INGINERIE SOFTWARE

Refolosirea produselor software

Studentii:

Bestea Vladut, 442 A

Gradinaru-Tascau Gabriel, 442 A

Oprea Alin, 442 A

Perianu Razvan, 442 A

Pirvu Sorin, 442 A

Cuprins

Introducere (Perianu Razvan)

Ingineria software a parcurs o cale lungă începând cu 1968, an în care acest termen a fost utilizat pentru prima oară la o conferinţă NATO. Iar de atunci software-ul a pătruns în viaţa fiecăruia dintre noi în diverse moduri, aşa cum puţini anticipaseră chiar cu un deceniu în urmă. Aşadar cunoaşterea noţiunilor de bază legate de teoria şi practica ingineriei software este esenţială pentru înţelegerea tehnicilor de construire a unui software performant şi de asemenea a metodelor de evaluare a riscurilor şi oportunităţilor pe care software-ul le oferă vieţii noastre de zi cu zi.

În anul 1946 Goldstine şi von Neumann apreciau că 1000 de instrucţiuni reprezintă o limită superioară rezonabilă pentru complexitatea problemelor ce pot fi concepute ca rezolvabile cu ajutorul calculatorului. După ce a prevăzut că nici un program pentru calculatoare personale nu va necesita vreodată mai mult de 64 KB de memorie RAM, Bill Gates a admis în 1995 că lucrurile sau schimbat în ultimele două decenii.

Următoarele exemple oferă o imagine asupra gradului de complexitate la care au ajuns programele:

• Sistemul de rezervare a biletelor pentru compania aeriană KLM conţinea, în anul 1992, două milioane de linii de cod în limbaj de asamblare;

• Sistemul de operare System V versiunea 4.0 (UNIX) a fost obţinut prin compilarea a 3700000 linii de cod;

• Programele scrise pentru naveta spaţială NASA au circa 40 de milioane de linii de cod obiect;

• Pentru realizarea sistemului de operare IBM OS360 au fost necesari 5000 de ani-om.

Pentru a contracara ceea ce se prefigura ca fiind o criză a programării, a fost propus în anul 1968 termenul de “ingineria software” (software engineering), într-un mod oarecum provocator. Se dorea ca arta programării să împrumute din rigoarea stiinţelor inginereşti pentru a putea livra programe la timp şi în mod economic.

Reutilizarea resurselor software este un concept care în ultimii 20 de ani a devenit din ce in ce mai popular. Ideal, dezvoltatorii de produse software ar trebui să poată crea sisteme software de înaltă calitate prin asamblarea de componente software existente. În trecut accentul s-a pus mai mult pe reutilizarea (la scară mică) unor funcții individuale, a librăriilor de funcții și tipuri de date și a tehnologiilor independente de domeniu. Astfel de abordări s-au dovedit a fi benefice, dar nu exploatează la maxim conceptul de reutilizare software.

Reutilizarea resurselor software este ușor de aplicat atunci când componentele software sunt cunoscute foarte bine și realizează exact ceea ce este nevoie. Dar o componentă care face “aproape ceea ce trebuie” este de ce cele mai multe ori complet inutilă. Abordările moderne în ceea ce privește reutilizarea resurselor software, precum Liniile de Produse Software (Software Product Lines - SPL), oferă suport pentru reutilizarea la scară largă. Dezvoltarea bazată pe Liniile de Produse Software s-a dovedit a fi un mod eficient de a beneficia de reutilizarea și variația resurselor software. Acest tip de dezvoltare dacă este folosit corect duce la reducerea costurilor, la reducerea timpului de dezvoltarea și la creșterea productivității.

Dezvoltarea bazată pe SPL constă din dezvoltarea produselor software similare prin combinarea componentelor software comune cu cele specifice fiecărui produs, obținându-se astfel o variație a funcționalității oferite de componentele centrale. În sunt prezentate schematic două produse de tipul calculator, amândouă folosind ca și motor de calcule o componentă care implementează funcționalitatea unui calculator generic. Funcționalitatea care diferă este implementată de componente separate, de exemplu cele două calculatoare folosesc butoane diferite.

Pentru orice produs dintr-o linie de produse software, aproape toată funcționalitatea este implementată prin refolosirea de componente de bază. Aceste componente de bază implementează funcționalitate de bază care este uniformă pentru toate produsele din SPL și oferă suport pentru funcționalități care variază și care pot fi selectate de către fiecare produs. Punctele de variație ale componentelor de bază oferă o interfață prin care se poate alege funcționalitatea care variază. Pentru fiecare

produs sunt instanțiate anumite puncte de variație ale componentelor de baza, în plus se pot dezvolta noi componente specifice produsului.

Variația software are mai multe roluri atunci când se folosește dezvoltarea bazată pe SPL. Cel mai evident rol este acela de a oferi suport pentru diferențele de funcționalitate între produsele unui SPL. Variația software poate fi folosită și pentru a oferi suport pentru diferențe non-funcționale (scalabilitate, performanță sau securitate). Un alt rol mai puțin recunoscut este acela de a ajuta la rezolvarea conflictelor între modificările cerute pentru componentele de bază.

SPL nu este doar o problemă de arhitectură, design sau programare. SPL influențează întregul ciclu de viață al procesului de dezvoltare software.

