Automatic testing of specialized software products

Prof.dr. Ion IVAN, lect.dr. Paul POCATILU
Catedra de Informatică Economică, A.S.E. Bucureşti

In this paper is defined the automated testing concept for special applications and examples are given for software that solves linear equation systems. The proposed method allows analyzing the quality of the software results.
***Keywords**: software testing, automated testing, test data generators, test cases.*

Testarea software

Testarea este o etapă deosebit de importantă din ciclul de realizare a produselor software. Obiectivul testării este de a evidenția concordanța dintre produsul software finit şi specificaţiile de programare. Există mai multe metode de testare, dintre care cele care au la bază experiența, structura produsului finit se dovedesc a fi cele mai eficiente.

Testarea simbolică, testarea folosind metode care se bazează pe acoperirea arborescenței asociată produsului şi testarea orientată pe structura datelor definite şi referite, au ca obiectiv scoaterea în evidență a fluxurilor de prelucrare şi mai ales a capacitații produsului de a dezvolta prelucrări care să conducă la rezultate intermediare acceptate de specificații. În dezvoltarea unui produs software testarea se realizează pe mai multe niveluri: testarea de module, testarea de integrare, testarea de sistem şi testarea de acceptare (validare). În figura 1 sunt prezentate aceste niveluri ale testării [TEST90].

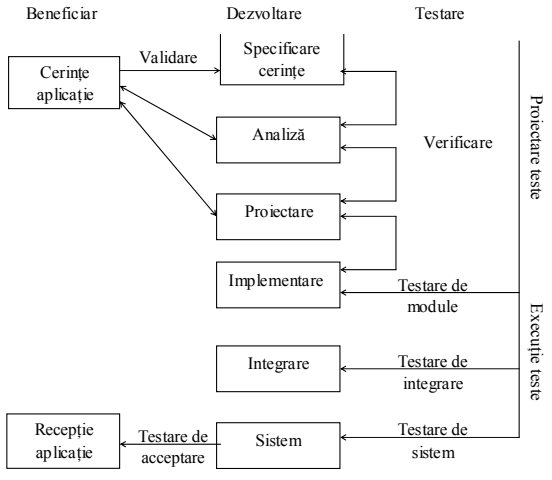

Fig. 1. Nivelurile testării software

Testarea programelor se realizează static şi dinamic. Testarea statică are în vedere analiza codului sursă a programului fără rularea acestuia. Testarea dinamică presupune examinarea aplicațiilor software în scopul generării datelor de test şi execuția acestora cu seturile de date de test obținute.

Există două strategii de dezvoltare a cazurilor de test: o strategie bazată pe structura programelor şi o alta, bazată pe funcționalitatea acestora. Testarea structurală (*white-box testing*) este o strategie de testare care necesită accesul la codul sursă şi la structura programului şi pune accentul pe acoperirea prin teste a căilor, ramificațiilor şi fluxurilor programului. Principalele metode de testare structurală au în vedere gradul în care cazurile de test acoperă sau execută codul sursă al programului.

Cele mai importante tehnici de testare funcțională (*black-box testing*) sunt testarea cu intrări aleatoare, partiționarea pe clase de echivalențe, analiza valorilor limită, graful cauză-efect şi ghicirea erorilor.

Testarea software se realizează de personal calificat - testerii- care au menirea de a prelua exemplele de date de test din specificații dar şi de a construi exemple de date de test proprii. Testerii, au capacitatea de a lansa în execuție totală sau parțială produsul software şi de a urmări exact comportamentul. De asemenea, realizează înregistrări de durate, de rezultate intermediare, de paşi executați. Rapoartele de testare sunt construcții care vizează esența unui produs program, ca întreg sau a componentelor acestuia.

În contextul creşterii complexității software, este dificil să se dezvolte procese de testare care să conducă la concluzii ferme în legătură

cu calitatea produsului finit. Tehnicile şi metodele avansate sau empirice de testare lasă un loc larg activităţii testerilor, artei acestora de a identifica punctele tari şi punctele slabe ale produselor program.

