AUTOTESTAREA APLICAȚIILOR INFORMATICE DISTRIBUITE CU CONȚINUT OFERIT

Ion Ivan
ionivan@ase.ro

Cristian Ciurea
cristian.ciurea@ie.ase.ro

Daniel Milodin
daniel.milodin@ase.ro

Rezumat: Se descriu tipurile de aplicații informatice distribuite. Sunt identificate clase de erori care afectează calitatea aplicațiilor informatice distribuite. Sunt date modalități de eliminare a erorilor din aceste aplicații. Pentru o serie de aplicații, sunt prezentate rezultate experimentale.

Autotestarea este specifică procesului de dezvoltare. Ciclul de dezvoltare este format din etape, iar testarea se realizează atât în cadrul fiecărei etape, cât și în final, pentru a lua fie decizia de implementare, fie decizia de continuare a ciclului de dezvoltare prin efectuarea de corecții.

Autotestarea precede trecerea la o etapă următoare și condiționează această trecere. Se execută de către cei care desfășoară activități în etapa curentă.

Este diferită de testare atât prin obiective și sarcini, cât și prin persoanele care o efectuează. Dacă testarea este efectuată de personal specializat, autotestarea este realizată de către toți cei care au derulat activități în cadrul fiecărei etape. Autotestarea revine la a compara ceea ce este făcut cu ceea ce trebuia făcut, așa cum rezultă din specificații.

Cuvinte cheie: autotestare, aplicații on-line, erori, calitate.

Abstract. There are described the types of distributed applications. There are identified classes of errors that affect the quality of distributed applications. Are given modalities to eliminate errors in these applications. For a number of applications are presented experimental results.

The auto-testing is specific for the development process. The development cycle consists of stages, and the testing is performed both within each phase, and in the end, in order to take either the decision to be implemented or the decision to continue the development cycle by making corrections.

Auto-testing proceeds moving to a next step and determining this transition. It is executed by those engaged in the current stage.

Auto-testing is different from testing, both by objective and tasks, and by persons who carry it out. If testing is performed by specialized personal, auto-testing is achieved by all those who have conducted activities under each phase. Auto-testing returns to compare what was done with what has to be done, as shown in the specifications.

Key words: auto-testing, on-line applications, errors, quality.

1. Tipuri de aplicații informatice distribuite

Există numeroase criterii de clasificare a aplicațiilor informatice distribuite. După criteriul complexității, se identifică:

- *aplicații simple*, care realizează o funcție, de regulă, de informare, prin selecții;
- *aplicații medii*, în care se efectuează prelucrări cu datele utilizatorului și afișare de rezultate, de informare;
- *aplicații complexe*, în care, pe baza datelor furnizate de utilizator, se efectuează alocări de resurse, rezervări, plăți, analize și se asistă procese de decizie.

După criteriul conținutului digital, aplicațiile informatice se clasifică în:

- *aplicații cu conținut oferit*, care sunt construite de echipe sau de persoane, pentru a oferi informațiile pe care acestea le structurează și le selectează după criterii proprii;
- *aplicații cu conținut așteptat*, în care este prezentată informația care pornește de la grupul țintă, de la nevoile de cunoaștere, de orientare, de formare ale acestuia;
- *aplicații cu conținut realizat după un șablon*, așa cum sunt prezentările de universități, de orașe, de enciclopedii;
- *aplicații cu conținut orientat spre continuitate*, care pornesc de la ceea ce există, ajutându-l pe utilizator, pe baza experienței acumulate, să-și soluționeze propriile probleme.

În toate cazurile, se pornește de la ideea de a oferi un conținut încât persoanele ce accesează astfel de aplicații să soluționeze probleme, să atingă obiectivul pentru care au efectuat accesarea [1].

Întrucât aplicațiile de acest tip sunt utilizate pentru luarea de decizii cu efecte imediate asupra resurselor decidentului, conținutul digital furnizat trebuie să fie:

- *complet*, ceea ce revine la a furniza toate datele referitoare la problematica abordată; în cazul în care unele dintre date lipsesc, trebuie menționat explicit acest lucru;
- *corect*, ceea ce corespunde realizării concordanței între ceea ce se prezintă și realitatea pe care o reflectă; corectitudinea se asigură prin redarea fidelă a realității, prin preluarea fără omisiuni sau interschimburi a datelor, indicând sursa;

- *coerent*, aspect asigurat prin dezvoltarea unui plan de structură, astfel încât să se asigure o dezvoltare logică de la simplu la complex, de la dreapta spre stânga, de sus în jos;
- *consistent*, prin evitarea elementelor care au rol de a anula elemente definite anterior, de a contrazice ceea ce a fost afirmat într-una dintre componente;
- *neambiguu*, caracteristică obținută prin nivelul de înțelegere de către dezvoltătorul de conținut digital; se va evita utilizarea unor expresii *posibil ca*, *ar fi dacă*, *și nu în ultimul rând*;
- *clar*, ceea ce presupune o dezvoltare a unei structuri simple, articulate, acceptate și pe înțeles, cu abordare graduală folosind elemente sugestive;
- *omogen*, din punct de vedere al structurii și din punct de vedere al nivelului de abordare; se construiește un șablon și se urmărește ca toți dezvoltatorii de conținut digital să urmărească realizarea, folosind jaloanele definite.

Conținutul digital se structurează astfel încât traversarea lui să se bazeze pe reguli acceptate, în număr cât mai restrâns, dacă se forțează lucrurile, folosind numai o regulă.

Este de preferat să se dezvolte o structură arborescentă, cu traversare numai din aproape în aproape, (figura 1):

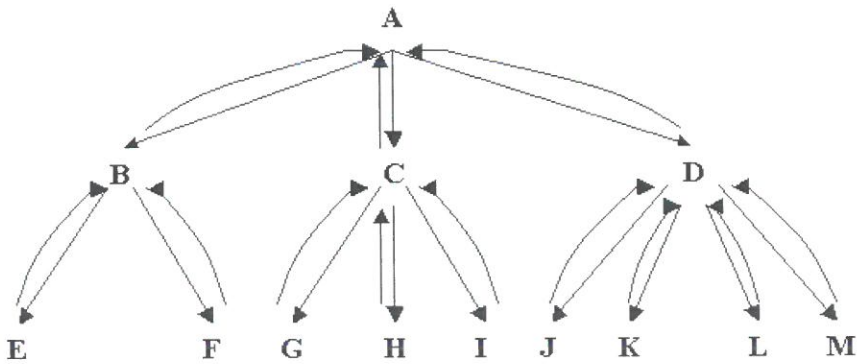


Figura 1. Structură arborescentă de conținut cu referire din aproape în aproape

Pornind de la obiectivul urmărit, se stabilește conținutul digital ca lungime, structură, complexitate, nivel de abordare și modalitate de accesare.

