / ieșire (Read_Write);
  - instrucțiuni de atribuire / memorare;
  - instrucțiuni de transfer (Return_Exit);
  - instrucțiuni de operare liste (inserare, extragere).
- **structurare**:
  - instrucțiuni secvențiale (Begin_End);
  - instrucțiuni decizionale / decizia:
    - instrucțiuni cu selecție simplă (If_Then_Else);
    - instrucțiuni cu selecție multiplă (Case);
  - instrucțiuni repetitive / repetiția:
    - ciclu cu test inițial (While_Do);
    - ciclu cu test final (Repeat_Until);

- ciclu cu contor (For_Do);
- apel de procedură / funcție.

Instrucțiunile executabile apelează diferite categorii de funcții de bibliotecă. Aceste funcții sunt parametrizate și, prin apelarea lor, se realizează particularizarea parametrilor cu care funcția a fost construită.

```
using System;                          //declară domeniul de nume utilizat

public class Exceptie : Exception      //declară clasa Exceptie derivată din clasa Exception
{                                      //declară început de bloc
  public Exceptie(string p_szMotiv)    //declară antet constructor
    : base(p_szMotiv) { }              //apel instrucțiune executabilă, execută constructorul
                                       clasei moștenite
}                                      //sfârșit de bloc
```

Un modul de cod poate începe cu o secțiune de declarații.

Prin ***declarații*** se înțeleg ***instrucțiuni neexecutabile*** prin care se definesc constante, variabile și proceduri externe.

```
#define DEBUG                  //declară constantă preprocesare
...
If def DEBUG                   //verifică declarare constantă preprocesare
...
```

Principala componentă a unui program este *blocul* format din *declarații* și instrucțiuni, din linii sursă ce includ:

- partea de declarare a etichetelor, în care se definesc etichetele folosite în partea de instrucțiuni a blocului;
- partea de declarare a tipurilor, care conține definirea unor tipuri de date pe baza altor tipuri definite anterior, standard sau ale utilizatorului;
- partea de declarare a constantelor, care cuprinde identificarea *constantelor simbolice* (identificatori asociați unor constante) și a constantelor cu tip (variabile inițializate în momentul compilării);
- partea de declarare a variabilelor, care conține definirea tuturor variabilelor specifice blocului respectiv;
- partea de declarare a procedurilor și / sau funcțiilor.

Programele C++ sunt entități text dezvoltate după regulile de sintaxă ale limbajului.

Se consideră P mulțimea programelor C++, definită prin $P = \{ P_1, P_2, ..., P_N \}$, unde N reprezintă numărul de programe existente într-o companie de dezvoltare software.

Fie programele P_i *si* P_j aparținând mulțimii P. Se spune că P_i si P_j sunt ortogonale dacă indicatorii folosiți pentru determinarea gradului de asemănare au valori cuprinse în intervalul [0.92;1]. Ortogonalitatea programelor identifică diferențele între programe, pe baza conținutului lor și pe baza modalității de aranjare a conținutului. Sunt avute în vedere declarațiile făcute în cadrul programelor, cuvintele cheie folosite, structurile repetitive ce stau la baza implementării programelor, precum și rezultatele obținute plecând de la același set de date de intrare.

Programele sunt identice dacă au un grad de asemănare ridicat, au aceeași structură, aceleași cuvinte cheie, ordinea de folosire a acestora este aceeași, rezultatele furnizate sunt identice și numărul de pași parcurși pentru furnizarea rezultatelor corespund.

Programe ortogonale prezintă diferențe de structură, de realizarea și de finalitate.

Programe care suferă modificări sunt programele de la care se pleacă în realizarea de noi programe sursă. Diferențele între aceste programe nu sunt mari, noul program reținând în structura sa aspecte ce se regăsesc în programul original.

Programe care suferă interschimb de comenzi, suferă variații în ceea ce privește dispunerea în codul sursă a comenzilor, a declarării variabilelor de lucru, fără a suferi schimbări majore privitoare la termenii utilizați și la structurile implementate.

Programele translatate automat din C++ în Pascal, trebuie considerate neortogonale.

Scrierea în corespondență a cuvintelor cheie permite realizarea de definiri de operanzi echivalenți în programele C++, pornind de la programele Pascal.

