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

Abstract. 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 şi 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 şi condiționează această trecere. Se execută de către cei care desfăşoară activități în etapa curentă.

Este diferită de testare, atât prin obiective şi sarcini, cât şi 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, aşa cum rezultă din specificații.

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

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 şi afişare de rezultate, de informare;
- *aplicații complexe*, în care pe baza datelor furnizate de utilizator, se efectuează alocări de resurse, rezervări, plăți, analize şi 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 şi le selectează după criterii proprii;
- *aplicații cu conținut aşteptat*, în care este prezentată informație care porneşte de la grupul țintă, de la nevoile de cunoaştere, de orientare, de formare ale acestuia;
- *aplicații cu conținut realizat după un şablon*, aşa cum sunt prezentările de universități, de oraşe, de enciclopedii;
- *aplicații cu conținut orientat spre continuitate*, care pornesc de la ceea ce există, ajutându-l pe utilizator, pe baza experienței acumulate, să-şi soluționeze propriile probleme.

În toate cazurile, se porneşte 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 şi 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 dezvoltatorul de conținut digital; se va evita utilizarea unor expresii *posibil ca*, *ar fi dacă*, *şi nu în ultimul rând*;
- *clar*, ceea ce presupune o dezvoltare a unei structuri simple, articulate, acceptate şi pe înțeles, cu abordare graduală folosind elemente sugestive;
- *omogen*, din punct de vedere al structurii şi din punct de vedere al nivelului de abordare; se construieşte un şablon şi se urmăreşte 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, daca se forțează lucrurile, folosind numai o regulă.

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

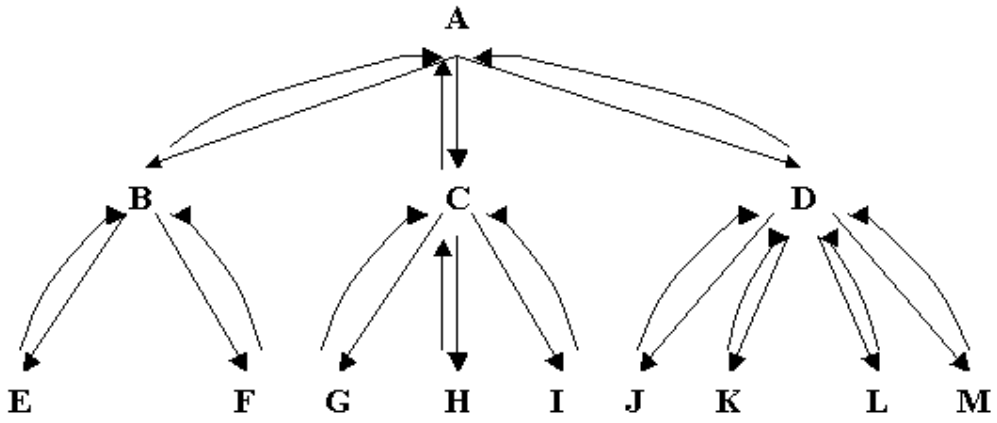


Fig. 1. Structură arborescentă de conținut cu referire din aproape în aproape

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

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

Între o aplicație sofisticată, nefuncțională la momentul stabilit, sau care va avea probleme tot timpul, şi 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

Conținutul oferit este, de regulă, dat sub forma unei structuri arborescente, ca în figura 2:

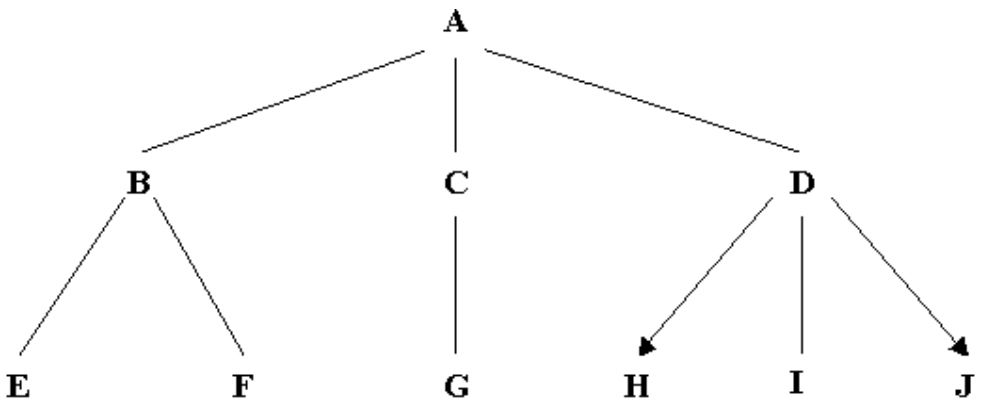


Fig. 2. Structură arborescentă a conținutului oferit

Rezultă că B, C şi 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 îşi 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 şi cel referit are la bază numeroase cauze, dintre care cea mai frecventă este legată de codificările construite pentru fişiere, mai ales atunci când se lucrează cu un număr foarte mare de fişiere.