Obiectivul se definește și în funcție de resursele disponibile, momentul în care aplicația trebuie să devină operațională și, mai ales, de experiența personalului.

Între o aplicație sofisticată, nefuncțională la momentul stabilit, sau care va avea probleme tot timpul, și o aplicație simplă, dar operaționala, fără probleme, evident trebuie aleasă cea de-a doua variantă, întrucât gradul de satisfacție al utilizatorilor este pe primul plan.

2. Tipuri de erori în aplicațiile cu conținut digital oferit

De regulă, conținutul oferit este dat sub forma unei structuri arborescente, ca în figura 2:

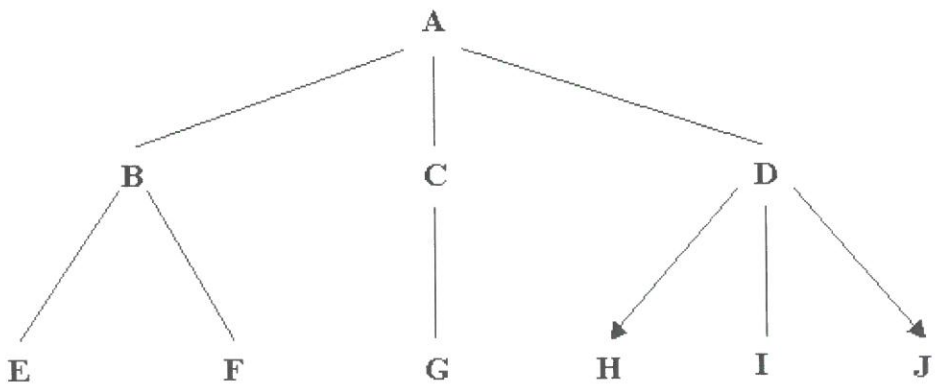


Figura 2. Structură arborescentă a conținutului oferit

Rezultă că B, C și D sunt părți din care este alcătuit A.

La construirea unui conținut digital, una dintre erorile frecvente este aceea de a nu include submulțimi omogene pentru a forma o mulțime.

În cazul în care structurii arborescente i se asociază slide-uri distincte pentru niveluri, conținutul digital își pierde coerența, atunci când pe nivelul i al structurii arborescente se anunță elementul X_j, iar la accesarea acestuia este conținutul digital corespunzător elementului X_k. Eroarea de neconcordanță între conținutul anunțat și cel referit are la bază numeroase cauze, dintre care cea mai frecventă este legată de codificările construite pentru fișiere, mai ales atunci când se lucrează cu un număr foarte mare de fișiere.

Inexistența conținutului anunțat este o eroare care pune într-o lumină absolut nefavorabilă o aplicație informatică distribuită [2].

Cauza este simplă: fișierele anunțate nu există pentru că sunt referite sub un alt nume decât cel sub care este definită acea parte de conținut digital; lipsa unui caracter, o extensie scrisă cu majuscule, când în realitate trebuie scrisă cu litere mici, neîncărcarea fișierului în aplicație reprezintă alte cauze.

Cele mai grave erori vizează interschimbul de legături.

Dacă structura inițială este cea din figura 3, prin interschimb se obține o nouă structură, figura 4.

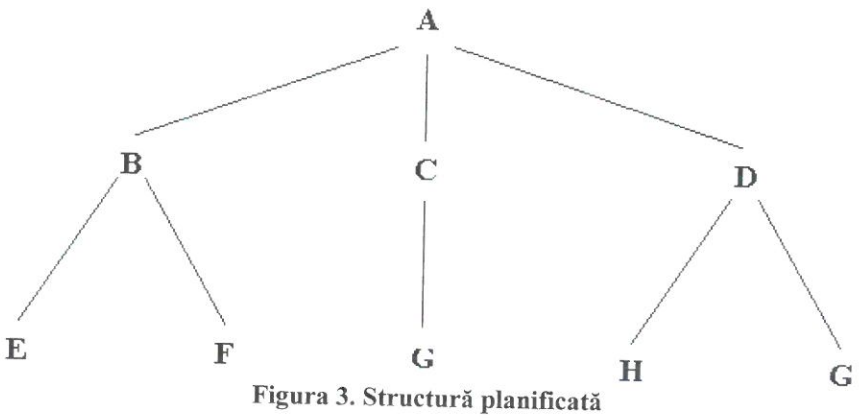


Figura 3. Structură planificată

În figura 3, se prezintă un model de structură planificată a unei aplicații informatice distribuite, iar în figura 4 - un exemplu de structură implementată:

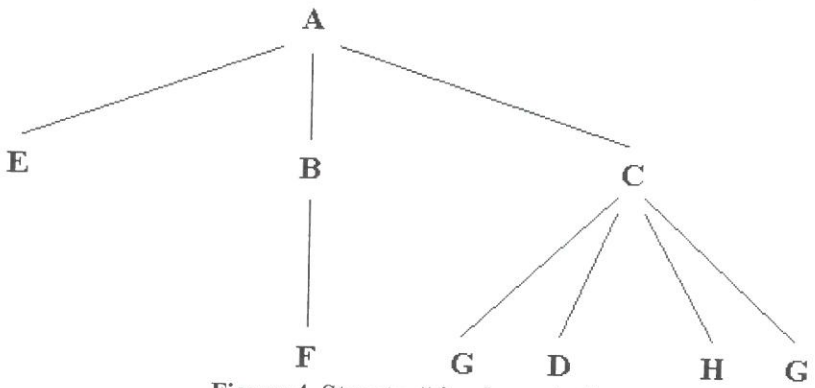


Figura 4. Structură implementată

Apar situații în care, deși există conținut, acesta nu este referit. În figura 5, se prezintă structura planificată a unei aplicații informatice distribuite, în care conținutul este referit:

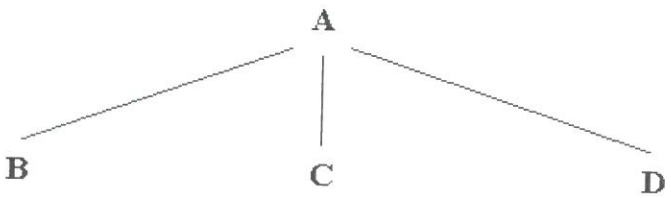


Figura 5. Structură planificată

În figura 6, se prezintă structura implementată a unei aplicații informatice distribuite, în care

conținutul nu este referit în totalitate:

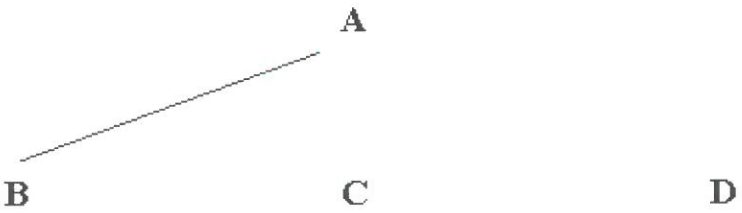


Figura 6. Structură implementată

Aceste situații determină o destructurare a conținutului digital.