Se prezintă corespondența între programele scrise în C++ și Pascal, astfel:

- pentru definirea variabilelor:

C++	Pascal
int	*integer*
float	*real*
struct	*record*
switch	*case*
*char **	*string*

- pentru definirea unui articol din programe C++:

```
struct nume {
  tip1 nume1;
  tip2 nume2;
  ...
  tipn numen;
};
```

îi va corespunde construcția în limbajul Pascal, obținută prin translatarea automată:

```
nume = record
  nume1: tip1;
  nume2: tip2;
  ...
  numen: tipn;
end;
```

Structurilor alternative multiple din programele C++:

```
switch (expresie)
  {
  case val1: { ... break; }
  case val2: { ... break;}
  ...
  default: { ... break;}
  }
```

le corespund, prin translatare automată, descrierile următoare în limbajul Pascal:

```
case expresie of
   begin
   val1: begin ... end;
   val2: begin ... end;
   ...
   valn: begin ... end;
   end;
```

Dezvoltarea de software impune o atitudine de asigurare a ortogonalității programelor. Programele aflate într-un depozit software trebuie supuse unui proces de analiză a ortogonalității.

Conceptele utilizate în analiza și proiectarea produselor – program sunt:

- ***clasă de obiecte*** – o mulțime de obiecte care împart o structură și un comportament comune; este un șablon pe baza căruia se instanțiază obiecte;

- ***moştenire*** – o relaţie între clase prin care o clasă preia atributele şi metodele definite în una sau mai multe clase;
- ***instanţiere*** – proces de creare a unui obiect şi iniţializarea sa cu date specifice;
- ***atribut*** – variabilă definită în cadrul unei clase de obiecte; mulţimea atributelor reflectă starea unui obiect;
- ***metodă*** – operează asupra datelor ca răspuns a mesajelor primite, fiind parte a declaraţiei clasei de obiecte; mulţimea metodelor precizează comportamentul unui obiect;
- ***obiect*** – o instanţiere a unei clase capabilă să salveze o stare şi care oferă o serie de operaţii care examinează starea şi / sau o modifică.

2. Indicatori ai ortogonalităţii programelor C++

Construirea de indicatori ai ortogonalităţii, care să respecte toate cerinţele impuse indicatorilor, este un obiectiv destul de dificil de realizat. Se urmăreşte identificarea cerinţelor importante pentru conceptul de ortogonalitate şi care, implementate asupra produselor - program supuse analizei, ajută la identificarea elementelor de originalitate.

Există numeroase abordări ale analizei ortogonalităţii:

- analiză pentru întreaga entitate;
- analiză pe amprente.

Pentru studierea ortogonalităţii, se propune algoritmul următor:

- ***selecţie limbaj***, prin care utilizatorul produsului - program are la dispoziţie o varietate de limbaje de programare din care poate realiza selecţia înainte de procesul de analiză a textelor sursă. Selecţia în cauză se realizează prin intermediul unui control aparţinând mediului de dezvoltare C#, denumit ComboBox. Acest control are asociată o colecţie, în cazul nostru, această colecţie fiind reprezentată de: C#; PHP; C++; JAVA; VB;
- ***selecţie fişiere***, care, după încheierea cu succes a primul pas, face trecerea la selecţia fişierelor de comparare. Acestea sunt definite după un şablon prestabilit de programator. Ele trebuie să respecte o structură, iar în denumirea fişierelor ce conţin texte sursă, să fie inclus numele limbajului corespunzător. De asemenea, se pot stabili reguli cu privire la locaţia fizică, unde trebuie să fie regăsit textul sursă;
- ***selecţie separatori*** este realizată automat. Implicit se creează un fişier în care sunt incluşi toţi separatorii, în funcţie de specificul fiecărui limbaj în parte. Aceste fişiere pot fi editate prin intermediul produsului - program;
- ***comparare texte sursă*** este procesul care se realizează la nivel de cod, fără a vizualiza în mod grafic procesele desfăşurate pentru determinarea soluţiei;
- ***parcurgere fişiere*** prin care textele sursă aparţinând aceluiaşi program sunt parcurse în vederea determinării apariţiei cuvintelor din vocabular şi calculării gradului de asemănare dintre cele două. Parcurgerea se realizează linie cu linie;
- ***consultare vocabular*** se realizează de fiecare dată când se determină un separator de cuvinte. Cuvântul rezultat se compară cu cel din vocabular. Dacă acestea sunt identice, este reţinut pentru o prelucrare viitoare;
- ***calcul indicatori*** calculează frecvenţele de apariţie ale fiecărui cuvânt din vocabular.
- ***grad de similaritate***: este calculat cu ajutorul indicatorilor prezentaţi mai sus, formula de calcul fiind dată de relaţia:

$$G_s = \frac{h}{\gamma}$$

unde:

G_s - grad de similaritate;

h – numărul de apariţii, în cadrul cuvintelor, ale simbolurilor comune;

γ - numărul maxim de simboluri care formează cuvintele analizate.

- ***Grad de asemănare***: determinat cu ajutorul matricelor care evidențiază pozițiile de apariție a cuvintelor din vocabular. Acesta se calculează după formula:

$$D(T_j) = \sum_{k=1}^{n} f_k^j \lg_2 f_k^j$$

unde:

$D(T_j)$ - grad de asemănare a cuvântului T_j;

f_k^j - frecvențele de apariție ale cuvintelor din vocabular în textul T_j.

- ***comutativitate*** arată că, în caz de comutativitate totală, prin interschimbarea lui li cu lj, textul rămâne la fel;
- ***interschimb*** este procedeul prin care matricele sunt ordonate pentru calcularea gradului de asemănare. Se realizează interschimbul liniilor sau coloanelor astfel încât pe primele linii să fie afișate cuvintele din vocabular cu gradul de apariție cel mai mare;
- ***afișare*** este procedeul prin care rezultatele obținute în urma proceselor prezentate mai sus, sunt furnizate utilizatorului prin intermediul unui mediu vizual, folosind controale specifice mediului de dezvoltare C#.

3. Structura software pentru implementarea metricilor ortogonalității

Produsul - program realizează verificarea gradului de ortogonalitate în cadrul unui depozit software. Sunt definite criteriile de ortogonalitate, structura produselor - program, caracteristicile ce sunt urmărite pentru verificarea ortogonalității.

La crearea software-ului ce testează ortogonalitatea se are în vedere faptul că programele pentru o identificare corectă a gradului de ortogonalitate trebuie să aibă aceeași structură, să respecte o anumită compoziție.

În acest sens, software-ul propus spre analiză realizează testarea ortogonalității pentru un depozit de produse - program construite doar în limbajul C++.

La crearea software-ul s-a ținut cont de particularitățile limbajului C++ și, în același timp, s-a urmărit ca software-ul rezultat să aibă un grad de generalitate cât mai ridicat pentru a fi aplicat și pentru studierea ortogonalității altor limbaje de programare și chiar pentru studierea ortogonalității bazelor de date și chiar a alfabetelor construite cu ajutorul termenilor și denumirilor folosite în cadrul unui anumit limbaj.

Demersul este dat de:

- multitudinea de produse software, existente la ora actuală pe piață;
- stabilirea gradului de asemănare între produsele software;
- stabilirea produselor software identice;
- stabilirea produselor software originale;
- definirea de operații care să ducă la creșterea gradului de ortogonalitate a produselor - program;
- diferențierea clară a conceptului *reutilizare* de conceptul *inovație*;
- crearea unui software care să asigure un proces eficient de filtrare a produselor - program stocate în cadrul depozitului de software;
- construirea unui concept de transformare a programelor aparent diferite pentru a identifica aspectele în fapt comune, dar care creează o falsă impresie de originalitate; spre exemplu secvențele:

```
int a,b=a+1;
```

și

```
int a,b;
b=a+1;
```

la prima vedere sunt diferite, însă, în fapt, produc același rezultat; conceptul trebuind să țină cont de modalitatea ulterioară de utilizare a variabilelor definite;

- crearea unui software care să determine dacă un produs - program respectă criteriile de ortogonalitate, astfel încât să fie stocat în depozitul de produse - program; se urmăreşte astfel ca informaţia conţinută în cadrul depozitului să aibă redundanţă minimă.