Caracteristici de calitate (Perianu Razvan)

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 in 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’ului si cantitatea de resurse utilizate, in conditiile stabilite);
- mentenabilitatea(set de atribute care se bazeaza pe efortul necesar pentru a face modificari 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 incredere potentialilor reutilizatori. Calitatea ridicata nu asigura insa 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 inteles si de utilizat pentru dezvoltarea unui nou produs software).

Reutilizarea software are impact asupra caracteristicilor de calitate:

1. Functionalitatea: Reutilizarea unor sisteme existente ca prototipuri permite definitivarea cerintelor functionale, atunci cand utilizatorul nu le are clar definite in faza initiala. Evaluarea componentelor candidate la reutilizare, in cazul dezvoltarii cu reutilizare, permite clientului sa-si revizuiasca cerintele functionale. In dezvoltarea cu reutilizare sunt implicati mai multi clienti, care se pot influenta reciproc si pot elucida intr-o faza timpurie cererile functionale inca neidentificate. Reutilizarea unei componente ofera posibilitatea de a asigura o functionalitate suplimentara sistemului in care este inclusa. In dezvoltarea cu reutilizare trebuie realizat un compromis intre solicitarile functionale ale mai multor clienti. Functionalitatea conflictuala trebuie eliminata, dar functionalitatea aditionala reprezinta un castig.

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

3. Utilizabilitatea: Realizarea unei componente reutilizabile necesita mai mult efort decat realizarea uneia cu o singura utilizare, ceea ce are un efect pozitiv asupra utilizabilitatii ei. Un aspect negativ il 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 decat realizarea uneia cu o singura utilizare. Componentele care sunt realizate pentru a satisface cerinte specifice sunt mai eficiente decat componentele reutilizabile cu caracter mai general.

5. Mentenabilitatea: In cazul unei componente reutilizabile, care are un caracter general, multe din modificarile care ar putea fi necesare sunt deja incorporate sau planificate in cerintele extinse, in cazul dezvo ltarii 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 imparti intre mai multi utilizatori.

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

Arhitectura unei linii de produse (Oprea Alin)

Dezvoltarea bazată pe SPL folosește Arhitectura bazată pe Linii de Produse (Product Line Architecture - PLA). O arhitectura PLA asigură reutilizarea componentelor de baza ale unui SPL. Obiectivele unei arhitecturi PLA în ceea ce privește reutilizarea și variația sunt:

- suport sistematic pentru funcționalitate variată pre-planificată;
- posibilitatea produselor din SPL să aleagă cu ușurință opțiunile din funcționalitatea variată.

O arhitectură PLA îndeplinește aceste obiective utilizând o varietate de mecanisme tehnice care permit reutilizarea și varietatea.

Mecanisme care asigură reutilizarea (Oprea Alin)

Pentru a putea reutiliza componentele software dezvoltatorii software trebuie să:

- să găsească și să înțeleagă componentele software;
- să încorporeze componentele software în propriul context de dezvoltare;
- să folosească componentele invocând funcționalitatea acestora.

Găsirea și localizarea componentelor software. Inginerii software folosesc documentație API și manualele pentru a permite

reutilizarea librăriilor software. Pentru dezvoltarea bazată pe SPL se folosește documentarea procedurilor de utilizare și creare a instanțelor componentelor software care reprezintă nucleul SPL-ului.

Introducerea componentelor software în contextul de dezvoltare. După ce o componentă software a fost găsită un programator trebuie să o facă disponibilă pentru a putea fi utilizată. Există mai multe modalități prin care o componentă poate fi adusă într-un context de dezvoltare. Aceste modalități pot fi împărțite în categorii în funcție de momentul de timp când se realizează legătura cu componentele software reutilizabile. Principalele momente de timp la care se poate realiza legătura cu o anumită componentă sunt:

- în timpul programării – utilizând controlul versiunii pentru codul sursă;
- în timpul compilării – utilizând controlul versiunii pentru librării statice;
- în timpul linkeditării – se realizează pentru librăriile dinamice de către sistemul de operare sau mașina virtuală;
- în timpul rulării – se realizează de către tehnologii middleware sau prin mecanisme specifice aplicației de configurare și plug-in-uri dinamice.

Realizarea legăturii la momente de timp timpurii (în timpul programării sau compilării), facilitează utilizarea variației ad-hoc. Realizarea legăturii la momente de timp târzii (linkeditare sau în timpul rulării), întârzie legarea de o anumită soluție si facilitează obținerea de avantaje de pe urma variației sistematice. Arhitecturile PLA mature ale unui SPL tind să folosească legăturile târzii.

Invocarea componentelor software. Pentru a permite invocarea componentelor software limbajele de programare oferă mecanisme de apelare de tipul: procedură, funcție și metodă. Pentru sistemele software distribuite bazate pe standarde precum CORBA sau SOAP, sunt disponibile mecanisme care permit programatorilor să apeleze componente software care rulează pe alte mașini. Mecanismele de invocare disponibile pentru un SPL sunt cele disponibile în cazul dezvoltării clasice a unui sistem software.

Gestionarea configurației software pentru reutilizare (Oprea Alin)

Cele mai utilizate metode de legare a componentelor reutilizabile sunt cele din timpul programării și a compilării. Acest lucru face ca,

gestionarea configurației software (Software Configuration Management - SCM) să fie un proces critic pentru un SPL. SCM constă din controlul versiunii și controlul modificărilor.