Interschimbul este voluntar şi presupune modificări în specificații pentru a realiza concordanța acestora cu structura reală a aplicației.

În cazul în care interschimbul este involuntar, rezultat al unei erori, prin autotestare se evidențiază modulul interschimbat, efectele interschimbului şi se procedează la efectuarea corecției, astfel încât la etapa următoare aplicația informatică distribuită apare sub forma unui stadiu intermediar ce respectă cerințele din specificații.

Un alt tip de erori vizează conținutul însuşi, erorile fiind generate de:

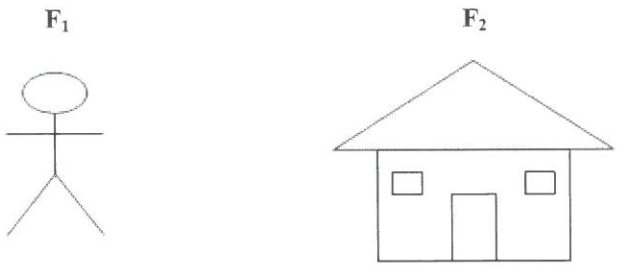
- inexactități de terminologie;
- localizări eronate;
- indicarea incorectă a numelor de persoane;
- utilizarea incorectă a momentelor de timp;
- punerea incorectă în corespondență a textelor cu imaginile;
- prezentarea de texte si imagini trunchiate;
- utilizarea de reguli ortografice, altele decât cele aflate în vigoare.

Erorile de asociere de fişiere

Există fişierul F_1 conținând imaginea cu portretul aparținând persoanei X şi alături imaginea unei picturi făcută de el, în fişierul F_2. Aceasta este asocierea corectă.

Dacă se pune F_1 şi F_3, reprezentând un tablou făcut de altul, este deja o eroare.

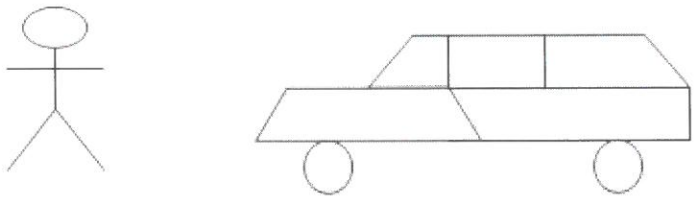
Folosind pentru exemplificare imagini reprezentând doi pictori şi tablourile realizate de aceştia, rezultă următoarele situații de asociere a fişierelor:



Corect

Text 1
F_1

Text 2
F_3



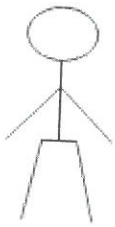
Incorect, pentru că nu este tabloul cu casa

Text 1
F_4

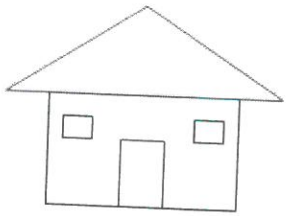
Text 2
F_2

Incorect, pentru că nu este pictorul din F_1, ci

pictorul din F_4.



Text 1



Text 2

Apare incorectitudine şi în situaţia în care se gestionează texte, nu numai în cazul în care se lucrează cu imagini. În tabelul 1, se prezintă combinaţii de erori care apar atât la asocierile de imagini, cât şi la asocierea de texte la imagini.

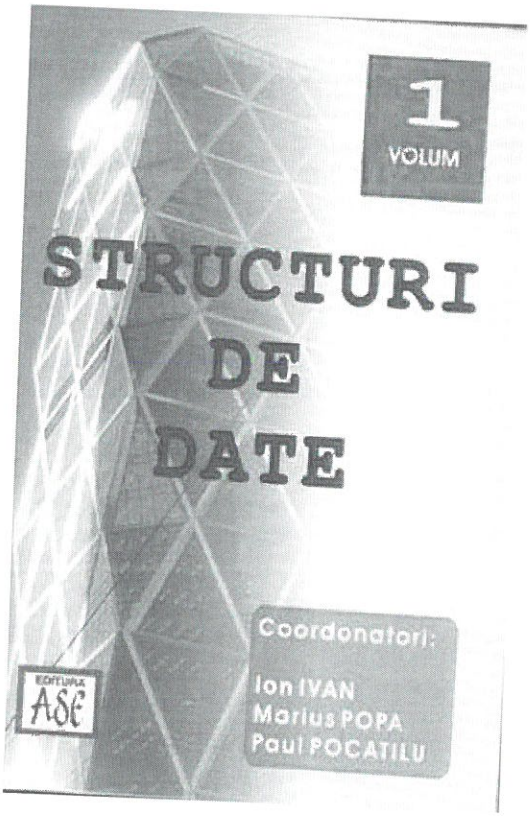
Tabelul 1. Combinaţii de erori legate de înlocuire poze şi texte

	Portret pictor	Tablou	Text sub pictor	Text sub tablou	
Erori înlocuire imagini	F1	F2	T1	T2	Corect
	F1	F3	T1	T2	Incorect, apare F3 în loc de F2
	F4	F2	T1	T2	Incorect, apare F4 în loc de F1
	F4	F3	T1	T2	Incorect, apare F4 în loc de F1 şi F3 în loc de F2
Erori înlocuire texte	F1	F2	T1	T3	Eronat, explicaţia T2 este înlocuită cu T3
	F1	F2	T4	T2	Eronat, explicaţia T1 este înlocuită cu T4
	F1	F2	T4	T3	Toate textele sunt eronate

Soluţia este dată de codificare şi de existenţa în specificaţii explicit imagini şi texte corecte.

Se consideră două cărţi de informatică, ale căror coperţi scanate sunt prezentate în continuare. Se asociază câte un comentariu la fiecare carte. În situaţia în care comentariul asociat unei cărţi corespunde cărţii respective, atunci asocierea este corectă [3]. Dacă autorii de la o carte sunt trecuţi în comentariul aparţinând altei cărţi, atunci asocierea este greşită.