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

Cauza este simplă: fişierele 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 fişierului 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.

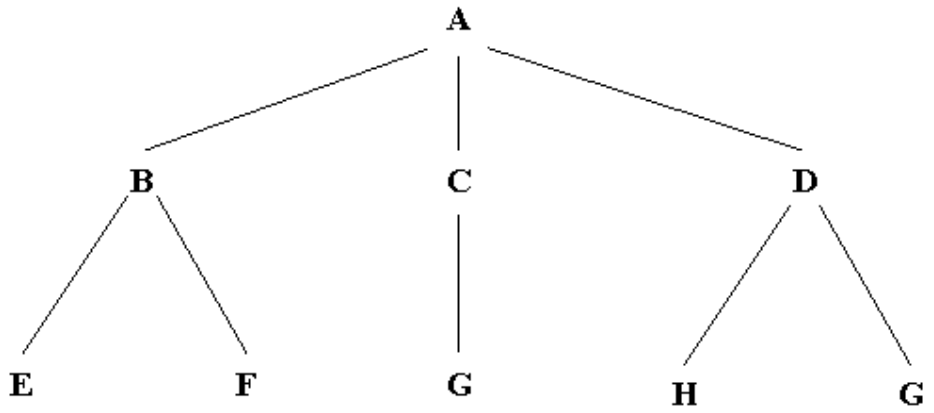


Fig. 3. Structură planificată

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

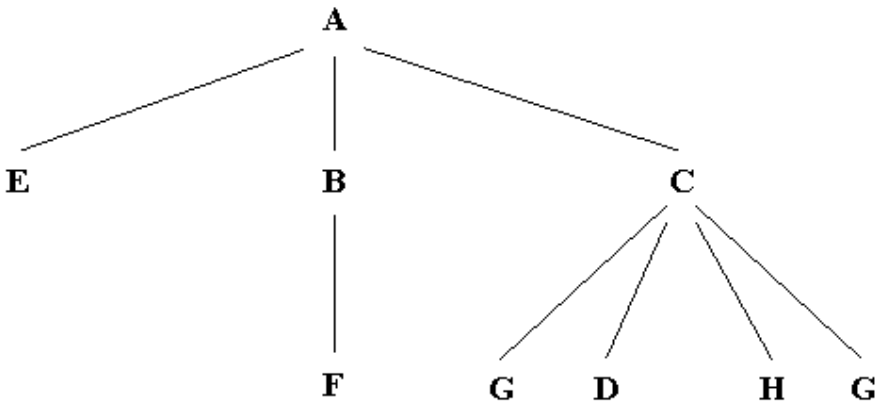


Fig. 4. Structură implementată

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

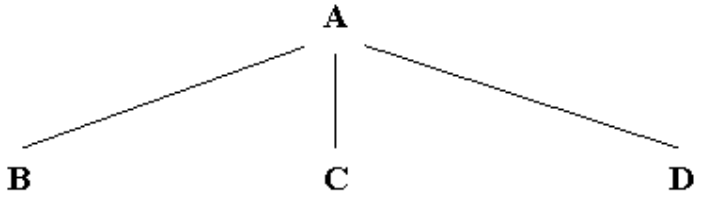


Fig. 5. Structură planificată

În figura 6 se prezintă structura implementată a unei aplicații informatice distribuite, în care conținutul nu este referit în totalitate:

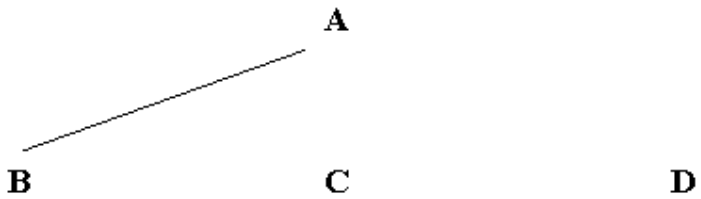


Fig. 6. Structură implementată

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.