SCM este mult mai complicat pentru dezvoltarea bazată pe SPL, acest lucru datorându-se parțial faptului că identificarea configurației este mai greu de realizat. Identificarea configurației este activitatea prin care sunt specificate numele, atributele și relațiile dintre configurații. În cazul dezvoltării normale configurația are o structură simplă, dar în cazul SPL, fiecare componentă care face parte din nucleul SPL-ului, fiecare componentă specializată și fiecare produs are o configurație care trebuie identificată, în plus trebuie specificate și relațiile dintre aceste configurații.

O posibilă abordare pentru managementul configurației software pentru un SPL constă în folosirea unei linii de dezvoltare pentru fiecare componentă de bază, respectiv pentru fiecare produs. Fiecare versiune de produs conține componentele specializate, precum și versiunile componentelor de bază. Sistemul de control al versiunii asigură faptul că, componentele de bază sunt doar citite și că ele nu sunt modificate în contextul unui produs. Totuși linia de dezvoltare a unui produs poate să folosească mai târziu o versiune mai nouă a componentei, care însă a fost dezvoltată în linia de dezvoltare a componentei.

Mecanisme care asigură varietatea (Pirvu Sorin)

Într-un SPL, componentele de bază oferă suport pentru funcționalitate variabilă prin intermediul punctelor de variație. Într-un SPL suportul pentru varietate este specificat chiar de la nivelul arhitecturii. Totuși, pot fi folosite și mecanisme de variație care sunt non-arhitecturale. În afară de mecanismele de variație de la nivelul arhitecturii există și mecanisme de variație la nivelul proiectării și la nivelul codului sursă. Aceste trei tipuri de variații nu sunt incompatibile, astfel la un moment dat se pot folosi atât mecanisme de variație la nivelul fișierelor cât și la nivelul arhitecturii.

Variație la nivelul arhitecturii. Mecanismele de variație la nivelul arhitecturii sunt strategii de proiectare la nivel superior care au scopul de a permite sistemului software să suporte un anumit set de funcționalități. Aceste strategii sunt slab legate de facilitățile oferite de un anumit limbaj de programare. De exemplu arhitecturile de platforme software și plug-in-uri.

Variație la nivelul proiectului. Granița dintre arhitectură și proiect nu este întotdeauna bine delimitată. Mecanisme de variație la nivelul proiectului sunt cele care sunt suportate direct prin facilitățile puse la dispoziție de limbajul de programare folosit. Mecanismele de variație la nivelul arhitecturii trebuie create prin programare, ele nu sunt suportate direct de limbajul de programare folosit. Exemple de mecanisme de variație la nivelul proiectului sunt: definirea de interfețe pentru componentele software care pot fi implementate în diferite moduri, suportul pentru moștenire și supra-încarcare permite obiectelor să conțină funcționalitate diferită care satisface clasele de bază.

Variație la nivelul fișierelor. Mediile de dezvoltare și limbajele de programare oferă mecanisme prin care se poate implementa variația la nivelul codului sursă. Astfel unele limbaje de programare oferă suport pentru compilare condiționată și macro definiții. De asemenea scripturile de compilare pot realiza variație logică sau fizică la nivelul fișierelor, acestea pot fi folosite pentru a varia funcționalitatea. 5

Variație la nivelul managementului configurației software. Principalul rol al managementului configurației software într-un SPL este acela de a oferi suport pentru refolosirea componentelor software prin identificarea și gestionarea versiunii unui produs și a componentelor care formează respectivul produs. Noi versiuni de produse nu trebuie obligatoriu să folosească cea mai nouă versiune a unei componente de bază. Managementul configurației software permite ca un produs să folosească orice versiune a unei componente de bază. Istoricul versiunilor și ramificarea versiunilor dintr-un sistem care gestionează configurația software, pot fi folosite pentru a reprezenta variația.

Specificatii (Pirvu Sorin)

O specificatie este un protocol sau un limbaj ce defineste un set de componente ierarhice de baza, fiecare avand o interfata ce detaliaza acea componenta. Componentle de baza pot reprezenta tipuri de date, subrutine sau obiecte (module). Alegerea abstractizarii interfeteleor este importanta si are un mare impact asupra reutilizarii si acceptarii softwareului sau specificatiilor. Este important sa se poate faca distingerea intre ce o specificatie recomanda si ce nu.Descrierea unei specificatii detaliate este o sarcina grea. O specificatie trebuie sa nu fie ambigua. Pentru fiecare intrare valida la o componenta, iesirea trebuie sa fie bine definita si documentata. Specificatiile sunt in general acompaniate de tutoriale, verificari sau exemple. O specificat in sinea ei

nu reprezinta in general un obiect interesant pentru prelucrare. O implementare software a specificatiei este de asemenea dorita. Documentatia ce insoteste o specificatie ar trebuie sa includa o vedere de ansamblu aspura consideratiilor de design. Acestea trebuie sa se concentreze asupra problemelor intalnite si alegerilor facute pentru a permite dezvoltatorilor sa compare situatia lor cu cea descrisa de documentatie. In plus, exemplele, rezultatul testelor si masuratori ale componentelor ar trebui incluse. Rezultatele performantelor sunt adesea un criteriu important pentru utilizatori sa se adapteze softului, in special daca exista soft cu functii similare. In cazul in care softul nu este de tipul “cutie neagra”, documentatia ar trebui sa ajute utilizatorii in adaptarea componentelor si ar trebui sa ajute persoanele ce realizeaza intretinerea in gasirea problemelor si a procesului de rezolvare. In final, documentele ar trebui sa includa cateva ghiduri de instalare, ghiduri, etc.