Testarea software implică un volum mare de muncă şi deci costuri care afectează eficienţa întregului proces de dezvoltare software. În cazul în care resursele alocate procesului de testare sunt insuficiente, se produce o analiză incompletă a produsului, concluziile sunt eronate şi efectele pe termen lung sunt însoţite de costuri foarte mari de remediere a erorilor de execuţie la fiecare client în parte.

Generatoare de date de test

Problema testării este prin importanţa ei deosebit de costisitoare în ceea ce priveşte datele de test. Construirea manuală a datelor de test nu este dificilă. Rezolvarea manuală, cu grad ridicat de rigurozitate a rezultatelor finale sau intermediare este însoţită de riscuri majore. Sunt numeroase situaţiile în care rezultatele corespunzătoare iteraţiilor s-au dovedit a fi incorecte. De asemenea, chiar în cazuri de reverificare, unele rezultate, considerate iniţial corecte s-au dovedit a fi incorecte.

Generatoarele de date de test vizează definirea unor mecanisme care să conducă la obţinerea de fişiere sau a unor seturi de date care se bucură de anumite proprietăţi şi care devin intrări în produse software care se supun procesului de testare.

Generatoarele de fişiere, sunt programe care primesc la intrare: structura pe câmpuri a articolelor unui fişier; pentru fiecare câmp se indica tipul conţinutului; pentru fiecare câmp se indica domeniul care este un interval, o enumerare de cuvinte sau simboluri dintr-un alfabet; dimensiunea fişierului ca număr de articole.

Va rezulta la ieşire un fişier ale cărui articole conţin câmpuri iniţializate cu valori generate după reguli folosind generatoare de numere pseudo-aleatoare, cuprinse în intervalele date sau care sunt poziţii ale elementelor din mulţimea de simboluri sau de cuvinte.

Generatoarele de matrice, sunt de asemenea programe care primesc la intrare:

- dimensiunea matricei;
- tipul elementelor;
- intervalele cărora trebuie să aparţină elementele sau mulţimea de cuvinte sau mulţimea de simboluri;
- caracteristici particulare ale matricei: matrice simetrică, matrice bandă, matrice pozitiv definită etc.

La ieşire se obţine o matrice salvată într-un fişier.

Generatoarele de date de test sunt programe care asigură proprietatea de reproductibilitate a fişierelor sau matricelor generate. Generatoarele de date de test pornind de la structura programului presupun citirea textului sursă a programului şi identificarea structurii arborescente asociată acestuia. Folosind structura arborescentă se produce generarea de date de test care să activeze toate ramurile arborescenţei, în aşa fel încât să se obţină atingerea tuturor frunzelor acestei arborescenţe. Generatoarele de date de test servesc la popularea fişierelor şi bazelor de date în vederea testării. Popularea se face fie cu date aleatoare, fie cu date obţinute prin specificarea unor condiţii. Aceste instrumente se utilizează în general pentru volume mari de date necesare testărilor operaţionale şi la capacitate maximă. În activitatea de cercetare a fost realizat un generator de date de test pentru fişiere. Pe baza descrierii tipurilor şi a domeniilor câmpurilor şi a numărului de articole de pentru test, se generează fişiere de test corespunzătoare.

Generatoare de mecanisme de testare

Testarea automată este caracteristica unei etape superioare a producţiei de software. Testarea automată presupune:

- existenţa unui produs software complex care să dezvolte astfel de procese;
- definirea la intrare a programului care face obiectul testării;
- realizarea unei varietăţi de mecanisme de testare care să acopere o gamă cât mai variată de programe;
- identificarea de criterii de evaluare a rezultatului testării care să stea la baza revizuirilor ulterioare;
- prezentarea de procese ale prelucrărilor inverse care să permită verificarea în mod independent a calităţii soluţiilor produsului testat.

Mecanismele de testare vizează:
- numărul seturilor de date de test;
- diversitatea seturilor de date de test;
- soluțiile inițiale, exacte, recunoscute ca fiind corecte;
- definirea sistemului de generare a datelor de bază;
- recrearea de seturi de date derivate care se concatenează cu datele de bază şi devin în totalitate componentele seturilor de date;
- integrarea în procesul de testare automată a programului care face obiectul testării;
- includerea unui sistem de indicatori de apreciere a rezultatului testării;
- agregarea de indicatori privind comportamentul de ansamblu a procesului de testare automată.