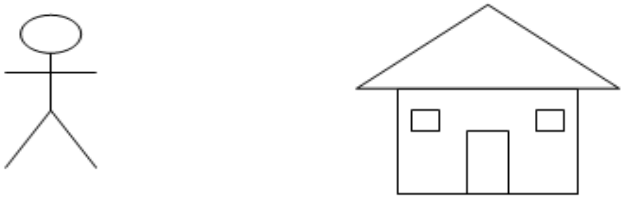
Erorile de asociere de fişiere

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:

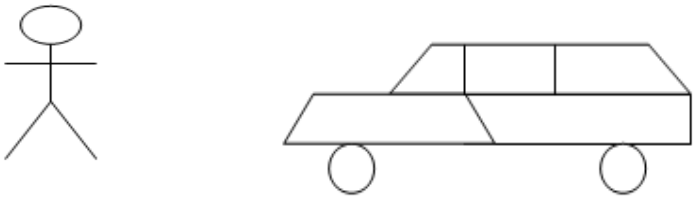
F_1 **F_2**



Corect

Text 1 **Text 2**

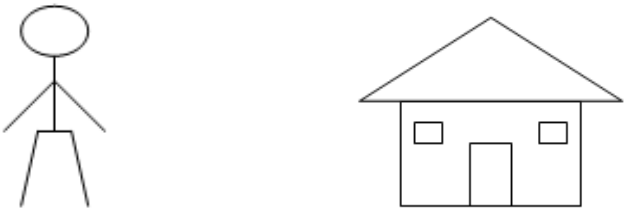
F_1 **F_3**



Incorect, pentru că nu este tabloul cu casa

Text 1 **Text 2**

F_4 **F_2**



Incorect, pentru că nu este pictorul din F_1, ci pictorul din F_4.

Text 1 **Text 2**

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

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

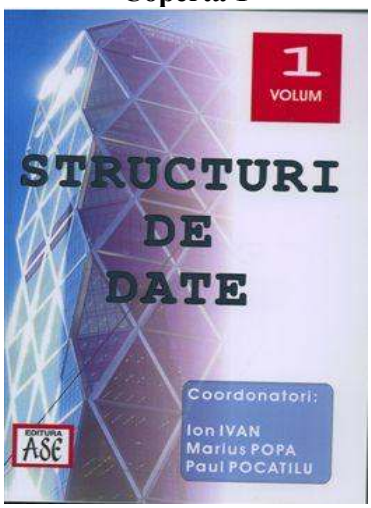
	Portret pictor	**Tablou**	**Text sub pictor**	**Text sub tablou**	
Erori înlocuire imagini	F1	F2	T1	T2	Corect
	F1	F3	T1	T2	Incorect, apare F3 în loc de F2
	F4	F2	T1	T2	Incorect, apare F4 în loc de F1

	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, explicația T2 este înlocuită cu T3
	F1	F2	T4	T2	Eronat, explicația T1 este înlocuită cu T4
	F1	F2	T4	T3	Toate textele sunt eronate

Soluția este dată de codificare şi de existența în specificații explicit imagini şi texte corecte.

Se consideră două cărți de informatică, ale căror coperți scanate sunt prezentate în continuare. Se asociază câte un comentariu la fiecare carte. În situația în care comentariul asociat unei cărți corespunde cărții respective, atunci asocierea este corectă [3]. Dacă autorii de la o carte sunt trecuți în comentariul aparținând altei cărți, atunci asocierea este greşită.

Coperta 1

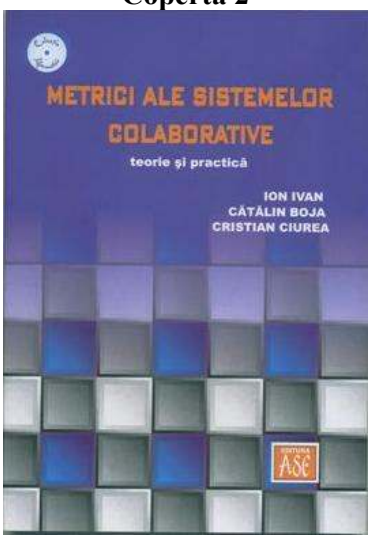


Comentariul 1

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

Corect

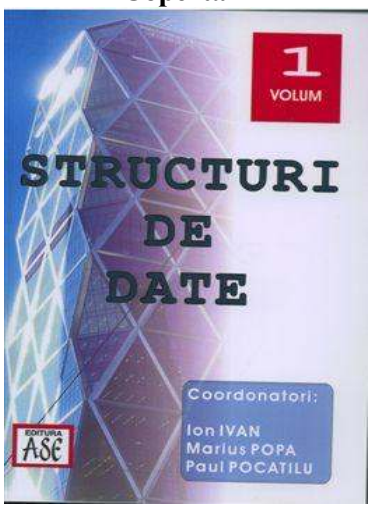
Coperta 2



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 si practică*, Editura ASE, Bucureşti, 2007.	**Incorect, pentru că nu este comentariul asociat cărții de Structuri de date, ci cărții 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 îmbunătățeşte 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 defineşte 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 realizarea unui stadiu al aplicației găsesc erorile mult mai repede şi 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.