Utilizatorul are posibilitatea de a modifica textele sursă existente, de a crea elemente de comparaţie, prin intermediul interfeţei utilizator pusă la dispoziţie, sau direct din interfaţa Windows, cu utilitarele specifice fişierelor text: NotePad, WordPad, MicrosoftWord.

Este oferită posibilitatea de selecţie a modului de rulare a produselor program prestabilite, în care sunt prezente cazuri predefinite de programator, singura interacţiune dintre utilizator şi datele de test fiind vizualizarea, consultarea sau folosirea acestor date în alte produse program prin copierea datelor cu ajutorul instrumentelor Copy şi Paste.

Există o componentă care necesită interacţiunea utilizatorului, prin inserarea datelor de text în două controale TextBox şi vizualizarea rezultatelor prin apelul evenimentelor de calcul cu ajutorul controlului Button.

De asemenea, se configurează *vocabularele* şi *separatorii.*

Produsul - program este extins pe diferite domenii, cum ar fi cele contabil şi statistic.

3.1. Cerinţe hardware şi software

Cerinţele hardware ale sistemului sunt influenţate de natura distribuită a acestuia. Este necesară o staţie pe care să ruleze un sistem Windows XP sau 2003 server, pentru a exista compatibilitatea sistemului cu Framework 2.0.

Cerinţele software implică existenţa unui Windows Installer 3.0 sau a unei versiuni mai recente, de asemenea, existenţa Framework 2.0 pentru compatibilitatea între mediul în care a fost dezvoltată aplicaţia şi fişierele existente pe staţia de lucru.

Structura produsului - program este prezentată în figura 1:

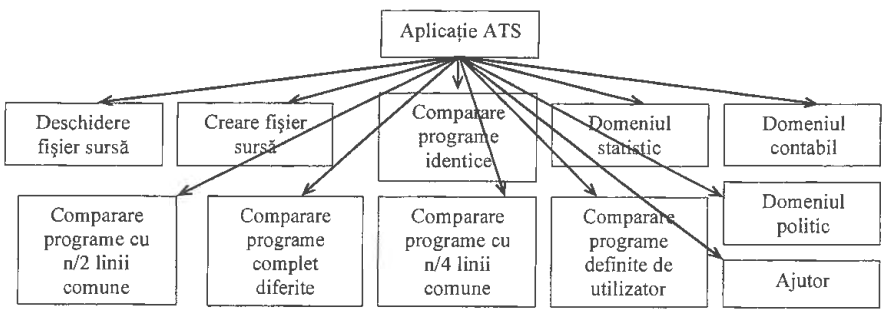

Figura 1. Structura aplicaţiei

Fiecare opţiune prezentată are drept corespondent o formă sau mai multe, în funcţie de complexitate, pentru a se realiza cu succes trecerea de la nivelul de cod sursă, la nivelul utilizator.

Codul sursă este împărţit în clase, proceduri, funcţii. În continuare, vor fi prezentate funcţiile cu caracter determinant în derularea algoritmilor de comparare a textelor sursă:

- ***esteElement***: determină dacă elementul curent se regăseşte în vocabularul specificat la începutul apelului;
- ***calcFrecvCuv***: realizează identificarea frecvenţelor de apariţie a unui cuvânt din vocabular;
- ***calcPozCuv***: prin intermediul acestei proceduri, se determină poziţia la care a fost găsit elementul, pentru a se construi ulterior matricele de corespondenţă;
- ***arataMatr***: se populează cele două controale textbox cu elementele matricei compuse din poziţiile de apariţie a cuvintelor din vocabular;
- ***calcGradAsem***: realizează identificarea gradului de asemănare a textelor sursă, care se determină prin raportul dintre numărul de frecvenţe identice de apariţie şi numărul elementelor de comparaţie din vocabular;

- ***calcGradSim***: se prelucrează matricea pentru a se determina cu funcția logaritmică gradul de asemănare a două texte sursă în funcție de pozițiile de apariție a cuvintelor definite în vocabular în textul sursă;
- ***ordonareMatrFrecv***: realizează ordonarea matricelor în funcție de numărul maxim de apariții ale cuvintelor. Această ordonare se realizează prin interschimbul liniilor și ținând cont de comutativitate, care este considerată totală;
- ***public class FctComp***: este o clasă definită pentru a ușura lucrul cu funcțiile și procedurile create. Cu ajutorul acestei clase, apelul funcțiilor se realizează prin apelul numelui clasei, alături de numele funcției/procedurii.