Testarea automata este un deziderat. Maximizarea gradului de generalitate trebuie inclusă în categoria dezideratelor majore. Daca se începe cu testarea automată a programelor aparținând unei clase de aplicații se obțin concluzii valoroase care stau la baza trecerii la software cu grad de complexitate ridicat. Prin paşi succesivi se creează premisele favorabile fundamentării unei teorii coerente a testării automate. Creşterea gradului de generalitate este o chestiune legată de experiența acumulata în procesul testării, coroborată cu utilizarea unor tehnici şi metode evoluate de extragere a informațiilor dintr-un text sursa devenit intrare pentru un software orientat pe testare automată.

Software pentru rezolvarea sistemelor de ecuații liniare

Una dintre problemele cu larga aplicabilitate în economie este crearea de software destinat rezolvării sistemelor de ecuații liniare, de dimensiuni mari. Balanța legăturilor dintre ramuri şi toate modelele asociate acestei tipologii de aplicații, necesită sisteme de programe deosebit de performante care să conducă la soluționarea de sisteme de ecuații liniare cu sute de necunoscute, respectiv, de ecuații.

Metodele analizei numerice includ numeroşi algoritmi destinați soluționării unei astfel de probleme. Algoritmii exacți, în condițiile creşterii dimensiunii problemei de rezolvat, datorită caracterului finit a regiştrilor unui sistem de calcul şi a modului în care se realizează reprezentarea numerelor reale, sunt însoțiți de rezultate perturbate de erori a căror mărime se estimează şi care, generează inexactități în ceea ce priveşte probele efectuate asupra soluției. Aceşti algoritmi trebuie abandonați atunci când dimensiunile sistemului depăşesc 10 ecuații.

Algoritmii care aproximează soluția sistemului liniar de ecuații sunt numeroşi şi dintre aceştia, evident, se aleg aceia care oferă un rezultat cât mai satisfăcător după un număr cât mai redus de iterații. Studiile statistice pe care programatorii le efectuează au menirea de a selecta unul dintre algoritmi ca fiind cel mai avantajos. Este rezonabil să se utilizeze analiza algoritmului folosind programul cu care algoritmul este implementat, pentru ca algoritmul este ceva, iar programul este cu totul altceva, mai ales atunci când este vorba de structurare pe module, de utilizarea de structuri de date a căror referire generează cicluri maşină care influențează radical viteza de prelucrare per iterație.

Algoritmii bazați pe generări de matrice şi pe efectuarea exclusivă de operații de înmulțire sunt importanți când dimensiunile problemelor depăşesc ordinul sutelor de ecuații şi se lucrează cu matrice rare.

Realizarea unui produs software pentru rezolvarea unui sistem de ecuații liniare presupune existența următoarelor module:
- citire dimensiune;
- citire nume variabile;
- citire tip sistem (matrice rară, matrice integrală de date, matrice bandă, matrice diagonală, matrice simetrică, sub-blocuri în matrice);
- citire matrice de date;
- citire termen liberi;
- stocarea datelor inițiale pe suport;
- citire precizie sau număr de iterații;
- generare soluție inițială;
- calcule pentru iterație;
- verificarea criteriului de final;
- efectuarea trecerii la iterația următoare;
- afişare mesaje;
- afişare soluție finală;
- afişarea stadiului prelucrării
- stocarea rezultatelor finale pe suport;
- stocarea rezultatelor intermediare pe su-

port.
Structura produsului este astfel realizata încât răspunde unor cerințe de calitate stabilite încă în fazele de definire a obiectivului şi de analiza a problemei pentru care se construieşte produsul software.
Produsul trebuie proiectat încât să includă şi modulele de validare a datelor.

Mecanism de testare automată

Mecanismele necesare testării automate vizează de la problemă la problemă realizarea unor componente care să efectueze şi prelucrările inverse proceselor pentru care s-a elaborat software. În aceste situații speciale, mecanismul de testare automată este cel prezentat în figura 2.

