Metrici ale re-ingineriei software

Prof.dr. Ion IVAN, prep. Felix SIMION, ec. Emilia STANCIU, Alexandra KARADIMOU
Catedra de Informatică Economică, A.S.E., Bucureşti

Lucrarea realizeaza o introducere în re-ingineria software şi reutilizarea software şi o trecere în revistă a unor metrici şi modele care facilitează evaluarea unor aspecte ale acestora.
***Cuvinte cheie**: metrici, modele, re-inginerie software, reutilizare software, programare orientată obiect.*

1. Introducere

Unul dintre motivele pentru care costurile mentenanţei software sunt mai mari decât cele ale dezvoltării de software este acela că, de cele mai multe ori, este dificil de pus în evidenţă structura unui produs software. Pentru a fi mentenabil, un produs software trebuie să fie uşor de înţeles, modificat şi testat. Dacă structura produsului software este ambiguă şi nu poate fi pusă în evidenţă cu uşurinţă, produsul nu poate fi întreţinut şi există în acest caz doua opţiuni: fie este rescris în întregime, fie este restructurat parţial sau total. Restructurarea se realizează utilizând metoda re-ingineriei software.
Re-ingineria software este procesul de analiză şi modificare a unui sistem existent cu scopul de a-l reconstitui într-o formă nouă [AYAC95]. Aceasta permite atât transformarea unor sisteme vechi într-o formă mai uşor de întreţinut, cât şi exploatarea unor tehnologii noi (interfeţe grafice, arhitecturi de aplicaţii distribuite incluzând structuri client-server etc.) în cadrul unor aplicaţii existente.

2. Re-inginerie software

Re-ingineria utilizează tehnicile ingineriei directe şi ale ingineriei inverse. *Ingineria directă* corespunde procesului de dezvoltare software tradiţională, având ca scop realizarea unui sistem nou. *Ingineria inversă* este procesul de analiză a unui sistem existent în vederea identificării componentelor şi relaţiilor dintre ele pentru trecerea de la o formă de reprezentare la alta, aflată cel puţin pe acelaşi nivel de abstractizare [AYAC95].
Pentru aplicarea tehnicilor de re-inginerie este utilizată o categorie aparte de instrumente de întreţinere. Printre acestea se numără: analizatori de cod, reformatoare, instrumente de restructurare, translatoare, suporturi de comparare a codului sursă, analizatoare de documente şi instrumente de inginerie inversă. De obicei sunt supuse re-ingineriei software acele componente pentru care restructurarea constituie o soluţie mai bună decât rescrierea integrală. Urmatoarele componente se încadrează în această categorie: componente la care s-a constatat un procent mare de defectări; componente care au o rata anuală a modificărilor ridicată; componente cu o complexitate ridicată; componente care nu îndeplinesc standardele.
Exemple ale aplicării re-ingineriei software sunt trecerea unui program dintr-un limbaj de programare în altul sau transformarea unui produs software neorientat obiect într-unul orientat obiect. Pentru ambele cazuri există instrumente software care realizează conversiile în mod automat, în special trecerea dintr-un limbaj de programare în altul. Majoritatea instrumentelor CASE actuale (şi în special cele orientate obiect) dispun de o componentă care realizează partea de re-inginerie software, generând - pe baza codului - diverse tipuri de diagrame ce descriu structura produsului software. Astfel:

- ObjectTeam al firmei Cadre [BALA96] permite citirea şi translatarea codului sursa C++ şi SmallTalk într-un model obiectual conform convenţiilor metodologiei OMT. Utilizatorii pot aplica metoda ingineriei inverse numelor caracteristicilor şi structurilor de moştenire ale claselor.
- Rational Rose al firmei Rational Software Corporation [SUCI96] permite actualizarea diagramelor de clase ale modelului aplicaţiei realizate pe baza codului modificat manual.

Analiza se poate efectua la diverse nivele de detaliere.

- Analytical Engine al firmei ViaSoft decompilează programe şi creează un depozit on-line cu informaţii despre program. Construieşte şi stochează cunoştinţe referitoare la modul de lucru al programelor.