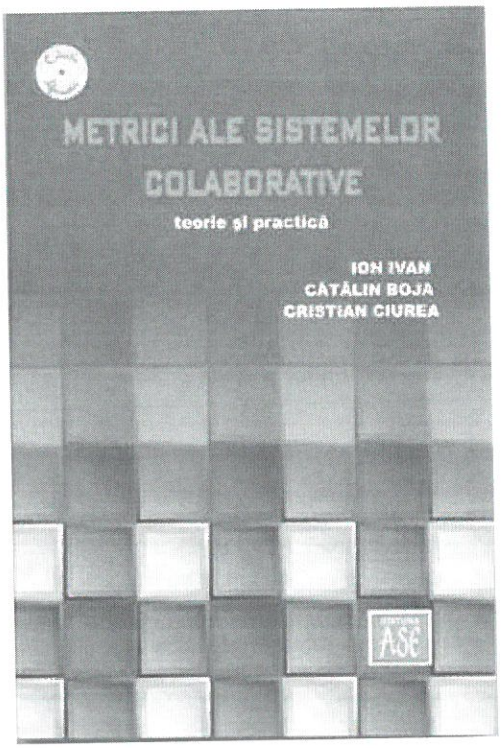
Coperta 1

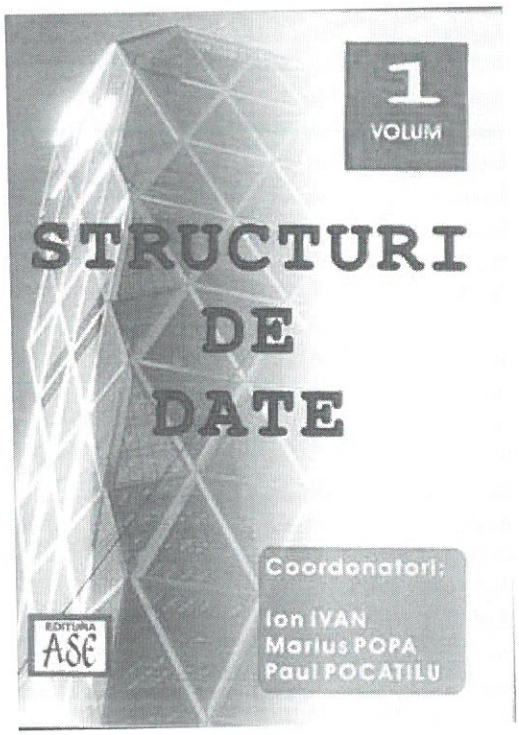


Comentariul 1

Ion IVAN, Marius POPA, Paul POCATILU - *Structuri de date*, vol. 1, Editura ASE, Bucureşti, 2008.

Corect

Coperta 2	Comentariul 2	
	**Ion IVAN, Cătălin BOJA, Cristian CIUREA – *Metrici ale sistemelor colaborative, teorie si practică*, Editura ASE, Bucureşti, 2007.**	**Corect**

Coperta 1	Comentariul 2	
	**Ion IVAN, Cătălin BOJA, Cristian CIUREA – *Metrici ale sistemelor colaborative, teorie şi practică*, Editura ASE, Bucureşti, 2007.**	**Incorect, pentru că nu este comentariul asociat cărţii de Structuri de date, ci cărţii de Metrici ale sistemelor colaborative**

Unele dintre erori trec neobservate, însă cele mai multe compromit integral aplicația. Numeroşi utilizatori au comună caracteristica în ceea ce priveşte referirea de aplicații informatice, referire care presupune ca, la prima eroare detectată, să întrerupă accesul, şi să nu repete încercarea la un alt interval de timp. Existența forumurilor de comentarii nu face altceva decât să propage erorile întâlnite, compromițând iremediabil aplicația. Singura soluție este abandonarea aplicației. Există însă o soluție care îmbunătățește această abordare: lansarea numai după efectuarea autotestării [4].

3. Autotestarea completă a legăturilor

O aplicație informatică oarecare se testează în diverse proporții, întrucât erorile nu au caracter ireversibil. În cazul aplicațiilor informatice care operează online, orice eroare are caracter destructiv, efectele resimțindu-se imediat, iar ulterior, prin propagare se amplifică. De aceea, aplicațiile informatice distribuite trebuie complet testate.

În cazul aplicațiilor cu conținut oferit, autotestarea este esențială, procesul de testare trecând pe plan secundar, având rolul de a constata că lucrurile sunt în regulă.

Autotestarea se efectuează de către dezvoltatorul aplicației cu conținut oferit.

Se definește autotestarea ca proces de evidențiere de către executant a măsurii în care un stadiu al aplicației informatice distribuite este în concordanță cu cerințele specificațiilor.

Necesitatea autotestării este dată de:

- reducerea numărului de erori de la un stadiu la altul;
- detectarea erorilor în faza de realizare a aplicației, întrucât persoanele implicate în realizarea unui stadiu al aplicației găsesc erorile mult mai repede și le corectează fără costuri prea mari;
- erorile dintr-un stadiu nu sunt amplificate de dezvoltarea stadiilor următoare.

Autotestarea pe niveluri presupune atingerea tuturor nodurilor de pe un nivel de către seturile de date de test.

Pentru structura din figura 7, autotestarea presupune referirea nodurilor B, C, D mai întâi, iar după aceea a nodurilor E, F,G,H, I, J, K, L, M, N.

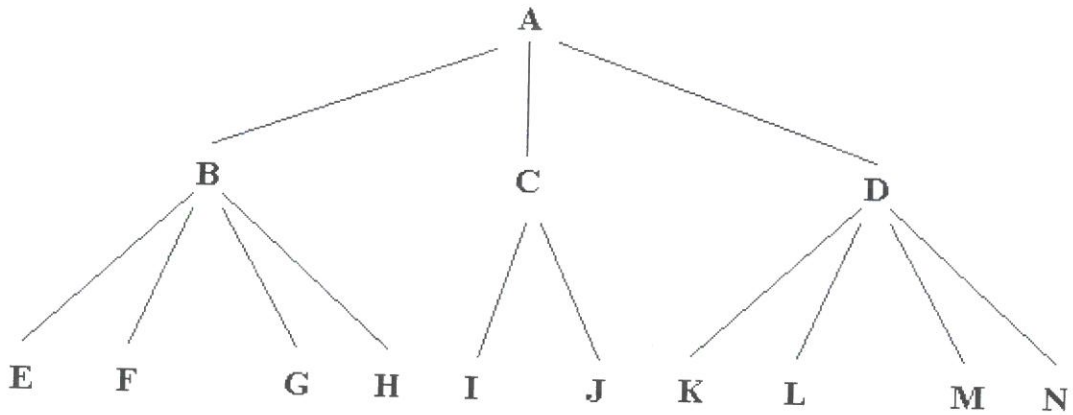


Figura 7. Structură planificată

Autotestarea în adâncime presupune traversarea tuturor nodurilor de la rădăcină până la frunză, pentru fiecare frunză în parte. Pentru structura din figura 7, referirile din procesul de autotestare sunt ABE, ABF, ABG, ABH, ACI, ACJ, ADK, ADL, ADM, ADN.