Librarii (Pirvu Sorin)

O specificatie este un protocol sau un limbaj ce defineste un set de componente ierarhice de baza, fiecare avand o interfata ce detaliaza acea componenta. Componentle de baza pot reprezenta tipuri de date, subrutine sau obiecte (module). Alegerea abstractizarii interfeteleor este importanta si are un mare impact asupra reutilizarii si acceptarii softwareului sau specificatiilor. Este important sa se poate faca distingerea intre ce o specificatie recomanda si ce nu.Descrierea unei specificatii detaliate este o sarcina grea. O specificatie trebuie sa nu fie ambigua. Pentru fiecare intrare valida la o componenta, iesirea trebuie sa fie bine definita si documentata. Specificatiile sunt in general acompaniate de tutoriale, verificari sau exemple. O specificat in sinea ei nu reprezinta in general un obiect interesant pentru prelucrare. O implementare software a specificatiei este de asemenea dorita. Documentatia ce insoteste o specificatie ar trebuie sa includa o vedere de ansamblu aspura consideratiilor de design. Acestea trebuie sa se concentreze asupra problemelor intalnite si alegerilor facute pentru a permite dezvoltatorilor sa compare situatia lor cu cea descrisa de documentatie. In plus, exemplele, rezultatul testelor si masuratori ale componentelor ar trebui incluse. Rezultatele performantelor sunt adesea un criteriu important pentru utilizatori sa se adapteze softului, in special daca exista soft cu functii similare. In cazul in care softul nu este de tipul

“cutie neagra”, documentatia ar trebui sa ajute utilizatorii in adaptarea componentelor si ar trebui sa ajute persoanele ce realizeaza intretinerea in gasirea problemelor si a procesului de rezolvare. In final, documentele ar trebui sa includa cateva ghiduri de instalare, ghiduri, etc.

Generarea codului (Pirvu Sorin)

Generarea codului este pocesul prin care anumite intstrumente semi-autmate sunt folosite pentru a transforma intrari de nivel inalt in iesiri de nivel jos. Iesirea generatorului de cod necesita adesea o procesare suplimentare inainte sa fie gata. Generarea automata a codului este dat de programarea generativa. Unul din principalele scopuri ale acesteia este inlocuirea cautarii manuale, adaptarea, implementarea si asamblarea componentelor cu versiuni generate automat ale lor facut la comanda. Programarea generativa se concentreaza asupra maririi automatizarii dezvoltarii aplicatiei : dand o specificatie de sistem, generatorii folosesc un set de componente reutilizabile pentru a genera sisteme concrete. Aceasta reprezinta designul si implementarea modulelor software care pot fi combinate pentru a genera sisteme specializate ce sunt bine optimizate cu necesitati minime. Scopurile sunt diverse : reducerea gaurii conceptuale dintre codul programului si conceptele de domeniu, atingerea unei reutilizabilitati si adaptabilitati inalte, simplificarea gestionarii componentelor, marirea eficientei pentru marimea codului si a timpului de executie si marirea productivitatii.

Programarea generativa inroduce notiunea de limbaj specific domeniului. Aceasta este un limbaj de programare sau de specificatii executabile ce ofera prin notatii corespunzatoare si abstractizari, puterea expresiva intr-un domeniu particular de problema.

Compilatoarele pentru limbajele de programare cum ar fii Fortran, C, C++ si Java sunt cateva din cei mai buni generatori de cod existenti. Ei nu numai ca genereaza fisierele executabile dar pot sa fie executati de o masina dar de asemenea sa evalueze parti mari ale codului sursa de nivel inalt si sa incerce reducerea lui intr-o implementare mai eficienta prin luarea in consideratie a contextului. Astfel, limbajul de programare poate tinti asupra mai multor probleme de domeniu, in timp ce scopul

gasirii implementarilor efiente este dat compilatorului. Generatoarele de cod sunt instrumente foarte puternice ce faciliteaza reutilizarea software. Cand sunt adaptate la domeniul aplicatiei, ele pot sa prelueze automat si sa adapteze codul dar si documentatia. Exemplele sunt asa numiti experti in mediile de dezvoltare pentru sistemele microcomputer, care produc sabloane functionale pentru interactivitatea incorporata bazata pe program.

Din pacate, dezvoltarea a astfel de generatoare este foarte grea si necesita cunostiinte si experienta atat in designul limbajului cat si in dezvoltarea compilatorului. Dezvoltarile recente cum ar fii tehnologiile bazate pe XML au facut programarea generativa mai accesibila dezvoltatorilor si asadar au radus la viata interesul pentru aceasta.

Open Source (Gradinaru Gabriel)

Prin open source, se prezinta unul din cele mai interesante trenduri din cadrul comunitatii software din ultima decada. Multi dezvoltatori de soft si organizatii considera acesta ca o cale de a da mai multa flexibilitate in cadrul dezvoltarii, realizand unele salturi in eforturile depuse aspura dezvoltarii prin reutilizarea de cod deja existent si de a primi acces la un grup mult mai mare de utilizatori de pe piata. Ca un rezultat al acestei activitati, acum exista o vasta cantitate de software reutilizabil disponivil dezvoltatorilor pentru folosire. Exista multe produse de tip open source care au devenit un bloc de baza in procesul de dezvoltare software. Astfel de prduse se concentreaza asupra produselor populare cum ar fii Linux.