Există preocupări în vederea elaborării unor metode de abordare generică a ingineriei inverse. Astfel, în cadrul Proiectului Esprit au fost dezvoltate [AYAC95]:

- *metoda REDO* include fazele: analiza sistemului pornind de la sursele şi documentele existente; inginerie inversa; stabilirea procedurilor de test; re-inginerie; exploatare.
- *metoda RE-ORDER* presupune: analiza sistemului existent, a obiectivelor urmărite şi a cerinţelor domeniului pentru stabilirea tehnologiei de utilizat; pregătirea instrumentelor şi aplicarea procesului de re-inginerie.

Aplicaţii ale ingineriei inverse au avut ca obiect programe scrise in limbaje procedurale [RAJL96], limbaje de programare orientate obiect [RAJL96], limbaje orientate proces [KUNZ95].

O altă faţetă a re-ingineriei software este re-ingineria în scopul reutilizării. Re-ingineria software în general are ca scop refacerea proiectului pornind de la cod şi modificarea apoi a proiectului în vederea obţinerii unui produs software mai flexibil. Produsul software este privit în acest caz ca un întreg. Re-ingineria software în scopul reutilizării se aplică la nivel de componente. Se încearcă divizarea întregului sistem în blocuri din care unele pot fi reutilizate, altele nu. Componentele reutilizabile au o funcţionalitate clară şi interfeţe bine definite. Realizarea internă nu prezintă interes când se reutilizează componenta respectivă şi nu trebuie să afecteze sau să depindă de alte componente ale sistemului în care va fi integrată. Re-ingineria în scopul reutilizării concentrează eforturile în direcţia găsirii şi extragerii componentelor reutilizabile dintr-un produs software existent. Sistemul din care au fost extrase componentele reutilizabile rămâne nemodificat, componen-tele fiind extrase, modificate şi utilizate în afara sistemului de proveniență. Este o diferență majoră față de re-ingineria software al cărei scop este chiar modificarea sistemului existent în vederea îmbunătăţirii sale.

3. Reutilizare software

Reutilizarea software este procesul de creare de sisteme software din elemente software existente [KARL95]. Elementele software care constituie blocurile constructive se numesc *componente software reutilizabile*.

Reutilizarea nu constituie un scop în sine, ci un mijloc de obţinere a obiectivelor generale ale organizaţiei. Este o sursă de creştere a productivităţii şi de îmbunătăţire a calităţii produselor realizate. De asemenea, contribuie la standardizarea şi interoperabilitatea produselor.

În prezent se consideră ca reutilizarea software vizează toate resursele utilizate şi produse în timpul dezvoltării software. Exemple de produse reutilizabile: specificaţii funcţionale, proiecte, module de cod, documentaţie, date de test şi instrumente. Ideea de reutilizare în sens larg este importantă, favorizând reducerea substanţială a costurilor deoarece reutilizarea unui rezultat dintr-o faza incipientă creşte probabilitatea reutilizării unor produse ulterior dezvoltate din el [BARN91].

Au fost propuse clasificări ale cunoştintelor reutilizabile, care se bazează pe unul din următorii trei factori sau pe o combinatie a lor: (a) stadiul dezvoltării în care cunoştinţele sunt produse şi/sau utilizate; (b) nivelul de abstractizare; (c) natura cunoştinţelor.

Jones [MILI95] a identificat patru tipuri de produse reutilizabile: *datele*, implicând o standardizare a formatului datelor; *arhitectura*, care presupune standardizarea unui set de convenţii de proiectare şi programare referitoare la organizarea logică a softwareului; *proiectul de detaliu*, pentru anumite aplicaţii financiare; *programul*, care se referă la reutilizarea codului executabil.

La rândul sau, Wegner clasifică elementele reutilizabile în: elemente care pot fi reutilizate într-un număr de aplicaţii (exemplu: o funcţie matematică); elemente care pot fi utilizate în versiuni succesive (o nouă versiune a unui produs, bazată pe versiunea