Dacă structura prevede referiri ca în figura 8, în cazul testării pe niveluri se realizează revenirile ABA, ACA, ADA, BEB, BFB, BGB, BHB, CIC, CJC, DKD, DLD, DMD, DND.

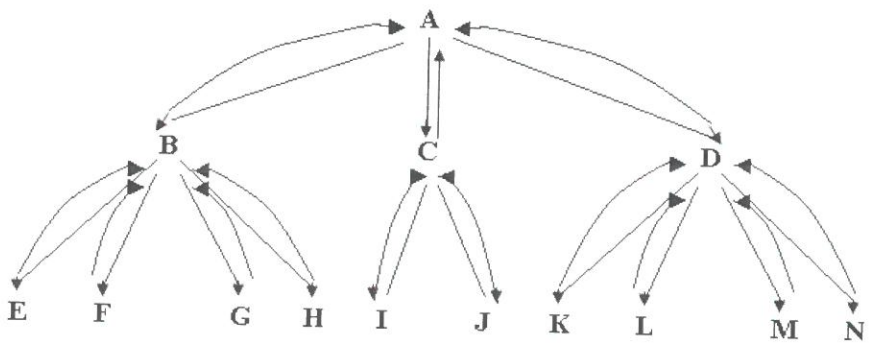


Figura 8. Structură planificată cu conținut digital oferit cu revenire din aproape în aproape

În cazul autotestării în adâncime, referirile sunt ABEBA, ABFBA, ABGBA, ABHBA, ACICA, ACJCA, ADKDA, ADLDA, ADMDA, ADNDA.

Autotestarea presupune traversarea integrală a structurii, fiind un proces de control complet pentru totalitatea nodurilor şi pentru totalitatea legăturilor.

Rezultatul autotestării este un raport care conţine:

- mulţimea seturilor de date de autotestare a legăturilor, prima coloană a tabelului 1, în care sunt prezentate cerinţele pentru aplicaţia informatică având conţinut oferit, având structura dată în figura 8;
- mulţimea rezultatelor efective ce vizează comportamentul real al aplicaţiei; în cazul unei construcţii corecte, între elementele din prima coloană şi aceasta nu există diferenţe; cu cât s-au produs erori la implementare, cu atât diferenţele sunt mai mari; erorile sistematice vizează o abordare superficială a problemei, dar şi o soluţionare nedescoperită a acesteia.

Tabelul 2. Raportul de autotestare

Seturi de date de test	Comportament efectiv	Comentarii
ABA	ABA	Ok
ACA	AC	Lipsa revenirii in A
ADA	ADA	Ok
BEB	BEB	Ok
BFB	BFB	Ok
BGB	BG	Lipsa revenirii in B
BHB	BHA	Eroare revenire in A in loc de B
CIC	CIC	Ok
CJC	CJB	Eroare revenire in B in loc de C
DKD	DKA	Eroare revenire in A in loc de D
DLD	DLB	Eroare revenire in B in loc de D
DMD	DM	Lipsa revenirii in D
DND	DND	Ok

Comentariile sunt efectuate tot de dezvoltatori şi sunt sursă sigură de eliminare a erorilor.

Există o colectivitate C formată din elementele $C_1, C_2, \ldots, C_N$. Se realizează imaginile $I_1, I_2, \ldots, I_N$.

Elementele colectivităţii C se grupează după criteriile $G_1, G_2, \ldots, G_K$, formând submulţimile $S_1, S_2, \ldots, S_K$.

La rândul lor, submulţimile $S_1, S_2, \ldots, S_K$ sunt alcătuite din subsubmulţimile nule $SS_{11}, SS_{12}, \ldots, SS_{1K1}, SS_{21}, SS_{22}, \ldots, SS_{2K2}, \ldots, SS_{K1}, SS_{K2}, \ldots, SS_{KLK}$.

Fiecare submulţime este prezentată prin imagini.

Între submulţimile S_i si S_j există relaţia $S_i \cap S_j = \Phi$.

Între submulţimile SS_{ij} si SS_{ih} există relaţia $SS_{ij} \cap SS_{ih} = \Phi$.

O subsubmulţime SS_{ij} este obţinută prin aplicarea criteriului $GG_{i1}, GG_{i2}, \ldots, GG_{iLhi}$.

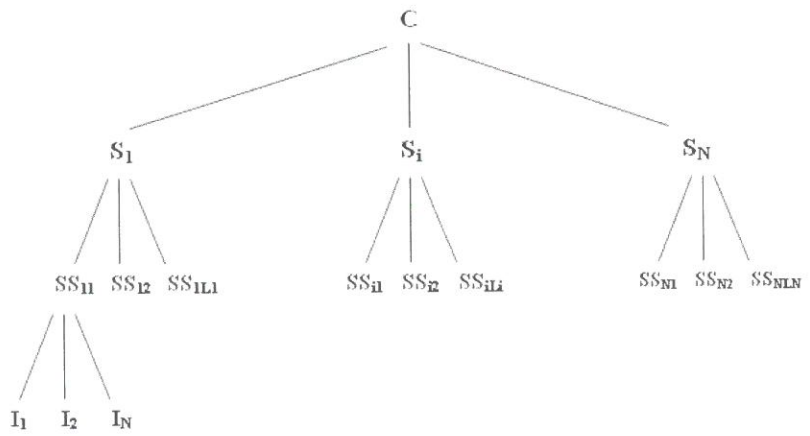


Figura 9. Legăturile dintre colectivitate, submulţimi, subsubmulţimi şi imagini

Aplicațiile construite pe bază de matriță sau șablon permit crearea de conținut digital standard și personalizarea acestui conținut cu elemente specifice fiecărui nivel al structurii arborescente [5].

O imagine apare la o singură submulțime:

- pozele dintr-o excursie;
- o carte de istorie, de fizică;
- o colecție de roci;
- o colecție de degetare;
- prezentarea unei case memoriale, a unui pictor, a unei colecții de obiecte dintr-o colecție privată;
- un CV.

Este prezentarea a ceea ce dorește cel care dispune de obiecte, de mărturii, să prezinte cât vrea el, cum vrea el, când vrea el.

În cazul în care $SS_i \cap SS_j \neq \Phi$, înseamnă că sunt elemente care, clasificate după mai multe criterii, aparțin mai multor colectivități.

Exemplu: degetarul cu un animal – elefant (SS_7), degetarul de metal (SS_8), degetarul dintr-o țară – India (SS_{12}), degetarul de lut (material, Bulgaria, arhitectură), arcul de triumf (țară – Franța, material, tehnică de realizare, monument, similitudine culoare, oraș – Paris).