Abordarea tip open source este poate cea mai de succes forma de reutilizare software la scara larga din ziua de azi. Softul open source este disponibil pentru multe activitati si este adesea suportat de mai multe platforme si poate fi folosit in mai multe domenii cum ar fii cel academic sau cel industrial. Imaginea publica a unui proiect de tip open source este adesea aceea a unei serii de actiuni haotice din care prin anumite miracole rezulta produsul software care se poate compila si executa. Dar realitatea este alta. Trendul de tip open source a evoluat inspre un proces de dezvoltare bine definit ce are nesitati particulare : coordonarea unui numar mare de dezvoltatori de cod pe Internet ce sunt

legati de interese si teluri comune. Abordarile de tip open source asupra dezvoltarii software au demonstrat ca software complex, limitat de timp poate fi dezvoltat prin distrubuirea lui unor echipe de dezvoltatori. Acest trend pare sa fie o colectie de idei diverse, cunostiinte, tehnici sau solurii. Cu toate acestea, toate activitatile sunt motivate de recunoasterea beneficiilor mai largi si accesul deschis la procesul de dezvoltare software. Elementele procesului de dezvoltare software de tip open source include largirea notiunii de echipa a unui proiect, marirea colaborarilor, eliberarea frecventa de variante soft si creearea de cod sursa disponibil pentru adaptare si reutilizare. Abordarea de tip open source este adesea o strategie planificata pentru a promova un produs popular grativ, de a profita o retea informala larga de dezvoltarori si utilizatori de acesta, si de a micsora folosirea resurselor interne de dezvoltare. Pe langa rezultatul concret sub forma de software, exista alte doua importante aspecte pentru open source :

- modeului business – prin facerea cadou unei parti a prorietatatii intelectuale, se obtine accesul la o paiata mai mare sau la ghiduri de baza. Cu toate acestea, inaintea ca o companie sa foloseasca un model open source pentru dezvoltare unor produse, vor trebui luati in considerare anumiti factori. Acestia includ de exemplu potentialul ca acel produs soft sa devina un model cheie pentru altii.

- procesul de dezvoltare – caractesiticle includ incurajarea unei mari comunitati de utilizatori, testeri si dezvoltatori comunicari frecvente si riguroase cu dezvoltatorii, eliberari software frecvente, coordonari puternice ale proiectului, responsabilitate si transparenta in procesul de dezvoltare.

Multi cred ca open source-ul este o problema legata de licenta dar multi mai mult de aceasta :

- eliberarile de cod open source extinde bazele de utilizatori si incurajeaza participarilor de tip third party ; de asemenea mareste aptitudinile dezvoltatorilor software.
- open source-ul mareste reutilizabilitatea codului

- open source-ul permite eliberari frecvente ale sistemelor software in linie cu asteptarile utilizatorilor pentru o mai mare receptivitate si pentru a micsora timpii cilcului de dezvoltare
- open source-ul se leaga de aspecte etice ale software-ului care sunt privite la libertatea folosirii si a participarii

Sistemul de operare open source, Linux, a capatat un impuls mare in ultimii zece ani. Aceasta a fost din cauza asemanarii cu sismele de operare bazate pe UNIX dar de asemenea din cauza lipsei fiabilitatii si vitezei platformei Microsoft Windows si a esuarii altor comercianti de a oferii alternative la costuri mai mici. Ca rezultat, multe organizatii comerciale cauta acum sa desfasoare solutii asupra unei platforme anume continand un numar de produse open source. In cadrul unei setari comerciale, licenta este o consideratie majora cand sau nu se doreste folosirea softului de tip open source. Licentele open source sunt in general simple dar implicatiile lor asupra unui sistem ce refoloseste acel soft pot fi profunde. In concordanta cu multe licente open source, reutilizarea software-ului de catre sistem trebuie sa fie si ea considerata open source si sa fie din cadrul acelorasi termeni de licentiere. Dezvoltarea software cu un potential comercial sau obiectiv este asadar un risc dat de incalcari a unor clause a unor licente.

Fiabilitatea si securitatea softului sunt alte consideratii majore ce trebuie luate in vedere cand se dorese folosiea softului de tip open source. Aceste consideratii sunt adesea motivate de ingrijorari asupra calitatii softului, cum a fost dezvoltat si in particular de care sunt originile acelui soft sau a partilor din el. Merita sa se tina minte ca softul open source este foarte probabil sa fie mai bine testat si examinat decat orice alt cod comercial proprietar.

Folosirea maxima de software open source si dependenta minima de tehnologia proprietara este in ziua de azi adesea considerata un pas mare in asigurarea portabilitatii unei aplicatii. Un parametru necunoscu pentru softul priprietar este adesea cand libraria tip closed-source sau intrumentul sau generatorul de cod sau protocolul de retea nu mai este suportat sau cand interfata va fi modificata intr-o maniera astfel incat sa fie compativila cu produse anterioare. Prin cod open source, se are intotdeauna accesul asupra codului sursa. Interfetele versiunilor de varf

ale softului open source se pot schimba dar prin faptul ca se are acces asupra codului sursa, faciliteaza modificarea si portarea aplicatiei.

Metode de reutilizare

Urmatoarele metode se folosesc de obiective, atribute cuantificabile ale software-ului ca o baza pentru reutilizabilitate. Majoritatea folosesc atribute orientate pe module dar metodele de inerpretare a atributelor variaza destul de mult.