anterioară); elemente care sunt reutilizate ori de câte ori programul care îl conţine este executat (compilatoare); elemente care sunt reutilizate într-un program (subrutine).
În [KARL95] reutilizarea este clasificată după următoarele criterii:
(a) scopul reutilizării - defineşte domeniul reutilizării potenţiale al unei componente: reutilizare generală (sau reutilizare orizontală); reutilizare într-un domeniu specific (sau reutilizare verticală); reutilizare într-o familie de aplicaţii. (b) reutilizatorul ţintă (vizat): reutilizare internă; reutilizare externă. (c) granulozitatea componentelor manipulate: reutilizare cu componente de mică granulozitate (scară mica), de exemplu: funcţii I/E; reutilizare cu componente de mare granulozitate (scară mare).
Se disting două aspecte relativ la reutilizare: (a) dezvoltarea *pentru* reutilizare, care se referă la dezvoltarea de componente software reutilizabile şi (b) dezvoltarea *cu* reutilizare, care constă în dezvoltarea sistemelor cu componente software existente.
Dezvoltarea de software reutilizabil are ca obiective maximizarea gradului de acoperire a domeniului de aplicare (maximizarea probabilităţii de a găsi o potrivire exactă respectiv de a găsi una aproximativă) şi minimizarea costului adaptării componentelor returnate prin căutare aproximativă.
Dezvoltările cu componente reutilizabile au ca obiective minimizarea costului căutarii (exactă sau aproximativă) şi a costului adaptării.

4. Modele

Conform standardului SR ISO/CEI [ISO/CEI] 9126, *metrica* pentru calitatea software este scara cantitativă şi metoda care poate fi utilizată pentru a determina valoarea pe care o ia o trăsătura a unui produs software specific. *Modelul* specifică relaţii între metrici. Karlson [KARL95] clasifică metricile în:
- *metrici ale procesului* - utilizate pentru a estima efectul reutilizării asupra duratei de realizare şi productivităţii pentru produse şi componente.
- *metrici ale produsului* - utilizate pentru a măsura reutilizabilitatea şi calitatea unei componente. Metricile produsului sunt utilizate de producătorii şi utilizatorii de componente software reutilizabile cu scopul evaluării acestora.

Frakes şi Terry [FRAK96] clasifică modelele şi metricile reutilizării în: (a) modele ale analizei cost-profit; (b) modele de evaluare a maturităţii reutilizării; (c) metrici ale volumului reutilizării; (d) analiza modurilor de eşec; (e) metrici ale reutilizabilităţii; (f) metrici ale bibliotecilor pentru reutilizare.

a) Modele de analiză cost-profit

În literatura de specialitate au fost raportate mai multe modele cost-profit ale reutilizării. Nici unul din aceste modele nu sunt derivate din date şi nici nu au fost validate cu date. Ele permit însă o simulare a corelaţiilor dintre parametrii economici, cum sunt costul şi productivitatea.
Gaffney şi Durek [FRAK96] au propus două astfel de modele. *Modelul simplu* evidenţiază costul reutilizării componentelor software. *Modelul costului dezvoltării* este construit pe baza modelului simplu prin includerea costului dezvoltării de componente reutilizabile.
Ecuaţiile pentru modelul simplu sunt:

$$C=(b-1)R+1$$

$$P=\frac{1}{C}=\frac{1}{(b-1)\cdot R+1}$$

unde:
C - costul dezvoltării software (C=1 pentru un produs care încorporeaza doar cod nou),
R - proporţia codului reutilizat în produs ($R\leq 1$), b - costul relativ al încorporării codului reutilizat în noul produs (b=1 pentru un produs care încorporează doar cod nou), P - productivitatea.
Pentru ca reutilizarea să fie convenabilă din punct de vedere al costului, trebuie ca $b < 1$. Mărimea lui b depinde de faza ciclului de viaţă a componentei reutilizabile. Dacă se reutilizează doar codul sursă, atunci este necesar să se parcurgă fazele de specificaţii, proiectare şi testare pentru a se completa dezvoltarea componentei reutilizabile. În acest caz, Gaffney şi Durek estimează b=0,85. Dacă se reutilizează specificaţiile,

proiectul şi codul, atunci trebuie efectuată doar faza de testare şi se estimează b=0,08.
În modelul costului dezvoltării, costul se calculează astfel:

$$C = (b + \frac{E}{n} - 1) \cdot R + 1$$

unde:
E - costul dezvoltării componentei reutilizabile relativ la costul producerii componentei care nu este reutilizabilă (E>1),
n - numărul de utilizări prin care se amortizează costul dezvoltării codului.
Barnes şi Bollinger [BARN91] au examinat costul şi trăsăturile de risc ale reutilizării software şi au sugerat o abordare analitică privind realizarea unor investiţii bune pentru reutilizare. Au propus ca indicator al calităţii investitiei pentru reutilizare raportul:

$$Q = \frac{P}{I}$$

unde: I - investitia totală de reutilizare; P - profitul total al reutilizării. Acesta se calculează astfel:

$$P = \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} (c_i - r_i)$$

unde: p_i - profitul rezultat din investiţia pentru reutilizare activităţii **i**; c_i - costul estimat al activităţii i fără exploatarea reutilizării; r_i - costul activităţii i cu exploatarea reutilizării; n - numărul de activităţi care beneficiază de investiţie.
Dacă Q este semnificativ mai mare ca 1, investiţia pentru reutilizare este rentabilă. Pentru mărirea valorii indicatorului Q trebuie acţionat în sensul creşterii profitului total al reutilizării şi/sau reducerii investiţiei necesare pentru reutilizare. Creşterea profitului total al reutilizării se poate realiza prin: creşterea nivelului reutilizării, reducerea costului mediu al reutilizării.

b) Modele de evaluare a maturităţii reutilizării

Modelele permit evaluarea programelor de reutilizare dupa cât de avansate sunt în implementarea unei reutilizări sistematice.
Koltun şi Hudson [FRAK96] au dezvoltat un model al maturităţii reutilizării care include fazele: iniţială/haotică, monitorizată, coordonată, planificată, încorporată. Pentru fiecare din aceste faze se iau în considerare 10 dimensiuni ale maturităţii reutilizării: motivaţia, planificarea reutilizării, susţinătorul reutilizării, responabilitatea producerii reutilizării, procesul prin care reutilizarea este asigurată, elementele reutilizate, activitatea de clasificare, tehnologia suport, metricile, consideraţiile legale/contractuale/contabile. Pentru fiecare din cele 10 dimensiuni, volumul implicării organizaţiei creşte pe măsură ce aceasta progresează de la reutilizarea iniţială/haotică la cea încorporată.
Pentru a utiliza modelul, o organizaţie va evalua nivelul maturităţii reutilizării înainte de a iniţia un program de îmbunătăţire a acesteia. Evaluarea se va face prin identificarea amplasării în fiecare dimensiune. În continuare, organizaţia va utiliza modelul pentru a ghida activităţile care trebuie efectuate pentru a atinge nivelele superioare ale maturităţii reutilizării.
Un alt model este cel dezvoltat de Software Production Consortium [FRAK96]. Modelul are doua componente: un model de evaluare şi unul de implementare.
Modelul de evaluare constă dintr-un set de factori de succes pe care o organizaţie îi poate utiliza pentru a evalua starea curentă a practicii sale privind reutilizarea. Factorii sunt organizaţi în patru grupe primare: management, dezvoltare aplicaţii, dezvoltare componente, tehnologie/proces.
Modelul de implementare ajută în stabilirea priorităţii obiectivelor şi în construirea stadiilor succesive ale implementării. Sunt identificate patru stadii: oportunistic, integrat, structural, anticipativ.

c) Metrici ale volumului reutilizării

În general, această metrică este:

$$R = \frac{\text{volumul elementului din ciclul de viata reutilizat}}{\text{dimensiunea totala a elementului din ciclul de viata}}$$

O formă comună a acestei metrici se bazează pe numărul de linii de cod:

$$R' = \frac{\text{numarul de linii de cod reutilizate in sistem / modul}}{\text{numarul total de linii de cod din sistem / modul}}$$

Banker [BANK94] a implementat într-un mediu CASE orientat obiect calculul unor metrici ca:
• *procentul de obiecte noi* măsoara structura obţinută prin reutilizare. Este proporţia

obiectelor dintr-un sistem care a fost scrisă efectiv pentru acesta.

$$\text{procent obiecte noi} = \frac{\text{numar obiecte noi}}{\text{numar total obiecte utilizate}} \times 100$$

- *structura reutilizării*

$$\text{structura reutilizarii} = \frac{\text{numar total obiecte utilizate}}{\text{numar obiecte noi}}$$

Metricile de reutilizare pot fi aplicate la diferite nivele de analiză. Valoarea agregată a acestora subestimează beneficiile reutilizării. De exemplu, timpul de realizare a diferitelor tipuri de obiecte diferă şi este importantă ponderea tipurilor de obiecte în total obiecte reutilizate.