În aceste cazuri:

- redundanța reduce riscul de referiri eronate;
- poza unică înseamnă o gestiune foarte riguroasă.

În figura 10, se prezintă structura unei aplicații pentru afișarea imaginilor unei colecții de degetare decorative, aplicație disponibilă la adresa http://www.degetare.ro:

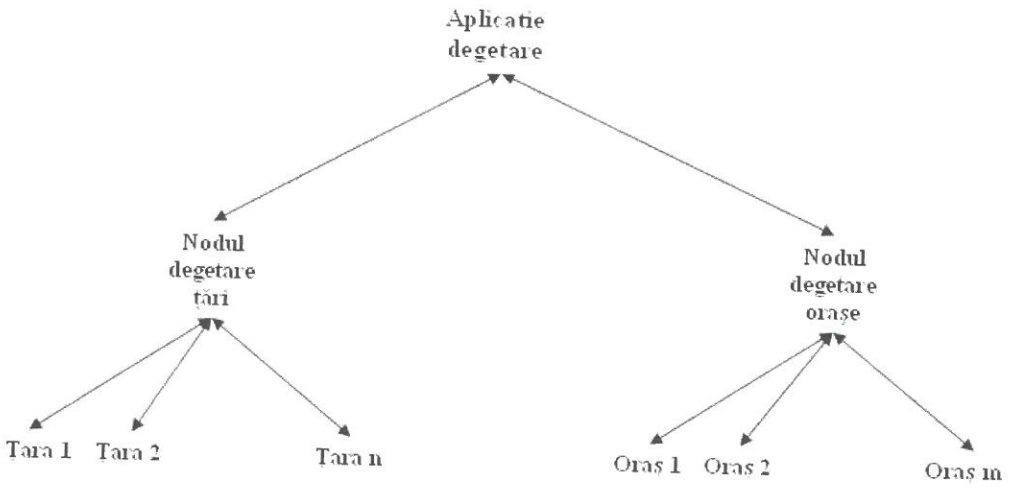


Figura 10. Aplicație pentru afișarea imaginilor unei colecții de degetare decorative

O aplicație pentru afișarea imaginilor unei colecții de degetare decorative are o structură arborescentă, arborele stâng fiind reprezentat de nodul degetare țări, iar arborele drept de nodul degetare orașe. Nodul degetare țări va conține imaginile unor degetare din n țări, iar nodul degetare orașe va conține imaginile degetarelor din m orașe ale lumii. Fiecare imagine este inserată într-o pagină a aplicației. Fiecare pagină conține o ancoră de revenire la nodul de pe nivelul superior al arborelui. Realizarea aplicației presupune crearea a n fișiere pentru țări și m fișiere pentru orașe. Cele n fișiere pentru țări diferă între ele prin imaginea degetarului din țara respectivă, ancorele de revenire fiind identice. Același lucru este valabil și pentru cele m fișiere pentru orașe. Pentru creșterea productivității în activitatea de programare, programatorul trebuie să țină cont de aceste aspecte.

Dacă se creează $k < n$ fișiere, în loc de n fișiere pentru țări, înseamnă că s-au pierdut o parte din imagini. Conștientizarea acestui aspect rezultă din specificațiile aplicației, unde este precizat numărul de țări la care se face referire.

4. Autotestarea completă a conținutului

În raport cu obiectivul propus, a fost construită o structură a aplicației, s-au elaborat specificații de realizare. Specificațiile de conținut au o modalitate aparte de realizare. Apar elemente de creativitate pe măsura dezvoltării aplicației. Esența conținutului definit în specificații nu trebuie să difere față de ceea ce conține aplicația reală.

Există conținutul anunțat dat sub forma unui titlu pe nivelul k al structurii și conținutul efectiv definit pe nivelul k+1.

Între cele două conținuturi trebuie să existe concordanță, în sensul că dezvoltatorul de conținut pentru nivelul k trebuie să se asigure că și dezvoltatorul de pe nivelul k+1 realizează texte și imagini referitoare la același lucru. Mai mult, la nivelul k+1 există responsabilități mult mai mari întrucât textul efectiv trebuie să fie complet, coerent, corect și consistent, în timp ce textul de pe nivelul k este numai un text care enumeră elemente sau anunță într-o formă sintetică un concept.

Responsabilitatea la nivelul k+1 este absolută pentru că atingerea obiectivului inițial depinde strict de conținutul acestui nivel. Realitatea arată că nivelurile 2, 3, 4 etc. sunt la rândul lor nivelul k+1, ceea ce demonstrează că responsabilitatea este transferată integral în structuri.

Există o modalitate aparte la nivelul frunzelor structurii arborescente de a asigura:

- detalierea descrierilor;
- compunerea, în cazul în care nu au fost incluse pe nivelurile precedente.

Când se efectuează autotestarea de conținut se urmărește fiecare drum de la rădăcină până la frunză, pentru a se vedea:

- *consistența*, fiind definită ca însușirea unei aplicații de a nu fi în contradictoriu, de a fi rezistentă la atacuri de securitate;
- *corectitudinea* este calitatea unei aplicații de a se comporta corect, de a respecta regulile de funcționare, de a fi lipsită de greșeli;
- *completitudinea* este proprietatea unei aplicații de a conține toate elementele necesare unei funcționări corecte și de a oferi toate informațiile solicitate și așteptate de utilizatori;
- *claritatea*, definită ca fiind proprietatea conținutului unei aplicații de a fi inteligibil, transparent și ușor de înțeles.

Lipsa de consistență într-o aplicație cu conținut digital oferit înseamnă sensibilitatea aplicației la atacuri de securitate și număr mare de accesări, lipsa de corectitudine presupune o funcționare aleatoare, lipsa de completitudine se referă la oferirea unor informații parțiale, iar lipsa de claritate înseamnă un conținut neinteligibil [6].

Se consideră un pictor care oferă primăriei orașului natal colecția de tablouri de care dispune pentru a face o expoziție permanentă. Lipsa de consistență înseamnă existența unor tablouri fragile și ușor de deteriorat, lipsa de corectitudine înseamnă asocierea etichetei cu descrierea unui tablou la un alt tablou, lipsa de completitudine se referă la o colecție incompletă de tablouri, iar lipsa de claritate se referă la descrieri ale picturilor conținând texte greu de înțeles.

În cazul în care apar neconcordanțe, se procedează la:

- trecerea de conținut de pe nivelul *k+1* pe nivelul *k* sau dacă pe nivelul *k* sunt detalii, acestea se trec pe nivelul *k+1*;
- se înlocuiesc imagini sau texte;
- se interschimbă imagini sau texte în cadrul aceluiași nivel;
- se elimină imagini sau texte;
- se inserează imagini sau texte.