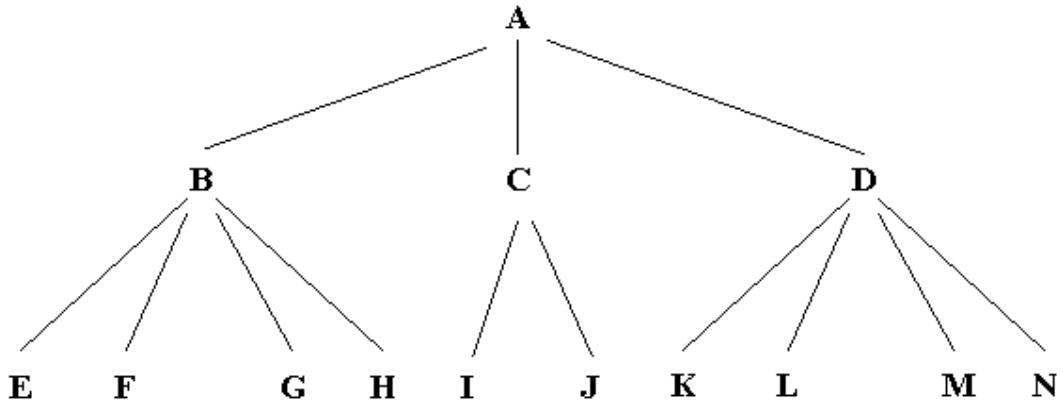


Fig. 7. Structură planificată

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.

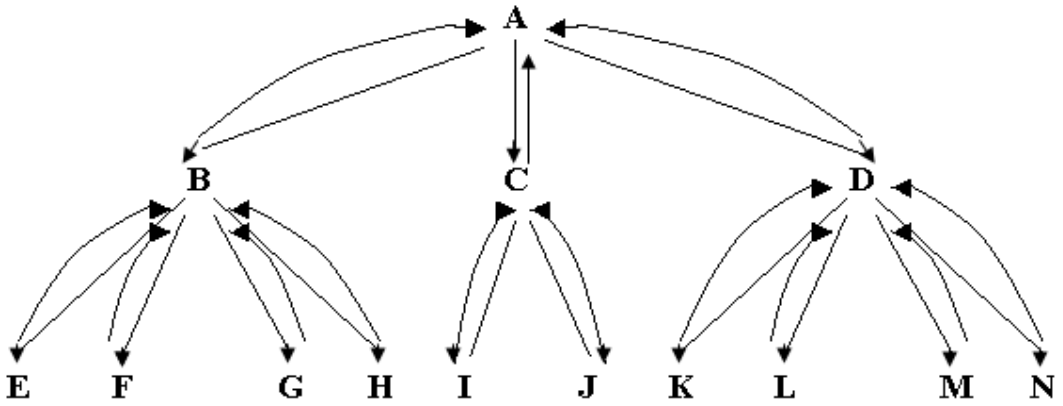


Fig. 8. Structură planificată cu conținut digital oferit cu revenire din aproape în aproape

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 conține:

- mulțimea seturilor de date de autotestare a legăturilor, prima coloană a tabelului 1, în care sunt prezentate cerințele pentru aplicația informatică cu conținut oferit, având structura dată în figura 8;
- mulțimea rezultatelor efective ce vizează comportamentul real al aplicației; în cazul unei construcții corecte, între elementele din prima coloană şi aceasta nu există diferențe; cu cât s-au produs erori la implementare, cu atât diferențele sunt mai mari; erorile sistematice vizează o abordare superficială a problemei, dar şi o soluționare 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 $C_1, C_2, ..., C_N$. Se realizează imaginile $I_1, I_2, ..., I_N$.

Elementele colectivității C se grupează după criteriile $G_1, G_2, ..., G_K$, formând submulțimile $S_1, S_2, ..., S_K$.

La rândul lor, submulțimile $S_1, S_2, ..., S_K$ sunt alcătuite din subsubmulțimile nule $SS_{11}, SS_{12}, ..., SS_{1K1}, SS_{21}, SS_{22}, ..., SS_{2K2}, …, SS_{K1}, SS_{K2}, ..., SS_{KLK}$.