- *procentul de reutilizare externă*

$$\text{procentul reutilizarii} = \frac{\text{numarul de obiecte ale altor sisteme}}{\text{numar total de obiecte utilizate}}$$

- *procentul de reutilizare internă*

procentul reutilizarii interne = 100 - procentul obiectelor noi - procentul reutilizarii externe

Reutilizarea este *internă* daca un obiect creat pentru un sistem este utilizat de mai multe ori în cadrul aceluiaşi sistem. Reutilizarea este *externă* dacă un obiect dintr-un sistem este utilizat cel puţin o dată în cadrul unui sistem nou. Cu toate că ambele tipuri de reutilizări sunt de valoare egală, pentru susţinerea lor sunt necesare politici manageriale diferite. Reutilizarea externă garantează realizatorului că obiectul a fost testat în altă parte înainte de a fi utilizat. Gradul reutilizării interne va depinde probabil de mărimea echipei de realizare a aplicaţiei şi de calitatea comunicaţiei în cadrul ei. Gradul reutilizării externe poate să depindă de calitatea sistemului de indexare utilizat pentru a ajuta programatorii să identifice obiectele existente pe care le-ar putea utiliza.

d) Modelul modurilor de eşec al reutilizării software

Modelul propus de Frakes şi Fox [FRAK96] permite evaluarea calităţii unui program de reutilizare sistematică, pentru a determina obstacolele în reutilizarea într-o organizaţie şi pentru a plănui strategia de îmbunătăţire.

Modelul are şapte moduri de eşec corespunzătoare paşilor pe care un inginer software trebuie să-i parcurgă pentru a reutiliza o componentă: nici o încercare de reutilizare; elementul nu există; elementul nu este disponibil; elementul nu este găsit; elementul nu este înţeles; elementul nu este valid; elementul nu poate fi integrat.

Fiecare mod de eşec are cauze asociate. Pentru a utiliza acest model, o organizaţie culege date referitoare la modurile de eşec al reutilizării şi la cauzele lui, şi apoi utilizează informaţiile pentru îmbunătăţirea activităţilor de reutilizare.

e) Metrici ale reutilizabilităţii

Reutilizabilitatea este un atribut extern al produsului care se referă la uşurinţa cu care acesta poate fi reutilizat în alta parte (de exemplu în alt mediu), fără modificări majore [FENT94]. Atributul este specific unui anumit tip de componentă şi limbajului în care este implementată.

Metricile reutilizabilităţii sunt utile pentru proiectarea reutilizării şi pentru re-ingineria pentru reutilizare.

f) Metrici ale bibliotecilor pentru reutilizare

Biblioteca pentru reutilizare este un depozit pentru stocarea elementelor reutilizabile plus o interfaţă pentru căutare în depozit. Componentele sunt clasificate pentru a facilita accesul la ele. Criteriile de evaluare ale schemei de indexare sunt: costul, efectivitatea căutarii, suportul pentru înţelegere, eficienţa.

5. Modificari structurale induse de orientarea obiect

În prezent, metoda de construcţie software orientată pe obiecte este cea mai bună soluţie pentru obţinerea unei înalte reutilizări a codului [DIFE93].

Tehnologiile orientate obiect au impus un nou mod de a gândi dezvoltarea software, centrându-se pe reprezentarea entităţilor din lumea reală. Pe lângă această modalitate naturală de analiză şi proiectare software, orientarea obiect presupune şi un alt tip de structurare a produselor software. Clasicele module, obţinute de regulă ca urmare a structurării funcţionale, sunt înlocuite de către conceptele de clasă şi obiect.

Modelul orientat obiect creează premisele unei bune reutilizări a produselor software şi în special a codului. De asemenea, metoda re-ingineriei software poate fi aplicată mult mai uşor sistemelor obiectuale datorită modalităţii naturale de structurare a acestora. Un astfel de sistem este uşor de întreţinut, deoarece modificările se efectuează la nivelul implementării şi nu afectează arhitectura sistemului. Componentele sistemului sunt identificabile cu uşurinţă şi în consecinţă se "extrag" fără dificultate în vederea reutilizării. Reutilizarea se poate face:

- prin preluarea fără nici o modificare a codului existent;
- prin adaptarea codului existent pentru a satisface mai bine cerinţele proprii.

Situaţia descrisă în primul caz corespunde utilizării claselor existente, adică instanţierea acestora sau folosirea lor în relaţii de conţinere sau utilizare. Clasa care se reutilizează trebuie să aibă un grad mare generalitate. Reutilizarea este facilitată de faptul că detaliile de implementare sunt ascunse şi nu trebuie gestionate.