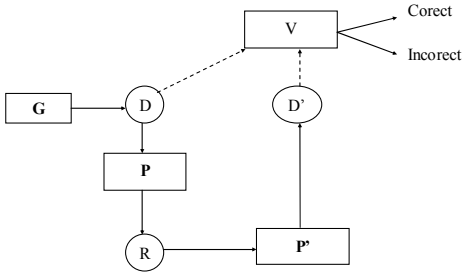

Fig. 2. Mecanism de testare automate pentru programe speciale

Din figura 2 se identifică următoarele elemente care apar în testarea automată a produselor software specializate:
- G – generatorul de date de intrare;
- D – datele de intrare generate;
- P – programul supus testării;
- R – rezultate obținute prin programul P cu datele de intrare D;
- P' – programul care implementează un algoritm invers celui implementat în programul P sau un algoritm pentru verificare;
- D' –rezultate obținute de programul P';
- V – program care compară rezultatele generate cu rezultatele obținute.

În funcție de specificul aplicației, fluxurile prezentate în figura 2 variază, însă principiul acestui mecanism de testare automată rămâne acelaşi.
De exemplu, pentru produsul software care rezolva sistemul de ecuații liniare: A * X = B
unde A este matricea coeficienților sistemului având n linii şi n coloane, B este vectorul termenilor liberi, având n componente, X este vectorul celor n necunoscute, testarea automată presupune parcurgerea următorilor paşi:
- se fixează pragul de eroare ε;
- se stabileşte numărul de teste, N;
- se generează soluțiile $X_1, X_2,...., X_N$;
- se efectuează înmulțirile $B_1=A*X_1$, $B_2=A*X_2,..$, $B_N=A*X_N$;
- se apelează programul care face obiectul testării pentru a rezolva sistemele $A*X=B_1$, $A*X=B_2,....$, $A*X=B_N$;
- se obțin soluțiile $X_1', X_2', ..., X_N'$ care aproximează soluțiile exacte;
- se calculează sumele $S_1, S_2,...., S_N$ ale pătratelor de diferențe dintre soluțiile exacte şi soluțiile aproximate;
- se afişează soluțiile pentru care sumele S_i sunt mai mari decât ε.

Testarea automată are menirea de a evidenția ponderea cazurilor în care programul nu satisface cerințele formulate în legătura cu problema de rezolvat.
Un alt subiect este acela al calculului salariilor nete pornind de la salariile brute. În acest scop se scrie un produs program P. Pentru testare automată se va scrie un program P' care calculează salariul brut pornind de la salariu net.
Testarea automată presupune parcurgerea următorilor paşi:
- se stabileşte lungimea fişierului de date ca număr de articole;
- se generează salariile nete ale persoanelor obținând fişierul F;
- se calculează cu programul P' salariile brute şi celelalte elemente care se scad din salariul brut până se obține salariul net;
- se obține un fişier F';
- cu fişierul F' se trece la calculul salariului net folosind programul P;
- se obține fişierul F";
- se compară salariul net din fişierul F cu salariul net din fişierul F";
- se afişează situațiile în care apar diferențe între cele doua salarii nete;
- se studiază ponderea situațiilor de rezultate eronat.

Al treilea exemplu de mecanism de testare automată vizează calculul inversei unei matrice. Se considera programul P care inver-

sează matrice. Pentru automatizarea testării se parcurg următorii paşi:

- se stabileşte pragul de eroare ε;
- se stabileşte numărul de testări;
- se stabileşte pasul r de variaţie a dimensiunii matricelor care se inversează;
- se stabileşte elementul de start pentru generatoarele de numere pseudo-aleatoare pentru a asigura reproductibilitatea procesului;
- se scrie programul G pentru generarea de matrice $A_{r,r}$ $A_{2r,2r}$,…, $A_{nr,nr}$;
- matricele sunt memorate în fişierele F_1, F_2,....,F_n;
- se apelează programul care inversează cele n matrice citite din cele n fişiere F_1, F_2,..., F_n;
- matricele inverse se stochează, respectiv în fişierele, $FINV_1$, $FINV_2$,…, $FINV_n$;
- se scrie un program P' care citeşte perechile de matrice din fişierele F_i şi $FINV_i$ şi calculează produsele de matrice;
- se calculează o sumă a pătratelor elementelor matricei rezultat care se compară cu suma pătratelor elementelor matricei unitate de acelaşi ordin pe care îl are matricea produs;
- se afişează situaţiile în care există diferenţe mai mari decât un prag ε definit;
- se face o analiză a calităţii procesului de inversare.