Toate acestea au menirea de a crește calitatea conținutului. Se elimină redundanța sau inconsistența.

Conținutul unei aplicații trebuie testat detaliat, în funcție de tipul aplicației și utilizatorilor acesteia, deoarece acesta trebuie să ofere informații corecte și complete și, mai ales, actualizate.

O aplicație informatică trebuie să conțină termeni de utilizare a acesteia și de acceptare a conținutului de către utilizatori. În funcție de tipul aplicației, în termenii de utilizare se va specifica clar că informațiile prezentate în aplicație nu au valoare contractuală, au caracter orientativ, iar proprietarul aplicației nu este răspunzător de eventualele erori de conținut sau neconcordanțe cu realitatea.

5. Căi de creştere a calităţii conţinutului digital oferit

Realizarea de specificații de conținut structurate, astfel încât fiecărei componente definită distinct să-i corespundă în aplicația informatică o legătură, deci un fișier.

Acest lucru este realizabil dacă și numai dacă cel care definește sau cei care definesc conținutul digital au o imagine clară asupra întregului, dar și asupra detaliilor, descompunerea top-down fiindu-le la îndemână într-un mod facil.

În cazul în care specificațiile nu sunt structurate, dezvoltatorului îi revine sarcina descompunerii în părți a conținutului digital. Dacă sunt mai mulți dezvoltatori care lucrează după concepții de structurare definite, unii după complexitate, alții după semnificație, iar alții după omogenitate, cu siguranță că vor apare elemente multiplu definite, cum tot așa apar elemente nedefinite. În ambele cazuri, există riscul unor referiri multiple, dar și a absenței referirilor, respectiv figurile 11 și 12:

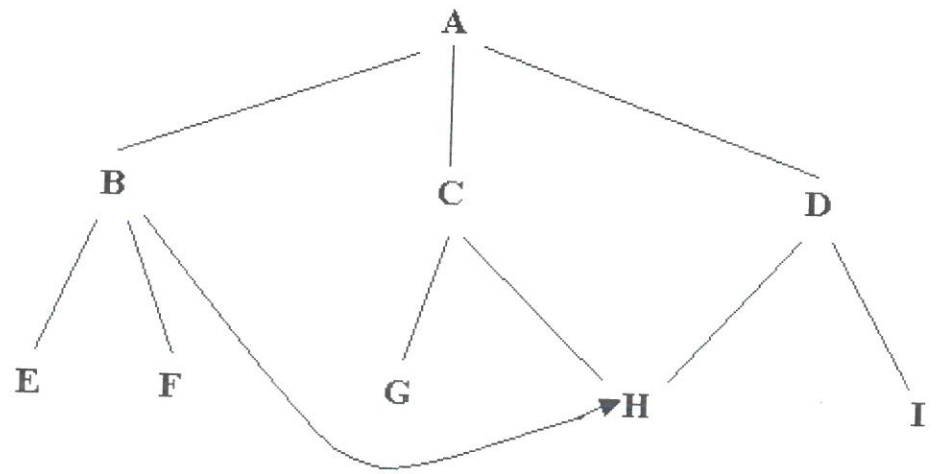


Figura 11. Referiri multiple ale nodului H

În figura 12, se prezintă o structură de aplicație având conținut nereferit de o parte din noduri:

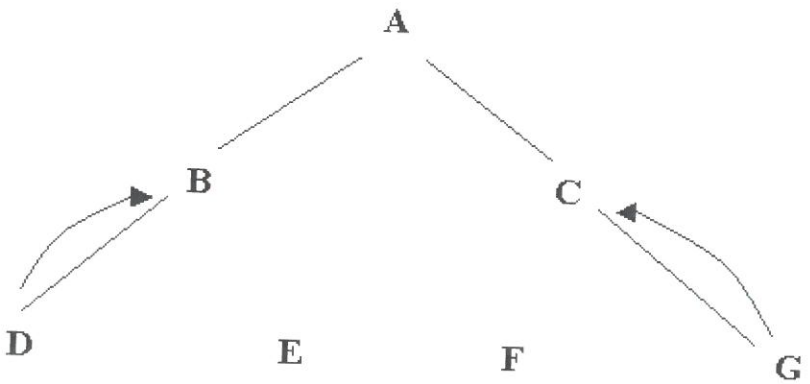


Figura 12. Conținut nereferit de nodurile E și F

Crearea de matrițe pentru tipurile de fișiere dezvoltate după aceleași reguli, asemenea șablonului, dezvoltatorii executând numai copierea lor în fișiere vide și adaptând textul lor la secvența de conținut digital care este referită. Matrița este autotestată astfel încât realizează cu siguranță tipul de funcție pentru care a fost creată. De asemenea, include secvența care asigură toate tipurile de reveniri, fie la nivelul k-1, dacă ea se află pe nivelul k, fie la nivelul k_0, dacă se acceptă revenirea la rădăcina structurii arborescente [7].

Matrița conține totalitatea structurilor tabelare, de fapt, de definire a imaginilor, de aliniere, obținându-se în acest fel omogenitatea tuturor textelor aflate pe același nivel.

Asigurarea gestiunii prin coduri și cifre de control presupune stabilirea numărului de noduri de pe fiecare nivel și, în acest fel, se va ști cu exactitate numărul de fișiere de pe fiecare nivel.

Și în cadrul fiecărui conținut digital, se vor gestiona fișierele cu imagini care se referă.

Pentru a realiza gestiunea fișierelor, se impune definirea unui sistem de codificare prin care:

- să se asigure diferențierea elementelor;
- să se obțină identificarea apartenenței la o anumită subcolectivitate omogenă;
- să se mențină crearea unei secvențe ordonate crescător;
- să se verifice dacă numărul de fișiere planificate a fi realizate și încărcate se află încărcate în aplicație și, mai mult, sunt referite, iar la rândul lor referă.

Autotestarea vizează măsura în care dezvoltatorul a avut capacitatea de a codifica adecvat fișierele ce definesc părți ale conținutului digital oferit.

Construirea de părți de conținut digital, integrarea lui în aplicație și realizarea autotestării pe loc, a modului în care noua componentă este referită. Pentru a lăsa loc dezvoltărilor ulterioare, se încheie conținutul cu un text care marchează faptul că acesta este în lucru sau în construcție sau are caracter provizoriu. Este preferabil să se precizeze când se încheie caracterul de provizorat al conținutul digital respectiv.

Se face verificarea textului conținut de fiecare fișier, care se încarcă în vederea referirii de pe nivelul k, dar și a referirii din el a altor fișiere de pe nivelul k+1. Se verifică dacă:

- numele fișierului referit din expresia de pe nivelul *k* este identic cu numele fișierului încărcat pentru nivelul *k+1*;
- textul de referire de pe nivelul *k* este în concordanță cu conținutul fișierului referit aflat pe nivelul *k+1*;
- structura conținutului este corectă din punct de vedere sintactic și generează exact ceea ce este dat în specificații.