De exemplu:

FctComp.CalcFrecvAparitie()

În vederea studierii ortogonalității programelor C++, se creează un depozit software, în care sunt stocate programe. Pentru a fi introduse în depozitul software, programele trebuie să respecte o serie de condiții legate de gradul de asemănare dintre ele, pentru a încărca doar programe originale.

Un criteriu esențial pentru diferențierea produselor software este gradul acestora de complexitate. Pe baza acestui criteriu, se realizează clasificarea programelor, astfel încât la studierea ortogonalității se realizează comparații pe clase de complexitate, contribuind astfel la îmbunătățirea timpilor de execuție ai programului.

4. Lotul de programe C++ pentru evaluarea ortogonalității

Lotul de programe destinat testării produsului software este compus din programe scrise în C++. Programele sunt astfel alese încât să fie atinse toate caracteristicile programelor C++.

Loturile de programe sunt create astfel încât să fie atinse toate situațiile în care se pot regăsi două programe sursă.

Din rularea produsului – program, a rezultat că cele două coduri sursă se aseamănă în proporție de 25%.

Compararea programelor sursă se face folosind același principiu ca și compararea textelor.

Compararea de texte este operația de punere în corespondență a două entități ET_i și ET_j în vederea stabilirii asemănărilor sau deosebirilor între ele.

Comparabilitatea textelor se analizează la următoarele niveluri:

- formă de reprezentare;
- conținut informațional.

Reprezentarea percepțiilor omului privind lumea înconjurătoare se face prin intermediul construcțiilor sintactice, formate din simboluri alfanumerice, grafice, spațiale.

În cea mai mare parte a lor, textele se realizează pe baza alfabetului A și sunt colecții de cuvinte aranjate într-o ordine conform regulilor definite. Există mai multe criterii de comparare a două entități text, ET_i și ET_j, pornind de la forma de reprezentare a acestora. Criteriile vizează:

- lungimea entității text – este exprimată în mai multe moduri;
- număr de simboluri utilizate;
- număr de caractere utilizate cu posibilitatea de a diferenția pe caractere mici și caractere mari;
- număr de cuvinte folosite;
- număr de pagini;
- număr de bytes ocupați, în cazul entităților text care sunt date în format electronic;
- lungimea vocabularelor entităților – exprimată ca număr de cuvinte distincte utilizate în construcția entităților text ET_i și ET_j;
- lungimea vocabularelor cuvintelor cheie – sunt luate în considerare cuvintele definitorii pentru domeniul abordat;
- numărul de cuvinte cheie din titlurile entităților;

- diversitatea vocabularelor entităților – raportează cuvintele din vocabularele entităților cu cele din vocabularul limbii în care au fost realizate;
- frecvențele de apariție a cuvintelor din entitățile ET_i și ET_j – pentru fiecare cuvânt din vocabularul unei entități se asociază numărul de apariții în cadrul entității;
- frecvențele de apariție a cuvintelor din vocabularele entităților în raport cu un vocabular definit de utilizator;
- construirea matricelor de precedențe în vederea analizei comparate a textelor în funcție de pozițiile construcțiilor sintactice.

Comparabilitatea textelor din punctul de vedere al conținutului informațional are în vedere următoarele aspecte:

- delimitarea domeniului abordat prin evidențierea semanticii cuvintelor cheie;
- înțelesul cuvintelor specifice domeniului și sensul în care acestea au fost utilizate în construcția entității text;
- acoperirea cadrului conceptual prin utilizarea cuvintelor specifice domeniului;
- aducerea entităților ET_i și ET_j la o formă comună prin eliminarea cuvintelor de legătură și a prefixelor / sufixelor adăugate conform regulilor gramaticale specifice limbii;
- identificarea sinonimelor pentru cuvintele din vocabularul entităților și modificarea textelor prin utilizarea aceleiași forme a cuvântului;
- identificarea și prezentarea elementelor de natură teoretică și practică, exprimate în cele două entități, și realizarea de asocieri;
- identificarea funcționalităților descrise în texte, inventarierea lor și evidențierea eventualelor asocieri.