1. Metoda Prieto-Diaz si Freeman (Oprea Alin)

Prieto-Diaz si Freeman identifica cinci atribute ale programelor pentru evaluarea reutilizabilitatii. Modelul lor incurajeaza refolosirea de tip “cutie alba” si presupune gasirea modulelor ce pot fi reutilizate, evaluandu-l pe fiecare, decizand care modul il poate modifica mai usor programatorul si mai apoi adaptarea moduluilui. In acest model se identifica patru metrici orientate pe module o a cincea metrica folosita pentru modificarea primelor patru. Urmatarea lista arata cele cinci metrici si o descriere a fiecareia:

1. Marimea programului. Reutilizarea depinde de o dimensiune mica a modulului, dupa cum este indicat de liniile codului sursa.
2. Structura programului. Reutilizarea depinde de o structura simpla a structurii programului cum este idicat de cateva legaturi catre alte module.

3. Documentatia programului. Reutilizarea presupune ca documentatia sa fie de calitate foarte buna.
4. Limbajul de programare. Reutilizarea depinde de limbajul de programare, de extensia care il ajuta sa refoloseasca un modul scris in acelasi limbaj de programare. Daca nu exista unmodul reutilizabil in acelasi limbaj de programare, gradul de asemanari intre limbajul ce se doreste a fi utilizat si cel in care a fost scris modulul afecteaza dificultatea de modificare a lui pentru a putea fi folosit in cadrul limbajului dorit.
5. Experienta in reutilizare. Experienta reutilizarii intr-un limbaj de programare si in domeniul aplicatiei afecteaza punctele de mai sus deoarece fiecare programator vede un modul din propria perspectiva. De exemplu, programatorii vor avea viziuni diferite a ceea ce face ca un modul mic sa depinda de cunostiintele lor. Aceasta metrica se foloseste pentru modificarea valorilor celorlalte metrici.

2. Metoda Selby (Pirvu Sorin)

Acesta ofera un stiudiu statistic al caracteristicilor de reutilizabilitate folosind date de la mediul software de la NASA. NASA a folosit mediul de dezvoltare pentru a produce soft scris in Fortran ce realizeaza suport pe teren pentru controlarea navelo fara pilot. Studiul pune la dispozitie dovezi statistice bazate pe analiza non parametrica a variantei asupra contributiilor a unei game mari de caracteristici ale codului. Studiul a validat maijoritatea clauzelor de mai jos aratand ca majoritatea modulelor s-au refolosit fara modificare :

- Sa fie de dimensiuni mici, in general mai putin de 140 de linii de cod.
- Sa aibe interfete simple
- Sa aibe pune apelari ale altor module.
- Sa aibe mai multe apelari de sisteme de nivel jos si a functiilor utilitare.
- Sa aibe putini parametri de intrare si iesire.
- Sa aibe putina interactivitate cu utilizatorul.
- Sa aibe o buna documentatie.
- Sa aibe nu fie modificata de multe ori in timpul implementarii.
- Sa aibe mai multe comenzi de asignare decat logice pe linie sursa.

- Sa nu aibe neaparat o complexitate mica a codului.
- Sa nu depinda de marimea proiectului.

3. Metoda Chen si Lee (Perianu Razvan)

Chen si Lee au dezvoltat in jurul a 130 de componente reutilizabile in C++ si au folosit aceste componente intr-un mediu controlat pentru a masura nivelul de reutilizabilitate a unui program a productivitatii software si a calitatii. In opozitie de Selby, care a lucrat cu programatori profesionisti, experimenul lui Chen si Lee a implicat studenti care au trebuit sa dezvolte si sa implementeze un mic sistem de baze de date. Metricile software adunate includeau, marimea programului, nivelul programului, dificultate estimata si efortul. Au aflat astfel ca, cu cat sunt mai mici valorile pentru aceste metrici, cu atat este mai mare productivitatea progmarii.

4. Metoda Caldiera si Basili (Gradinaru Gabriel)

Acestia au afirmat ca atributele de baza ale reutilizabilitatii depind de calitatea corectitudinii, citirii, testarii, usurintelor la modificare si performantei dar ei au inteles ca nu se pot masura sau prezice majoritatea acestor atribute in mod direct. Asadar au fost date patru masuri ale reutilizabilitatii bazate in mate pe metricile date de McCabe si Halstead. Aceasta abordare orientata pe modul are avantajul ca instrumentele pot calcula automat toate cele patru metrici si o gama de valori pentru fiecare:

1. Volumul programului al lui Halstead. Un modul trebuie sa contina indeajunse functii pentru a justifica costul de regasire si integrare a lui dar nu atat de multe functii incat sa poata fi pusa in pericol calitatea.
2. Complexitatea cyclomatica a lui McCabe. La fel ca si cea mai sus valoripe acceptabile pentru aceasta metrica trebuie sa reprezinte o balanta intre cost si calitate.
3. Regularitate. Aceasta masoara lizibilitatea si non-redundanta implementarii unui modul prin compararea valurii actuale cu valorile estimate ale celor doua metricilor de lungime ale lui

Halstead. Un modul scris clar va avea o marime Halstead apropiata de cea teoretica.

4. Frecven de refolosire. Freventa de refolosire indica utilitatea unui modul si este data de numarul de apelari static ale modulului.