Fiecare submulțime este prezentată prin imagini.

Între submultimile S_i si S_j exista relatia $S_i \cap S_j = \Phi$.

Între submultimile SS_{ij} si SS_{ih} exista relatia $SS_{ij} \cap SS_{ih} = \Phi$.

O subsubmulțime SS_{ij} este obținută prin aplicarea criteriului $GG_{i1}, GG_{i2}, ..., GG_{iLhi}$.

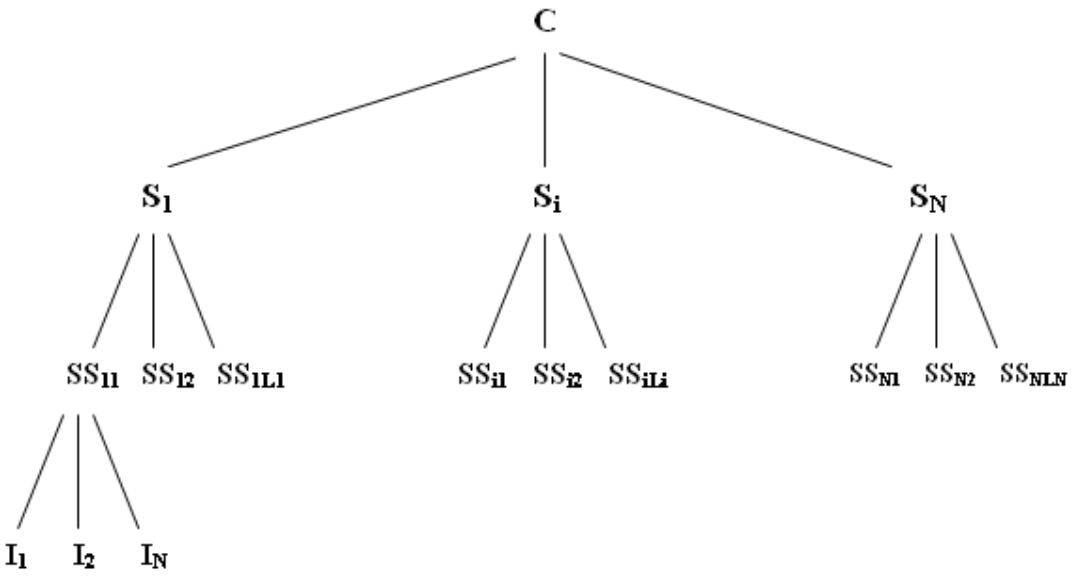


Fig. 9. Legăturile dintre colectivitate, submulțimi, subsubmulțimi şi imagini

Aplicațiile construite pe bază de matriță sau șablon permit crearea de conținut digital standard şi 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 doreşte 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 tara – India (SS_{12}), degetarul de lut (material, Bulgaria, arhitectură), arcul de triumf (țară – Franța, material, tehnică de realizare, monument, similitudine culoare, oraş – 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 afişarea imaginilor unei colecții de degetare decorative, aplicație disponibilă la adresa http://www.degetare.ro:

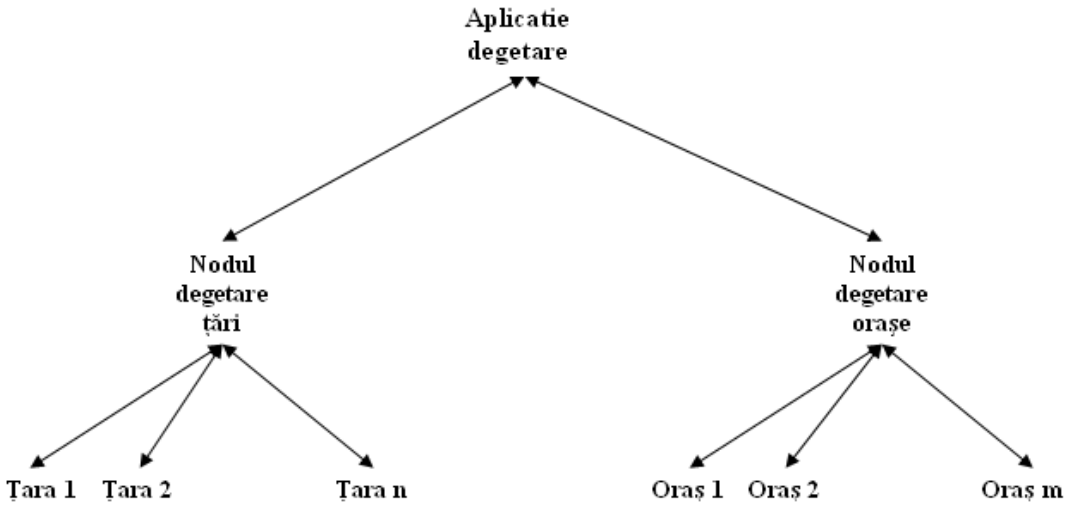


Fig. 10. Aplicație pentru afişarea imaginilor unei colecții de degetare decorative