Reutilizarea prin adaptarea codului existent presupune preluarea unor clase cu un nivel de abstractizare ridicat şi specializarea acestora astfel încât să corespundă noilor cerinţe. Acest lucru se face programând doar diferenţele specifice noii clase cu ajutorul relaţiei de moştenire.

Deşi preocupări pentru reutilizarea software s-au manifestat începând cu apariţia programării procedurale şi modulare, se constată că de abia după răspândirea utilizării tehnologiilor obiectuale s-au elaborat adevărate strategii şi politici pentru reutilizare. La baza acestora stă crearea bibliotecilor de clase reutilizabile. După ce o clasă a fost testată şi acceptată într-o bibliotecă nu mai este nevoie să i se aducă modificări în momentul când este reutilizată: fie se utilizează aşa cum este, fie se "personalizează" pentru o aplicaţie proprie. În cazul bibliotecilor procedurale există doar posibilitatea utilizării fără modificări a codului existent. Aceasta înseamnă că, dacă modulele din biblioteci nu pot fi utilizate aşa cum sunt, ele nu sunt utilizate deloc.

Există mai multe tipuri de biblioteci de clase: pentru aplicaţii specifice unui domeniu, pentru aplicaţii generale, pentru aplicaţii ce utilizează anumite interfeţe standard etc. De exemplu, pentru programarea aplicaţiilor Windows utilizând limbajul C++ există bibliotecile de clase ale firmelor Microsoft (MFC - Microsoft Foundation Classes) şi Borland (OWL - Object Windows Library). Beneficiile rezultate din utilizarea claselor din aceste biblioteci sunt importante, şi se pun în evidenţă dacă raportăm realizarea aceleiaşi aplicaţii utilizând o bibliotecă de clase respectiv folosind doar apeluri de funcţii din API (Application Programming Interface).

Firmele care dezvoltă software orientat obiect au în atenţie crearea propriilor biblioteci de clase. Nu toate clasele realizate pentru un proiect vor ajunge în bibliotecă, unele fiind specifice proiectului şi neputând fi generalizate. Fiecare firmă îşi are politica proprie în ceea ce priveşte reutilizarea. Unele dezvoltă software **cu** reutilizare, integrând în aplicaţiile proprii clase existente în biblioteci, altele îşi canalizează eforturile înspre dezvoltarea de software **pentru** reutilizare, creând clase cu grade de generalitate şi abstractizare ridicate. De cele mai multe ori ambele faţete ale reutilizării se regăsesc în politica unei aceleiaşi companii de software.

În [KARL95] sunt sugerate câteva direcţii generale de urmat în dezvoltarea softwarelui orientat obiect cu o reutilizabilitate ridicată. Astfel se recomandă:

- Utilizarea moştenirii pentru a pune în evidenţă specializarea entităţilor din domeniul problemei de rezolvat. Aceasta conduce la crearea unui numar mare de nivele de abstractizare. Clasele vor avea dimensiuni mai mici, vor fi mai uşor de înţeles, modificat şi utilizat în alte sisteme decât cel pentru care au fost create.
- Utilizarea cât mai rar cu putinţă a moştenirii multiple.
- Ierarhia de clase să fie rezonabil de adâncă şi îngustă. De exemplu, dacă o superclasă are mai mult de zece subclase, este recomandabil să se introducă un nou nivel de abstractizare. Ierarhiile adânci au deza-

vantajul că funcționalitatea unei clase este dispersată și este mai greu de înțeles și urmărit.

- Divizarea sistemului în subsisteme trebuie să se facă astfel încât un subsistem să cuprindă toate clasele între care există o strânsă interdependență și care împreună concură la realizarea aceleiași funcții a sistemului. Astfel, clasele dintr-un subsistem se pot reutiliza împreună, ca un întreg.
- Interfața unei clase (protocolul) nu trebuie să aibă dimensiuni foarte mari. Clasele cu un număr mai mare de 25 de metode (proprii sau moștenite) trebuie să fie verificate și eventual modificate.