În procesul de testare automată a acestui tip de aplicaţii pot să apară o serie de erori legate de implementarea algoritmului în programul de test P' şi verificarea acestuia necesită o procedură separată.

Concluzii

Testarea automată este un proces laborios care necesită o studiere sistematică a problemelor de rezolvat prin programele care se scriu. Este necesară o abordare sistemică a întregului ansamblu şi un mod deschis de tratare a fiecărei probleme în parte.

Este foarte dificil să se considere că este posibilă elaborarea unei metodologii unice destinată testării automate şi crearea unui software, de asemenea general, dedicat testării automate. Problema trebuie pusă altfel. Trebuie mai întâi identificate probleme şi modalităţi punctuale de testare automată aşa cum s-a procedat în acest articol. Trebuie grupate tipurile de soluţii obţinute pentru testarea automată în vederea obţinerii unor instrumente limitate la clase de probleme.

Dacă se atinge un nivel foarte ridicat de abstractizare şi dacă experienţa permite efectuarea unor generalizări care să se dovedească viabile în plan practic, cu luarea în considerare a problemelor reale, deosebit de complexe, se conchide că s-a implementat o tehnică de testare automată. Paşii de parcurs sunt numeroşi, trebuind să existe un start spre probleme reale, asemenea celor date ca exemple în prezentul articol.

Bibliografie

[HUBE99] Huber, Jon T. – *Efficiency and Effectiveness Measures to Help Guide the Business of Software Testing,* Hewlett Packard Company, 1999

[IVAN99] Ivan, Ion, Pocatilu, Paul – *Testarea Software Orientat Obiect*, Editura Inforec, Bucureşti, 1999

[IVAN02] Ivan, Ion, Pocatilu, Paul, Ivan, Anca Andreea – *Control Structure Oriented Software Testing*, in Economy Informatics, vol. II, nr. 1, 2002, pp. 73-80

[KANE96] Kaner, Cem – *Quality Cost Analysis: Benefit and Risks*, Software QA, Volume 3, No. 1, 1996, p. 23

[McGR97] McGregor, John D. – *An overview of testing*, Journal of Object-Oriented Programming, January 1997.

[MYER79] Myers, Glenford J. – *The Art of Software Testing*, John Wiley & Sons, New York, 1979

[PATT01] Patton, Ron. – *Software testing*, SAMS Publishing House, USA, 2001

[PETE00] Peters, James F., Pedrycz, Witold – *Software Engineering – An Engineering Approach*, John Wiley & Sons, Inc, 2000

[POCA02] Pocatilu, Paul – *Costurile testării software*, în Informatica Economică vol. VI, nr. 1, 2002, pag. 90-93

[POCA02a] Pocatilu, Paul – *Automated Software Testing Process*, în Economy Infor-matics, vol. II, nr. 1, 2002, pp. 97-99

[POCA03] Pocatilu, Paul – "Automatizarea testării software orientat obiect", în Revista Tinerilor Economisti, An I, nr. 1, Noiembrie 2003, pp. 121-125

[PRES00] Pressman, Roger S. – *Software Engineering – A Practitioner's Approach, European Adaptation Fifth Edition,* McGraw-Hill, 2000

[SIEG96] Siegel, Shel, *Object Oriented Software Testing: A Hierarchical Approach*, Wiley Computer Publishing, 1996

[SILV02] SilverMark – *Smalltalk Testing Tips*, SilverMark Inc., presentations@silvermark.com, 2002

[TEOD99] Teodorescu, Laurenţiu, Ivan, Ion – *Managementul calităţii software*, Editura Inforec, Bucureşti, 1999

[TEST90] ***** – *Chapter 6: Testing*, London,1990, pag 105-122