O aplicație pentru afişarea imaginilor unei colecții de degetare decorative are o structură arborescentă, arborele stâng fiind reprezentat de nodul *degetare țări*, iar arborele drept de nodul *degetare oraşe*. Nodul *degetare țări* va conține imaginile unor degetare din *n* țări, iar nodul *degetare oraşe* 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* fişiere pentru țări şi *m* fişiere pentru orașe. Cele *n* fişiere pentru țări diferă între ele prin imaginea degetarului din țara respectivă, ancorele de revenire fiind identice. Acelaşi lucru este valabil şi pentru cele

m fişiere pentru oraşe. Pentru creşterea productivităţii în activitatea de programare, programatorul trebuie să ţină cont de aceste aspecte.

Dacă se creează $k < n$ fişiere, în loc de *n* fişiere pentru ţări, înseamnă că s-au pierdut o parte din imagini. Conştientizarea acestui aspect rezultă din specificaţiile aplicaţiei, unde este precizat numărul de ţări la care se face referire.

4. Complete self test of the content

In relation with the proposed objective, was built a structure of the application and were developed specifications for achievement. The content specifications have a unique way of achieving. There are elements of creativity on the application development. The essence of the content defined in the specifications should not differ from what contains the real application.

There is the *announced content* given as a title on the *k* level of the structure and the *effectively content* defined on the *k +1* level.

Between the two contents must be concordance, in the sense that the content developer for the *k* level must ensure that the developer of the *k +1* level realize text and images on the same thing. Moreover, at the *k +1* level there are greater responsibilities whereas the actual text must be complete, coherent, correct and consistent, while the text on the *k* level is only a text listing elements or announce a concept in a synthetic form.

Responsibility at the *k +1* level is absolute because the achievement of initial objective depends strictly on the content of this level. The reality shows that the levels 2, 3, 4, etc. are themselves the level *k+1*, which shows that the responsibility is fully transferred to the structures.

There is a separate way at the leaves level of the tree structure to ensure:

- descriptions details;
- composition, if that were not included on previous levels.

When performing content self test must follow every way from root to leaf, to see:

- *the consistency*, being defined as an application characteristic to not be in contradictory, to be resistant at security attacks;
- *the accuracy* is the quality of an application to behave properly, to respect the functioning rules, to be free of mistakes;
- *the completeness* is the property of an application to contain all the elements necessary for correct functioning and to provide all information requested and expected by users;
- *the clarity*, being defined as the content property of an application to be intelligible, transparent and easily to understand.

The lack of consistency in an application with offered digital content means the sensitivity of the application to security attacks and high number of views, the lack of acuracy requires a random functioning, the lack of completeness refers to providing partial information, and lack of clarity means an obscurely content [6].

It is considered a painter who gives his collection of paintings to the city hall in order to make a permanent exhibition. The lack of consistency means the existence of fragile and easily damaged paintings, the lack of accuracy means associate the label describing a picture to another picture, the lack of completeness refers to an incomplete collection of paintings, and lack of clarity concerns the descriptions of paintings containing texts difficult to understand.

When discrepancies occur on recourse to:

- pass the content from the *k+1* level to the *k* level or if on the *k* level are details, these will pass on the level *k+1*;

- replace images and text;
- interchange images and text on the same level;
- delete images or text;
- insert images or text.

All these are meant to increase the quality of the content. Are removed the redundancy or inconsistencies.

The contents of an application must be tested in detail, depending on the application type and its users, because it must provide accurate and complete information, especially updated.

An informatic application must contain the terms of use and the content acceptance by the users. Depending on the application type, in the terms of use will clearly specify that the information presented in the application do not have contractual value, are indicative, and the application owner is not responsible for content errors or inconsistencies with reality.

5. Ways to increase the quality of the offered digital content

The stuctured specifications development of the content, so that to each component separately defined will correspond a link in the informatic application, so a file, is achievable if and only if they who define the digital content have a clear image of the whole, but also on the details, the top-down decomposition being at hand in an easy way.