Având în vedere natura calitativă a aspectelor din lumea reală, prezentate într-o entitate text, se impune derularea de procese și aplicarea de operații asupra entităților text, supuse analizei comparative. Scopul este de a aduce textele la o aceeași formă cuantificabilă din punct de vedere numeric.

Criteriile de clasificare a programelor stocate în depozitul software contribuie la identificarea de programe originale, calitative și cu un grad ridicat de solicitare.

Valoarea rezultată pentru indicatorul de ortogonalitate este supusă analizei de calitate, iar în urma testelor a rezultat că indicatorul respectă criteriile de calitate, respectiv necatastroficitate, senzitivitate și necompensatoriu.

În urma rulării produsului - program, s-a observat că programele din aceeași clasă de complexitate sunt mai puțin ortogonale, lucru explicabil prin faptul că vocabularul programelor este format din aceleași cuvinte cheie, din aceleași tipuri de date, folosește aceleași tipuri de structuri.

În tabelul 2 sunt prezentate produse - program identice pentru care este determinat indicatorul de ortogonalitate:

Tabel 2. Texte sursă ale programelor identice

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace CodPtLicenta
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace CodPtLicenta
```

```
{
    public partial class Form1 : Form
    {
        public Int32  suma;
        public Form1()
        {
            InitializeComponent();
        }
        private void suma_2numere(Int32 a, Int32
b)
        {
            this.suma = a + b;
        }
        private void Calculeaza_Click(object
sender, EventArgs e)
        {
            suma_2numere(Int32.Parse(tbxA.Text),
Int32.Parse(tbxB.Text));
            this.tbxSuma.Text = suma.ToString();
        }
    }
}
```

```
{
public partial class Form1 : Form
{
        public Int32  suma;
        public Form1()
{
InitializeComponent();
}
private void suma_2numere(Int32 a, Int32 b)
{
            this.suma = a + b;
}
private void Calculeaza_Click(object sender,
EventArgs e)
{
suma_2numere(Int32.Parse(tbxA.Text),
Int32.Parse(tbxB.Text));
this.tbxSuma.Text = suma.ToString();
}
    }
}
```

Aşa cum se observă, indicatorul de ortogonalitate are valoare *0*, programele fiind identice atât în ceea ce priveşte partea declarativă, cât şi în ceea ce priveşte conţinutul. Variabilele sunt declarate folosind acelaşi format, funcţiile sunt declarate şi apelate în acelaşi context, declararea de biblioteci se realizează în mod similar.

În tabelul 3, sunt prezentate două programe sursă, a căror ortogonalitate este maximă.

Tabelul 3. Texte sursă total diferite

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

```
namespace CodPtLicenta
{
   public partial class Form1 : Form
   {
      public Int32  suma;
      public Form1()
      {
         InitializeComponent();
      }
      private void suma_2numere(Int32 a, Int32
b)
      {
         this.suma = a + b;
      }
      private void Calculeaza_Click(object
sender, EventArgs e)
      {
         suma_2numere(Int32.Parse(tbxA.Text),
Int32.Parse(tbxB.Text));
         this.tbxSuma.Text = suma.ToString();
      }
   }
}
```

```
namespace CodPtLicenta
{
public partial class Form1 : Form
{

public Form1()
{
        InitializeComponent();
    }
    }

    }
```

Valoarea indicatorului de ortogonalitate este, în acest caz, *1*, rezultând că cele două produse - program sunt total diferite. Se observă o oarecare asemănare la declararea bibliotecilor, dar acest fapt nu afectează cu nimic caracterul de originalitate al programelor, ştiut fiind faptul că nu este important ceea se foloseşte din cadrul bibliotecilor, ci modul cum se foloseşte.

Au fost prezentate cele două situaţii extreme în care se află două programe sursă, respectiv total diferite şi identice.

În tabelul 4, este prezentată o situaţie intermediară.

Tabelul 4. Texte sursă în care apar unele diferenţe între programe

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Text;
using System.Windows.Forms;
Using System.SQL;

namespace CodPtLicenta
{
   public partial class Form1 : Form
   {
         public Int32  suma;
      public Form1()
      {
         InitializeComponent();
            this.suma=0;
            MessageBox.Show("Suma a fost
intializata cu valoarea 0:");
      }
      private void Calculeaza_Click(object
sender, EventArgs e)
      {
         form1.Close();
      }
   }
}
```

```
using System.Text;
using System.Windows.Forms;

namespace CodPtLicenta
{
public partial class Form1 : Form
{

public Form1()
      {
         InitializeComponent();
      }

      private void suma()
{
         break;
      }
   }
}
```

În acest exemplu, ortogonalitatea este 0.75.