5. Metoda Reboot (Gradinaru Gabriel)

Prin REBOOT (Reuse Based on Object-Techniques) acest proiect s-a dezvoltat o clasificare a atributelor reutilizabilitati. Ca parte a acestei clasificari, s-au listat patru factori de reutilizabilitate, o lista de criterii pentru fiecare factor si un set de metrici pentru fiecare criterie. Cu toate acestea, cateva din aceste metrici depind de articole subiective cum ar fi listele de verificare, un analist poate calcula in mod direct multe din aceste metrici din cod, cum ar fi complexistatea sau raportul comentariu – cod sursa. Analistul combina valuri individuale ale metricilor intr-o valoare per total pentru reutilizabilitate. Urmatoarea lista defeineste patrfu factori de reutilizabilitate ; tabelul da criteriul si metricile pentru fiecare factor.

- Portabilitatatea. Usurinta cu care cineva poate transfera softul de pe un systen pe altul. Criteriul include :
  - Modularitatea
  - Dependenta de mediu
- Flexibilitatea. Numarul de alegeri pe care programatorul le are in determinarea folosirii unei anumite componente ; de asemenea se mai cunoaste sub numele de generalitate. Criteriul mai include :
  - Generalitate
  - Modularitate
- Inteligibilitate. Usurinta cu care un programator poate intelege componenta. Criteriul include :
  - Complexitatea codului
  - Auto descriptivitatea
  - Calitatea documentatiei
  - Complexitatea modulului

- Increderea de sine. Probabilitatea subiectiva ca o componenta se va desfasura fara probleme intr-o anumita perioada de timp intr-un mediu nou. Criteriu include :
    - Complexitatea modulului
    - Fiabilitatea observata
    - Toleranta erorilor

6. Metoda Hislop (Bestea Vladut)

Hislop presupune trei abordari pentru evaluare ; functie, forma si asemanare. Evaluare softului pe functie ajuta la selectarea componentelor bazate pe ceea ce fac ele. De fapt, multe colectii folosesc o organizare ierarhica bazata pe functie. A doua abordare, forma, caracterizeaza softul bazat pe caracteristici observabile cum ar fi marimea sau structura. Aceasta abordare se preteaza binela instrumentele de analiza a codului ce se raporteaza la numarul de declaratii in codul sursa sau complexitatea codului. A treia abordare, compara modulele si le grupeaza bazandu-se pe atribute comune. Acest tip de analiza ajuta in studiul reutilizabilitatii printr-un numar de cai. Prima, un instrument ce poate identifica potentiale module reutilizabile in softul deja existent prin gasirea modulelor cu o metrica de asmanare apropiata de acelora a modulelor ce au fost reutilizate fara probleme. A doua, identificarea grupurilor de module similare poate ajuta in analiza domeniului prin identificarea oportunitatilor de reutilizare. A treia presupune automatizarea metricilor de reutilizare prin localizarea instantelor de reutilizare informala cu sau fara modificari. Instrumentul prototip al lui Hislop, este format din colectorul de date si analizorul de date. Colectorul parseaza software-ul si calculeaza masurile de forma pentru fiecare modul. Analizorul calculeaza asemanarile bazate pe o varietate de metrici a formelor cum ar fii complexitatea McCabe si metrici de structura a profilului.

7. Metoda Boetticher si Eichmann (Bestea Vladut)

Recunoasterea dificultatii definirii a ceea ce oamenii par sa accepte ca o notiune intuitiva a reutilizabilitatii, Boetticher si Eochmann au facut o abordare diferita a metricilor de reutilizabilitate. Ceea ce ei au facut se bazeaza pe antrenarea retelelor neurale pentru a simula un set de evaluatori umani. Au facut experimente care au variat mai multe

configuratii de retele neurale fiecare abilitate a retelei de a potrivi iesirea data de un grup de patru oameni. Parametrul de intrare selectat contine parametrii de cod reprezentand complexitatea modulului, adaptabilitatea si asocierea. Experimentul s-a desfasurat in trei faze, folosind parametrii de intrare signifianti in refolosirea de tip cutie neagra, cutie alba, si cutie gri. Parametrii tipului cutie neagra includ atribuiri ale codului, marime fizica, marime a fisierului si un numar de intrari si iesiri. Parametrii tipului cutie alba au inclus volumul Halstead, complexitatea ciclomatica, asemanarea si marimea. Metricile cutiei gri au combinat intrarile cutiei negre si albe. Experimentul s-a straduit sa realizeze o potrivire cat mai buna prin analiza sensibilitatii asupra parametrilor selectati pentru vectorii de antrenare, configuratia retelelor neurale si extensia antrenarii retelei. Cu toate ca rezultatele cutiei negre si albe au nu prezentat legaturi asupra iesirilor asteptate, cel de cutie gri s-a legat destul de bine.