If the specifications are not structured, the developer shall be responsible for the decomposition in parts of the digital content. If are several developers which work after defined concepts of structure, some after complexity, others by meaning, while others by homogeneity, that will certainly occur multiple defined elements, so as appear undefined elements. In both cases, there is a risk of multiple references, but also for the absence of references, like in figures 11 and 12:

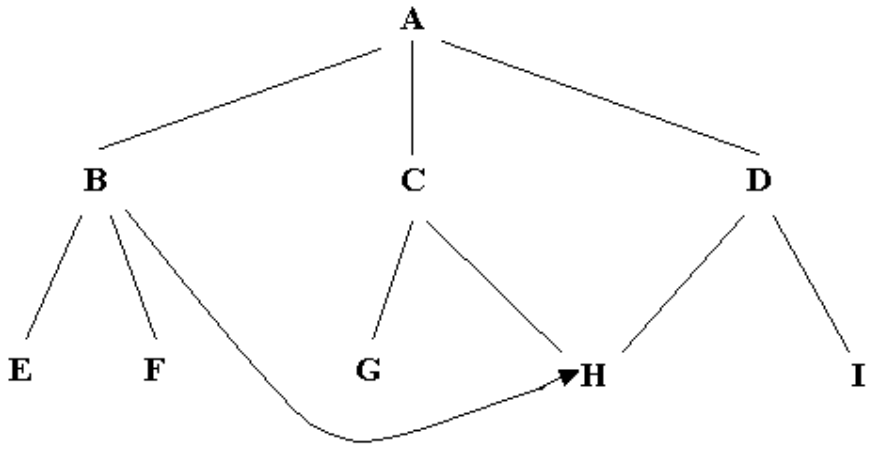


Fig. 11. Multiple references of the node H

In figure 12 is presented an application structure having content not referred by a part of nodes:

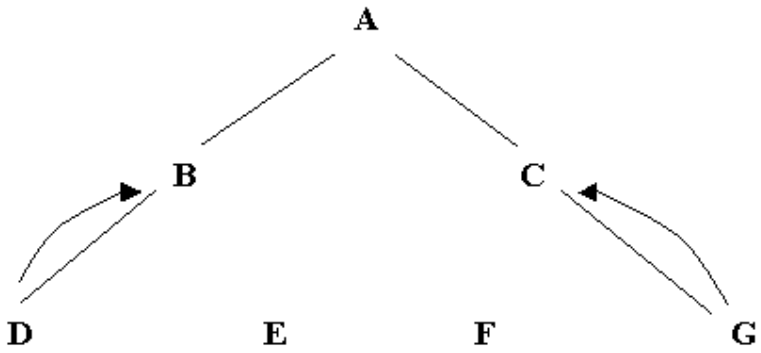


Fig. 12. Content not referred by the nodes E and F

Creating molds for the types of files developed by the same rules, such the template, the developers running only their copy in empty files and adapting their text to the sequence of digital content that is referred. The mold is self tested so that achieve with certainty the type of feature for which was created. It also include the sequence that provides all kinds of comebacks, or at the *k-1* level if it is on level *k*, or at k_0 level if it is acceptable to return to the root of the tree structure [7].

The mold contains all the tabular structures, in fact, for defining the image, for alignment, achieving in this way the homogeneity of all texts on the same level.

Ensure the management through codes and control numbers involves determining the number of nodes on each level and, thus, will know the exact number of files per level.

And within each digital content, the files will be managed with images that relate.

To achieve the file management is necessary to define a coding system that:

- ensure the elements differentiation;
- obtain the belonging identification to a homogeneous subcolectivity;
- maintain the creation of an ascending ordered sequence;
- check if the number of files scheduled to be realized and loaded are loaded into the application and, moreover, are referenced, and at their turn relate.

The self test concerns the extent to which the developer had the ability to properly encode files that define parts of the digital content offered.

Construction of parts of the digital content, the integration in the application and self test achievement in place and of the manner in which the new component is referred to. To allow further developments the content end with a text which mark that it is *in work now* or *under construction* or is *provisional*. It is preferable to indicate when is ending the provisional character of the respectively digital content.

Checking the text contained by each file that is loaded in order to refer it from the *k* level, but also refering from it other files from the *k +1* level. Is checking that:

- the name of the referred file from the expression from *k* level is identical to the filename loaded for the *k +1* level;
- the reference text from the level *k* is in concordance with the file content referred from the *k +1* level;
- the content structure is correct in terms of syntactic and generate exactly what is given in the specifications.

It is believed that the reality contain the components $C_1, C_2, ..., C_n$. Starting from these components, the digital content created will be composed from files $F_1, F_2, ..., F_m$. The content inserted into the application contain the elements $G_1, G_2, ..., G_k$.