Este foarte important ca, la analizarea surselor programelor, să fie construite o serie de criterii valide şi cuprinzătoare pentru caracterizarea produselor software. Două programe sunt aparent identice, dar schimbări care nu sunt evidente contribuie la furnizarea de rezultate total diferite. În exemplele de mai sus, se observă că toate secvenţele folosesc aceleaşi biblioteci, dar aceasta nu înseamnă că toate secvenţele furnizează acelaşi rezultat.

Pentru analiza programelor se folosesc criterii ce cuprind gradul de utilizare a cuvintelor cheie, ordinea de declarare a variabilelor, implementarea structurilor repetitive, variabilele folosite în cadrul structurilor repetitive, numărul de paşi parcurşi de program pentru a furniza rezultate, nivelul de folosire a variabilelor în cadrul programelor.

5. Căi de creştere a ortogonalităţii programelor C++

Prin construirea unui depozit de produse software, se urmăreşte crearea de software original, reducerea efortului depus pentru a crea produse - program care să corespundă cerinţelor formulate de clienţi, prin realizarea de căutări în cadrul depozitului şi identificarea soluţiilor propuse pentru aceeaşi problemă. Dacă au fost formulate soluţii de către programatori, în baza acordurilor realizate prin aderarea la comunitatea reprezentată de dezvoltatorii ce au contribuit cu programe la realizarea depozitului de

software, ele sunt preluate, studiate şi modificate pentru a întruni condiţiile clientului.

Pentru creşterea ortogonalităţii programelor se urmăreşte implementarea soluţiilor:

- specializarea programelor pe tipuri de problematică;
- crearea de biblioteci care conţin programe specializate şi apelarea procedurilor din cadrul bibliotecilor, funcţie de specificul fiecărei probleme de rezolvat;
- dezvoltarea unui stil de programare adecvat, care face ca la probleme diferite să fie folosite resurse diferite, în concordanţă cu specificul problemei; se evită astfel folosirea inutilă de resurse.

Creşterea ortogonalităţii programelor are ca rezultat important creşterea, în primul rând, a calităţii produselor software. Programele sunt create pentru a furniza soluţii. Cu cât soluţia pe care respectivul software o implementează este mai apropiată de cerinţe, cu atât calitatea produsului este mai bună. Se urmăreşte axarea pe rezultate şi mai puţin pe conţinutul în sine al produsului.

Înainte de achiziţie, sunt furnizate seturi complete de teste care identifică erorile de rezultat.

Prin construirea unui depozit software, se oferă spre consultare şi folosire un important stoc de funcţii proceduri, programe, comentarii asupra programelor, rezultate furnizate care ajută la crearea de software de calitate.

Problema în discuţie nu ţine de analizarea codului sursă pentru a determina gradul de ortogonalitate, acesta fiind preluat integral, ci de identificarea a procedurilor prin care respectivul cod sursă a fost încorporat în cadrul aplicaţiei, cum a preluat sarcinile din partea aplicaţiei generale şi cum prelucrează datele furnizate de utilizator. În final, întreaga aplicaţie va fi la rândul ei disponibilă, fiind creată pe bază de software „liber", programatorul ce se ocupă cu crearea programului principal fiind cel care face diferenţa între două sau mai multe produse software, care îşi propun ca finalitate realizarea aceluiaşi set de cerinţe.

Analiza programului apelator combinată cu analiza rezultatelor furnizate oferă suficiente rezultate pentru analiza ortogonalităţii a două produse software.

Un alt concept deosebit de important avut în vedere o dată cu crearea unui depozit software este cel al reutilizării produselor software, coroborat cu conceptul de inovaţie.

