Reutilizabilitatea software

Ec. Emilia STANCIU, Banc Post

Software reuse is the process of building or assembling software applications and systems from previously developed software parts designed for reuse. The reusability is the degree to which an asset can be used in more than one software system, or in building other assets. Software reuse impacts on the software quality factors (functionality, reliability, usability, efficiency, maintainability, portability). A reusable component should be of high quality to inspire confidence in its potential reusers. A set of general rules can be applied in order to improve software reusability.
Keywords: *software reusability, software quality, object orientation.*

Reutilizabilitatea si calitatea software

Calitatea software este definita ca totalitatea trasaturilor si caracteristicilor unui produs software care se refera la capacitatea lui de a satisface necesitatile declarate sau implicate. Conform standardului ISO 9126, caracteristicile de calitate, prin care se poate descrie si evalua calitatea, sunt: *functionalitatea* (set de atribute bazate pe existenta unui set de functiuni si proprietatile lor specificate); *fiabilitatea* (set de atribute care se refera la capacitatea software de a mentine nivelul sau de performanta în conditii stabilite pentru o perioada data de timp); *utilizabilitatea* (set de atribute care se refera la efortul necesar pentru utilizare si la estimarea individuala a fiecarei utilizari, de catre un set de utilizatori declarati sau implicati); *eficienta* (set de atribute care se refera la relatia dintre nivelul de performanta al software si cantitatea de resurse utilizate, în conditiile stabilite); *mentenabilitatea* (set de atribute care se bazeaza pe efortul necesar pentru a face modificarile specificate); *portabilitatea* (set de atribute care se refera la capacitatea software de a fi transferat dintr-un mediu in altul).

O componenta reutilizabila trebuie sa fie de calitate ridicata pentru a inspira încredere potentialilor reutilizatori. Calitatea ridicata nu asigura însa implicit reutilizabilitatea componentei. Reutilizabilitatea este o combinatie a doua atribute: *(re)utilitatea* (componenta se adreseaza unei anumite cerinte) si *utilizabilitatea* (componenta este de calitate corespunzatoare, usor de înteles si de utilizat pentru dezvoltarea unui nou produs software).

Bowen [1] considera ca reutilizarea consta mai ales în adaptarea software-ului existent. Din acest punct de vedere proiectarea unei componente având ca factori de calitate portabilitatea, flexibilitatea si expandabilitatea asigura un nivel de reutilizabilitate mai ridicat. Matsumoto [2] a subliniat ca generalitatea, claritatea si usurinta întelegerii, portabilitatea si usurinta regasirii sunt caracteristicile majore care contribuie la reutilizabilitatea unei componente.

În cadrul modelului conceptual al calitatii McCall, factorii sunt grupati în trei categorii, reutilizabilitatea fiind inclusa în categoria "tranzitie produs" [3]:

- exploatare/utilizare: eficienta, corectitudine, integritate, utilizabilitate, fiabilitate;
- tranzitie produs: reutilizabilitate, portabilitate, interoperabilitate;
- revizie produs: mentenabilitate, flexibilitate, testabilitate.

Reutilizarea software are impact asupra caracteristicilor de calitate [4]:

1. *Functionalitatea*: Reutilizarea unor sisteme existente ca prototipuri permite definitivarea cerintelor functionale, atunci când utilizatorul nu le are clar definite în faza initiala. Evaluarea componentelor candidate la reutilizare, în cazul dezvoltarii cu reutilizare, permite clientului sa-si revizuiasca cerintele functionale. În dezvoltarea cu reutilizare sunt implicati mai multi

clienti, care se pot influenta reciproc si pot elucida într-o faza timpurie cererile functionale înca neidentificate. Reutilizarea unei componente ofera posibilitatea de a asigura o functionalitate suplimentara sistemului în care este inclusa. În dezvoltarea cu reutilizare trebuie realizat un compromis între solicitarile functionale ale mai multor clienti. Functionalitatea conflictuala trebuie eliminata, dar functionalitatea aditionala reprezinta un câstig.

2. *Fiabilitatea*: Reutilizarea corecta a unei componente bine testate mareste fiabilitatea sistemului; mai multe reutilizari ale unei componente maresc încrederea în acea componenta.

3. *Utilizabilitatea*: Realizarea unei componente reutilizabile necesita mai mult efort decât realizarea uneia cu o singura utilizare, ceea ce are un efect pozitiv asupra utilizabilitatii ei. Un aspect negativ îl constituie faptul ca acele componente care au caracteristici putin diferite sunt inacceptabile din punct de vedere al utilizabilitatii.