Se consideră că realitatea sau contextul real conține componentele C_1, C_2, ..., C_n. Pornind de la aceste componente, conținutul digital creat va fi format din fișierele F_1, F_2, ..., F_m. Conținutul inserat în aplicație va conține elementele G_1, G_2, ..., G_k.

Dacă m > n, atunci există conținutul C_i căruia îi corespund cel puțin două fișiere F_k și F_{k+1}.

Dacă m < n, atunci există conținuturile C_i si C_j comasate în fișierul F_k.

Dacă m = n, atunci fiecărui conținut C_i îi corespunde un fișier F_i.

Ordinea în secvență

Dacă S este structura conținutului real și S' este structura în site, atunci trebuie ca S' să nu difere semnificativ de S.

Dacă m = k, înseamnă că fișierului F_i îi corespunde componenta G_i inserată (caz ideal, C_i -> F_i -> G_i).

Dacă m > k, înseamnă că a fost utilizat un fișier F_j.

Dacă m < k, înseamna ca au fost inserate fișiere din alte aplicații (este grav dacă în mai multe aplicații se mențin aceleași coduri).

Toate acestea presupun acceptarea unei activități de rutină în dezvoltarea conținutului oferit, astfel încât profesionalismul să vizeze atât latura calitativă, dar mai ales latura cantitativă. Latura calitativă vizează matrițele care, în diversitatea lor, acoperă tipologii de conținut digital de calitate foarte bună, din punct de vedere structural, și cel al referirilor [8]. Latura cantitativă vizează ca toate fișierele să fie de foarte bună calitate prin reutilizarea matrițelor.

6. Concluzii

Autotestarea este un concept nou, care este preluat din practica medicală tradițională, în conformitate cu care a preveni este cu mult mai eficient decât a trata o suferință. Întrucât autotestarea este realizată de însuși dezvoltatorul aplicației, acesta conștientizează că este cu mult mai profitabil pentru el să realizeze

de la început componentele de conținut digital corecte, complete, consistente și clare, întrucât agregarea lor va conduce, în final, la un conținut digital, de asemenea, corect, complet, consistent și clar. Riscul care apare, în cazul autotestării unei aplicații de însuși dezvoltatorul acesteia, este acela că unele erori trec neobservate. Se recomandă respectarea principiului celor patru ochi pentru a evita o astfel de situație.

În cazul testării, dezvoltatorul realizează componente, alții – testerii – le verifică și, de cele mai multe ori, depanarea o face o a treia categorie, astfel încât transferul de sarcini este delegat, cu consecințe dintre cele mai nefavorabile.

Autotestarea presupune ca o aceeași echipă să realizeze:

- preluarea specificațiilor;
- realizarea conținutului digital;
- verificarea legăturilor dintre niveluri;
- măsurarea nivelului caracteristicilor de calitate;
- identificarea erorilor;
- corectarea erorilor.

De fiecare dată, se va ști cu exactitate cine, când, cum, cât, unde a intervenit și cu ce rezultate.

Conținutul digital oferit, dar care rezultă din prelucrări, trebuie să respecte în mod obligatoriu cerințe legale de copyright. De asemenea, se prezintă corect și complet sursa și localizarea ca volum, pagină, autor etc.

În afară de autotestarea efectuată de echipa de dezvoltare, aplicațiile informatice trebuie să aibă capacitatea să se autotesteze. Proiectarea și implementarea unei aplicații trebuie să ia în considerare introducere unor variabile contor, definirea unor indicatori, cu ajutorul cărora să se verifice funcționalitatea aplicației. Rezultatele autotestării se livrează sub forma unor fișiere jurnal, pe care administratorul aplicației le vizualizează și ia măsurile necesare pentru corectarea erorilor apărute. Aplicația va funcționa asemenea unui sistem colaborativ, în care toate componentele cooperează pentru atingerea unui obiectiv comun reprezentat de o funcționare corectă și permanentă.

În aplicațiile distribuite, există tendința spre un nivel de creativitate exagerat. Se impune crearea unor șabloane, definirea unor reguli, de largă acceptare astfel încât, pentru clase de conținut digital oferit, să existe elemente de apropiere din punct de vedere structural. Pentru mulțimi omogene de elemente, în cazul în care se identifică un șablon și un set de reguli acceptat, se soluționează fără riscuri majore problemele de a:

- traversa conținutul digital;
- regăsi conținut digital;
- căuta conținut digital.

În cazul aplicațiilor reale, problemele de autotestare sunt esențiale întrucât absența unei caracteristici de calitate generează decizii care, în unele situații, au efecte ireversibile.

În autotestare, cifrele de control și inegalitățile sau egalitățile a căror existență este demonstrată deja îndeplinesc condițiile concrete etapei.

Autotestarea ia în considerare mult mai multe aspecte de profunzime decât procesul de testare dacă executantul este un profesionist și dacă își impune încadrarea în consumul de resurse planificat sau chiar o economie de resurse pentru activitatea supusă autotestării.

Bibliografie

1. **ANTONIOL, G.**: Keynote Paper: Search Based Software Testing for Software Security: Breaking Code to Make it Safer. Software Testing, Verification and Validation Workshops, 2009. ICSTW '09. Int. Conf. on, 1-4 April 2009, pp. 87-100.

2. **FU-SHIAU, LI, MA WEI-MING, A. CHAO**: Architecture Centric Approach to Enhance Software Testing Management. Intelligent Systems Design and Applications, *2008. ISDA '08.* Eighth Int. Conf. on, Vol. 1, 26-28 Nov. 2008, pp. 654-659.

3. **IVAN, I., C. BOJA, C. CIUREA**: Metrici ale sistemelor colaborative, Editura ASE, Bucureşti, 2007.

4. **POCATILU, P.**: Costul testării software, Editura ASE, Bucureşti, 2004.

5. **IVAN, I., P. POCATILU**: Testarea automată a produselor software specializate. Revista Informatica Economică, vol. VIII, nr. 2(30), 2004, pp. 116-120.

6. **IVAN, I., C. TOMA**: Testarea interfeţelor om-calculator. Revista Română de Informatică şi Automatică, vol. 13, nr. 2, 2003, pp. 22 – 29.

7. **NORLIN, E. C. M. WINTERS**: Usability Testing for Library Web Sites: a Hands-on Guide, American Library Association, 2002.

8. **KANER, C., J. FALK, Q. NGUYEN**: Testing Computer Software, John Wiley & Sons, 1999.