Se pune problema efortului depus pentru a descoperi noi produse software care să aibă o finalitate pentru utilizator comparat cu efortul depus pentru a îmbunătăţi un software existent care are aceeaşi finalitate.

Ortogonalitatea produselor software trebuie axată pe crearea de software care să rezolve o serie de cerinţe noi, nu să reinventeze software pentru aceleaşi cerinţe.

Creşterea ortogonalităţii produselor software trebuie să aibă în vedere că abordarea de noi domenii pentru crearea de software va face ca noile programe să difere fundamental de cele vechi. Plecând de la acelaşi domeniu de activitate, programele trebuie structurate astfel încât să ofere o maximă flexibilitate pentru adaptarea acestora la toate cerinţele domeniului. Trebuie avută în vedere la construirea de software cerinţe de extindere a aplicaţiei la nivelul întregii arii de activitate.

Abordarea de noi domenii face ca nivelul de aplicabilitate al informaticii să crească şi conduce la apariţia de noi oportunităţi privitoare la rolul informaticii în domeniul social.

6. Concluzii

Prin studierea şi aprofundarea conceptului de ortogonalitate a programelor sunt create premisele îmbunătăţirii calităţii programelor, realizării de software orientat pe problematică şi este definit mediul de lucru pentru implementarea conceptului de reutilizare a produselor software.

Crearea unui depozit software contribuie la creşterea eficienţei de lucru, la îmbunătăţirea timpilor de execuţie a programelor şi, nu în ultimul rând, la crearea de software superior din punct de vedere calitativ, prin oferirea acestuia spre testare şi spre integrare în cadrul altor aplicaţii. Depozitul software este structurat pe domenii de interese, sintetizând astfel rezultatele obţinute în diverse domenii de activitate. Persoanele care au acces în cadrul depozitului beneficiază de această resursă importantă şi foloseşte modulele deja implementate pentru obţinerea de rezultate conform cu cerinţele într-un timp optim.

Implementarea unui sistem al calităţii produselor stocate în depozitul software, abordarea de noi domenii de interes, creşterea gradului de ortogonalitate sunt factori care conduc la îmbunătăţirea calităţii produselor software, la creşterea gradului de aplicabilitate a acestora şi, nu în ultimul rând, la o mai bună utilizare a produsului software.

Bibliografie

1. **IVAN, I., D. MILODIN, M. POPA**: Ortogonalitatea alfabetelor. Revista Română de Informatică şi Automatică, vol. 15, nr. 3, 2005, pp. 41 – 56.
2. **IVAN, I., M. POPA, C. BOJA**: Operaţii pe entităţi text. Conferinţa Ştiinţifică Internaţională „The Dichotomy Poverty – Wealth and Romania's Integration in The European Union", Universitatea „Lucian Blaga" Sibiu, 20 – 21 mai, 2005, vol. 3, pp. 464 – 473.
3. **IVAN, I., M. POPA**: Consistenţa entităţilor text, Studii şi Cercetări de Calcul Economic şi Cibernetică Economică, 2005 (în curs de apariţie).
4. **IVAN, I., M. POPA, AL. POPESCU**: Classes of Text Entities. Lucrările Workshop-ului Internaţional Information Systems and Operations Management, Universitatea Româno - Americană, Editura ProUniversalis, Bucureşti, 20 – 21 aprilie, 2005, pp. 195 – 204.
5. **POPA, M.**: Text Entities Metrics. În: Informatica Economică, vol. 9, nr. 2, 2005, pp. 56 – 60.
6. **IVAN, I., M. POPA:** Reprezentativitatea entităţilor text. Conferinţa Internaţională The Impact of European Integration on the National Economy, Universitatea „Babeş-Bolyai" Cluj-Napoca, 28 – 29 octombrie, 2005.
7. **POPA, M.**: Structured Text Entities Metrics. Proc. of The 36th Informational Scientific Symposium of METRA, Bucureşti, 26 - 27 mai 2005, pp. 486 – 491.
8. **IVAN, I., M. POPA, AL. POPESCU**: The Text Entities' Engineering. Workshop-ul internaţional „Collaborative Support Systems in Business and Education", Cluj-Napoca, 28 – 29 octombrie 2005, pp. 320 – 351.