If $m > n$, then there is the content C_i on which correspond at least two files F_k şi F_{k+1}.

If $m < n$, then there are the contents C_i si C_j merged in the file F_k.

If $m = n$, then to each content C_i correspond a file F_i.

Order in sequence

If S is the structure of the real content and S' is the structure in site, then S' must not differ significantly from S.

If $m = k$, means that to the file F_i correspond the component G_i inserted (ideal case, C_i -> F_i -> G_i).

If $m > k$, means that was used a file F_j.

If $m < k$, means that were inserted files from other applications înseamna ca au fost inserate fişiere din alte aplicații (is seriously if in more applications are the same codes).

All of these imply the acceptance of a routine activity in the development of the offered content, so as the professionalism cover both the quality side, but mostly the quantitative side. The qualitative aspect concerns the molds which, in their diversity, cover types of digital content of high quality in terms of structural and the reference [8]. The quantitative aspect concerns that all files be with very good quality, through the molds reuse.

6. Conclusions

The self test is a new concept which is taken from traditional medical practice, which prevent is much more effective than dealing with a suffering. Since self test is achieved by the application developer himself, he acknowledges that it is more profitable for him to achieve at the beginning the correct, complete, consistent and clear components of the digital content, whereas their aggregation will lead, ultimately, to a digital content, also correct, complete, consistent and clear. The risk that occurs when self testing an application by the developer himself is that some errors go unnoticed. Is recommended the principle of the four eyes to avoid such a situation.

In the case of testing, the developer made components, the others – the testers – check them and, often, makes troubleshooting a third category, so that the transfer of tasks is delegated, with the most unfavorable consequences.

The self test assume that the same team to achieve:

- to take the specifications;
- realize the digital content;
- check the links between levels;
- measuring the level of quality characteristics;
- identify the errors;
- correct the errors.

Each time will know exactly who, when, how, how much, where on came and with what results.

The offered digital content, but which result from processing, must comply with the mandatory requirements of copyright laws. Also, are correct and complete presented the source and location like volume, page, author, etc.

Excepting the self test made by the development team, the informatics applications must be able to test themselves. Designing and implementing an application must consider the introduction of counter variables, setting of indicators by which to verify the application functionality. The self test results are delivered in the form of logs, which the application administrator view and take the necessary steps to correct the occured errors. The application will work like a collaborative system in which all cooperate to achieve a common goal represented by a correct and permanent functioning.

In distributed applications, there is a tendency towards an exaggerated level of creativity. It requires the creation of templates, the definition of rules of broad acceptance such that for

classes of offered digital content to exist nearby elements from the structural point of view. For many homogeneous elements, where it identifies a template and a set of accepted rules, shall be handled without major risk problems of:

- crossing the digital content;
- retrieving the digital content;
- searching the digital content.

In the self testing, the control numbers and the inequalities or equalities of whose existence is already demonstrated fulfill the concrete conditions of the stage.

The self test consider more depth issues than the test process if the execution person is a professional and if he require himself to include in the planned consumption of resources or even a economy of resources for the activity under self test.

References

[1] G. Antoniol, "Keynote Paper: Search Based Software Testing for Software Security: Breaking Code to Make it Safer," *Software Testing, Verification and Validation Workshops, 2009. ICSTW '09. International Conference on*, pp.87-100, 1-4 April 2009.

[2] Li Fu-Shiau, Ma Wei-Ming and A. Chao, "Architecture Centric Approach to Enhance Software Testing Management," *Intelligent Systems Design and Applications, 2008. ISDA '08. Eighth International Conference on*, vol. 1, pp. 654-659, 26-28 Nov. 2008.

[3] I. Ivan, C. Boja and C. Ciurea, *Metrici ale sistemelor colaborative*, Editura ASE, Bucuresti, 2007.

[4] P. Pocatilu, *Costul testării software*, Editura ASE, Bucuresti, 2004.

[5] I. Ivan and P. Pocatilu, "Testarea automată a produselor software specializate," *Revista Informatica Economică*, vol. VIII, nr. 2(30), 2004, pp. 116-120

[6] I. Ivan and C. Toma, "Testarea interfețelor om-calculator," *Revista Română de Informatică şi Automatică*, vol. 13, nr. 2, 2003, pg. 22 – 29.

[7] E. Norlin and C. M. Winters, *Usability testing for library Web sites : a hands-on guide*, American Library Association, 2002.

[8] C. Kaner, J. Falk, Jack and H. Q. Nguyen, *Testing computer software*, John Wiley & Sons, 1999.