8. Metoda Torres si Samadzadeh (Bestea Vladut)

Torres si Samadzadeh au facut un studiu pentru a determina corelatia dintre metricile de informatie teoretice si metricile de reutilizabilitate. Informatia teoretica masoara continutul de informatie ca nivelul entropiei intr-un sistem. Acest studiu examineaza efectele a doua metrici de informare teoretice, incarcarea entropiei si entropia structurii de control, in cadrul reutilizabilitatii software. Entropia incarcarii reflecta gradul de comunicatie inter proces. Cum cantitaea de comunicatie necesara intre partile a oricarui sistem mareste entropia, informatia teoretica cauta cai de a o reduce prin designul sistemelor cu o comuncare minima intre sub sisteme. Aplicand acest concept in software, cercetatorii au masurat cantitatea de comuncatie necesara dintre module si au atribuit o valoare pentru entopia incarcarii pentru fiecare modul. Entropia incarcarii corespunde conceptelor dezvoltarii software a legarii si coerentei ; programele care poseda mici valori pentru entropia incarcarii ar trebui de asemenea sa expuna proprietatile ce consista intra-o buna structura a programului si reutilizabilitate. Entropia stucturii de control cauta sa masoare complexitatea structurii logice a unui modul, ce este data de numarul de atribuirii if-then din

modul. Un experiment ce calcula cele doua metrici de informare teoretice asupra a sase programe a rezultat ca o entropia mare a incarcarii a avut un efect negativ asupra reutilizabilitatii. Consecvent, aceste metrici ar putea ajuta la selectarea cazului optim de reutilizare dintre diferitele optiuni.

9. Metoda Mayobre (Bestea Vladut)

Pentru a ajuta la identificarea produselor reutilizabile in codul existent, Mayobre a descris o metoda denumita Analiza Reutilizabilitatii Codului (CRA). CRA combina trei aprecieri ale reutilizabilitatii si un model de estimare economic pentru a identifica componente reutilizabile, rezultand astfel o combinatie de analiza automatica si analiza experta pentru a determina reutilizabilitatea tehnica si valoarea economica. CRA foloseste metoda CalDiera si Basili ca una din cele trei metode. A doua metoda, numita Procesul de Identificare a Componentelor Bazat pe Experienta in Domeniu (DEBCIP), depinde de analiza extinsa a domeniului a zonei probleie si foloseste un graf de decizie pentru a ajuta expertii in domeniu la identificarea componentelor reutilizabile. Prima iesire a DEBCIP da un estimat al numarului asteptat de instante de reutilizare al unei componente. A treia metoda numita Proceul de Identificare al Componentelor Bazat pe Analiza Variantei (VABCIP), foloseste de asemenea cunostiinte asupra domeniului dar la un grad mai mic. Aceasta foloseste metrica complexitatii ciclomate pentru estimarea distantei specicatiilor dintre modulul existent si cel necesar, dand un estimat al efortului necesar pentru modificarea componentei. Ultimul pas al analizei reutilizabilitatii consista in estimarea profitului intors din investitii, un proces ce realieaza compararea costului estimat al reutilizarii cu beneficiile asteptate.

10. Metoda NATO (Bestea Vladut)

Standarul NATO pentru procesurile de reutilizare software recomanda urmarirea celor patru metrici ca indicatori a calitatii software si a reutilizabilitatii :

- Numarul de inspectii. Numarul de ori cand cineva a considerat modulul pentru refolosire.

- Numarul de reutilizari. Numarul de ori cand cineva a reutilizat acel modul.
- Complexitatea. Complexitatea codului bazata pe metrica complexitatii McCabe.
- Numarul de probleme reportate. Numarul de defecte gasite asupra modulului.

Standarul sugereaza ca aceste metrici prezinta un bun estimat al reutilizabilitatii a unei componente is pot elimina optiunile nepotrivite. De exemplu, potentialii reutilizatori ar treb sa cuate o componenta cu numar mai are al reutilizarilor dar sa ramana sceptici cu privire la compontele cu un numar mare de inspectii si un numar mic de reutilizari. Standarul recomanda reutilizarea componentelor cu o valoare mica a complexitatii si a defectelor cunoscute.

Concluzii (Gradinaru Gabriel)

Calitatea software este obiectivul central al procesului de dezvoltare software. Pentru aceasta, numeroase cercetari au evidentiat o serie de caracteristici de calitate ce descriu in totalitate performantele produsului program si de care depinde si reutilizabilitatea acestuia.

In conditiile in care societatea depinde tot mai mult de aplicatiile informatice, consumatorul a capatat abilitatea de a diferentia produsele cu nivel ridicat la calitatii de produsele predispuse la erori si care nu ii satisfac nevoile legate de:

- obtinerea rezultatelor dorite;
- caracter general de rezolvare a problemelor;
- usurinta in utilizare si intelegere;
- instrumentele puse la dispozitie;
- prelucrarea datelor intre limite stabilite;
- probabilitatea de blocare a executiei si pierderea informatiei;
- resursele hardware si software minime.

Calitatea produsului software a devenit o conditie esentiala de a ramane pe piata. De altfel si reutilizabilitatea produsului a devenit un factor important, luat in calcul in procesul de dezvoltare al produsului software.

Pentru a invinge concurenta produsele trebuie sa coste putin, sa aiba calitate cat mai mare si sa fie realizate in cel mai scurt timp. Acest obiectiv este atins doar printr-o abordare organizata, planificata a procesului de determinare si implementare a caracteristicilor software de calitate.

Bibliografie

- http://www.biglever.com/papers/Krueger_AcmReuseSurvey.pdf
- Ian Gorton. Essential Software Architecture. Editura Springer. 2006.
- Waine C.L. Managing software reuse. Prentice Hall, Upper Saddle River, 1998.
- Poulin J.S. Measuring software reuse. Addison-Wesley, Massachutes, 1997.
- Karlsson E.A. Software reuse. A holistic approach. John Wiley & Sons, Chichester, 1995.
- McClure C. Software reuse techniques. Prentice Hall, New Jersey, 1997.
- McClure C. Software reuse: a standards based guide. IEEE Computer Society, Los Alamitos, 2001.