4. *Eficienta:* Realizarea unei componente reutilizabile eficiente necesita mai mult efort decât realizarea uneia cu o singura utilizare. Componentele care sunt realizate pentru a satisface cerinte specifice sunt mai eficiente decât componentele reutilizabile cu caracter mai general.

5. *Mentenabilitatea*: În cazul unei componente reutilizabile, care are un caracter general, multe din modificarile care ar putea fi necesare sunt deja încorporate sau planificate în cerintele extinse, în cazul dezvoltarii cu reutilizare. Modificarile suplimentare care trebuie facute unei componente reutilizabile sunt mai dificile, functionalitatea unei astfel de componente fiind mai complexa, dar costul modificarilor se va împarti între mai multi utilizatori.

6. *Portabilitatea*: Reutilizarea unei componente care nu este portabila poate compromite portabilitatea sistemului.

Cai de crestere a reutilizabilitatii software

Reutilizabilitatea poate fi introdusa în mod efectiv când o componenta este creata având reutilizabilitatea ca scop mai degraba decât atunci când este modificata ulterior pentru reutilizare. Introducerea caracteristicii de reutilizabilitate într-o componenta determina o marire a costului dezvoltarii. Pentru a crea caracteristica de reutilizabilitate pentru o componenta se poate aplica: generalizarea, standardizarea, automatizarea, certificarea, documentarea.

Dezvoltarea componentelor reutilizabile se poate face urmând etapele:

- Definirea unei solutii initiale si identificarea solutiilor anterioare la un set dat de cerinte;
- Identificarea unor posibile generalizari;
- Identificarea unor posibili reutilizatori si colectarea cererilor lor;
- Estimarea costului si beneficiilor functionalitatii adaugate;
- Propunerea unei solutii generalizate cu estimare de cost;
- Prezentarea solutiei reutilizatorilor si expertilor în reutilizare pentru validare si aprobare;
- Dezvoltarea si documentarea solutiei.

Rubin si Lim [5] au descris urmatoarele reguli generale pentru proiectarea codului în vederea obtinerii modularizarii si reutilizarii:

- Fiecare componenta trebuie sa efectueze o singura operatie logica.
- Componentele trebuie sa izoleze toate deciziile separate de proiectare. Acesta este un atribut important pentru portabilitate: diferenta de resurse dintre diferite sisteme trebuie sa necesite rescrierea unui numar cât mai mic de componente de interfata.
- Fiecare componenta trebuie sa interactioneze cu sistemul extern astfel încât implementarea sa interna sa poata fi reproiectata integral fara afectarea sistemului exterior.
- Componentele trebuie sa schimbe între ele doar minimum necesar de elemente de date.
- Daca o componenta produce erori, starea sistemului trebuie sa ramâna neschimbata (cu exceptia indicarii starii de eroare), adica sa pastreze starea de la apelarea componentei. În cazurile în care totusi sta-

rea sistemului se modifica, ea trebuie anuntata si explicata.
- Trebuie utilizate structurile de date corespunzatoare care sa permita o procesare eficienta.
- Trebuie utilizat un numar minim de structuri de date standard în interfata componentelor.
- Fiecare tip de structura de date din proiect trebuie manipulata de o componenta distincta.
- Trebuie înlocuite constantele cu parametri oriunde este posibil.
- Componentele trebuie astfel proiectate încât sa accepte un numar variabil de parametri.
- Componentele trebuie proiectate cu caracter general.
- Componentele trebuie însotite de o documentatie detaliata.
- Fiecare componenta trebuie sa fie corecta sub toate aspectele, pentru toate valorile parametrilor sai si în conditiile executiei în medii software necunoscute.

Cu toate ca abordarea orientata obiect faciliteaza reutilizarea mai mult decât alte metode de dezvoltare software datorita mecanismelor de încapsulare si mostenire, reutilizabilitatea nu se obtine automat în procesul de dezvoltare orientat obiect. Reutilizarea este o disciplina separata si trebuie privita ca fiind atât complementara cât si esentiala tehnologiei obiectuale.

McClure [6] propune urmatorul set de recomandari pentru crearea unei componente orientate obiect reutilizabile:

1. Aplicarea regulilor de module structurate metodelor sau operatiilor (sa reprezinte doar un concept sau o functie, sa aiba coeziune mare, sa respecte consistenta numelor, sa aiba cuplare redusa)
2. Separarea aplicatiei de implementare
3. Crearea de clase abstracte si utilizarea mostenirii pentru evitarea redundantei
4. Evitarea informatiei globale; practicarea ascunderii informatiei si încapsularii. Aceasta restrictioneaza utilizarea clasei doar la nivelul interfetei. Astfel implementarea clasei poate fi modificata fara afectarea utilizatorilor
5. Pastrarea unei adâncimi rezonabile a arborelui mostenirii pentru a asigura inteligibilitatea si o buna performanta. Prea multe nivele de mostenire implica un numar mare de obiecte în memorie la un moment dat. De asemenea, testarea este mai dificila pentru o arborescenta mare. Se recomanda sa nu se depaseasca cinci nivele în ierarhia claselor
6. Utilizarea subtipurilor pentru implementarea mostenirii. O subclasa trebuie sa extinda functionalitatea unei superclase. În cazul crearii unei subclase trebuie sa se ia în considerare urmatoarele reguli: cât mai putine metode supraîncarcate, cât mai putine metode adaugate în special la nivele inferioare ale ierarhiei, cât mai putine metode de stergere. Subtipul trebuie sa mosteneasca cât mai multe metode de la supertip
7. Evitarea mostenirii multiple deoarece face ierarhia clasei dificil de înteles si mai putin eficienta, necesitând mai multa memorie
8. Utilizarea cadrelor de lucru ca o forma controlata de reutilizare tip cutie alba, utilizatorul putând adauga subclase pentru completarea ei. Un cadru de lucru este un proiect reutilizabil al unui sistem sau subsistem. Se recomanda proiectarea cadrului de lucru astfel încât sa se adauge maxim sase nivele în cazul reutilizarii. De asemenea, este de preferat supraîncarcarea doar a metodelor vide pentru a controla modificarile
9. Pentru a mari reutilizabilitatea unei clase se recomanda:
   - generalizarea clasei pentru a avea o aplicabilitate mai mare (se includ doar operatiile esentiale);
   - declararea abstracta a clasei pentru a facilita reutilizarea metodelor si variabilelor prin mostenire. Trebuie minimizate atributele esentiale pentru clasa abstracta pentru a nu restrictiona subclasele la aceasta reprezentare;

- limitarea dimensiunii clasei (se recomanda maxim 10-25 metode per clasa);
- crearea abstracta a interfetei pentru a forta încapsularea;
- fortarea încapsularii pentru a realiza comunicarea între obiecte doar prin intermediul mesajelor;
- maximizarea coeziunii clasei; nu trebuie incluse în aceeasi clasa parti cu functionalitate diferita. Clasele reprezentând doua abstractizari trebuie separate în doua clase,
- minimizarea cuplarii claselor pentru a elimina dependentele între clase;
- documentarea completa a clasei.

10. Îmbunatatirea reutilizabilitatii unei metode prin:
- limitarea dimensiunii metodei, pentru a fi mai usor de înteles, testat si întretinut, ceea ce faciliteaza reutilizarea. Se recomanda limitarea la maxim 10 mesaje trimise, limitarea numarului de parametrii, reducerea complexitatii;
- generalizare prin extindere (generalizare tip argumente);
- maximizarea coeziunii si minimizarea cuplarii.
- modificarea codului obiect doar prin tehnici de mostenire;

11. Pentru crearea reutilizabila a unui model obiectual (care descrie structura unui domeniu si este util pentru crearea sistemelor în cadrul domeniului) se recomanda:
- generalizarea modelului prin includerea doar a claselor care sunt esentiale pentru întelegerea domeniului si generalizarea acestor clase;
- asigurarea completitudinii modelului;
- respectarea conventiilor pentru nume si alegerea unor nume semnificative pentru partile modelului.

Reutilizarea eficienta presupune activitati de reutilizare care sa faca parte inerenta din procesele ciclului de viata. Reutilizarea software ofera potential pentru obtinerea unor beneficii mari pe termen lung, în special când eforturile de reutilizare includ activitati initiale, de planificare din ciclului de viata, permitând dezvoltarea unor sisteme care partajeaza arhitecturi si elemente de proiectare comune.

Bibliografie

1. Waine C.L. Managing software reuse. Prentice Hall, Upper Saddle River, 1998.
2. Poulin J.S. Measuring software reuse. Addison-Wesley, Massachutes, 1997.
3. Ivan I. Objects quality characteristics. *Computer Science. The Proceedings of the 3rd International Symposium of Economic Informatics*, pg. 180-184, Editura INFOREC, Bucuresti, 1997.
4. Karlsson E.A. Software reuse. A holistic approach. John Wiley & Sons, Chichester, 1995.
5. Lim C.W. Managing software reuse. A comprehensive guide to strategically reengineering the organization for reusable components. Prentice Hall PTR, Upper Saddle River, 1998.
6. McClure C. Software reuse techniques. Prentice Hall, New Jersey, 1997.
7. McClure C. Software reuse: a standards based guide. IEEE Computer Society, Los Alamitos, 2001.