O politică de reutilizare eficientă trebuie să se bazeze pe aprecieri cantitative ce au la baza măsurători. De exemplu, măsurile utile pentru cuantificarea relației de moștenire sunt: numărul de metode nou adăugate (într-o subclasă), numărul de metode redefinite și numărul de metode moștenite. O ierarhie de clase se poate reprezenta prin intermediul unui arbore (cazul moștenirii simple), sau mai general, printr-un graf, pentru a surprinde și cazul moștenirii multiple. Nodurile grafului reprezintă clasele din ierarhie, iar arcele - relația de moștenire. Dacă *u* este nodul asociat clasei derivate și *v* nodul asociat clasei de baza, atunci convenim că există arcul (u,v). Se poate calcula adâncimea unei clase C_i în arborele de moștenire DIT (Depth of Inheritance Tree) după relația:

$$DIT_i = \begin{cases} 0, C_i \text{ nu moștenește de la altă clasă} \\ \dfrac{\sum_{j \text{părinte}} (1 + DIT_j)}{\sum_{j \text{părinte}} j} \end{cases}$$

unde DIT_i reprezintă adâncimea clasei i.

Pentru întreaga ierarhie este util să se determine adâncimea maximă a arborelui de moștenire. Tot la nivelul ierarhiei de clase se calculează adâncimea medie a unei clase în ierarhie ca fiind:

$$\overline{DIT} = \frac{\sum_{i=1}^{NC} DIT_i}{NC}$$

unde NC reprezintă numărul claselor din ierarhie.

Fiecare dintre metricile amintite are asociate valori prag, stabilite de către compania de software. Situarea valorilor obținute față de valorile prag furnizează informații pe baza cărora se iau decizii cu privire la: divizarea unei clase, introducerea unui nou nivel de abstractizare, introducerea unei clase în biblioteca etc.

6. Concluzii

Re-ingineria permite abordarea la același nivel a dezvoltării software și a întreținerii, utilizând tehnici și instrumente comune.

Efortul implicat în activitatea de re-inginerie depinde de obiectivele urmărite. O importanță deosebită o constituie exploatarea unor noi tehnologii în cadrul unor aplicații existente.

Bibliografie

[AYAC95] Ayache M., Ouhalima M., Menhoudj K. La re-inginerie: objectifs et techniques. *Software engineering and applications*, pg. 30-37, Tipografia ASE, Bucuresti, 1995.

[BANK94] Banker D.R, Kauffman J.R., Wright C., Zweig D. Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment. *IEEE Transactions on Software Engineering*, vol. 20, nr. 3, pg. 169-187, martie 1994.

[BARN91] Barnes H.B., Bollinger B.T., Making reuse cost-effective. *IEEE Software*, vol. 8, nr.1, pg. 13-24, ianuarie 1991.

[BALA96] Bălăceanu M. ObjectTeam for OMT. *PC-Report*, nr. 46, pg. 38-39, iulie 1996.

[DIFE93] Di Felice P. Reusability of mathematical software: a contribution. *IEEE Transactions on Software Engineering*, vol. 19, nr. 8, pg. 835-843, august 1993.

[FENT92] Fenton E.N. Software metrics. A rigorous approach. Chapman & Hall, London, 1992.

[FRAK96] Frakes W., Terry C. Software reuse: metrics and models. *ACM Computing Surveys*, vol. 28, nr. 2, pg. 415-435, iunie 1996.

[HEND96] Henderson-Sellers, B., Object Oriented Metrics: A Measure of Complexity. Prentice Hall, New Jersey, 1996.

[KARL95] Karlsson E.A. Software reuse. A holistic approach. John Wiley & Sons, Chichester, 1995.

[KUNZ95] Kunz T., Black J.P. Using automatic process clustering for design recovery and distributed debugging. *IEEE Transactions on Software Engineering*, vol. 21, nr. 6, pg. 515-527, iunie 1995.

[MILI95] Mili H., Mili F., Mili A., Reusing Software: Issues and Research Directions. *IEEE Transactions on Software Engineering*, vol. 21, nr. 6, pg. 528-562, iunie 1995.

[RAJL96] Rajlich V., Silva H.J. Evolution and reuse of orthogonal architecture. *IEEE Transactions on Software Engineering*, vol. 22, nr. 2, pg. 153-157, februarie 1996.

[SUCI96] Suciu D.M. O mică analiză. *PC-Report*, nr. 46, pg. 32-35, iulie 1996.

[****96] *** Reverse engineering newsletter. IEEE Computer Society/TCSE, vol. 14, nr. 2, 1996.

[ISO/CEI] *** Standardul SR ISO/CEI 9126 - Evaluarea produsului software - Caracteristici ale calității și linii directoare pentru utilizarea lor.