O propunere de arhitectură pentru evaluarea metricilor de execuţie folosind AOP

Adriana-Mihaela Guran
Universitatea Babeş-Bolyai
Str. M. Kogălniceanu nr. 1, Cluj-Napoca
adriana@cs.ubbcluj.ro

Grigoreta Sofia Cojocar
Universitatea Babeş-Bolyai
Str. M. Kogălniceanu nr. 1, Cluj-Napoca
grigo@cs.ubbcluj.ro

REZUMAT

Evaluarea utilizabilității rămâne un subiect de mare interes în domeniul interacțiunii om-calculator. Întrucât acest proces este unul costisitor, eforturile de a găsi metode *eficiente* de evaluare a utilizabilității sunt îndreptate în direcția identificării de oportunități de automatizare, care să suplinească prezența experților umani. Programarea orientată pe aspecte propune soluții avantajoase din punctul de vederii al gestiunii instrumentării codului, dar lipsesc instrumentele de modelare care să facă proiectarea unor astfel de module mai uşor de creat, reutilizat şi înțeles. Această lucrare propune o arhitectură pentru un cadru de evaluare a utilizabilității dezvoltat folosind paradigma orientată pe aspecte.

Cuvinte cheie

utilizabilitate, evaluare, metrici, AOP.

Clasificare ACM

H5.2. Information interfaces and presentation (e.g., HCI): Evaluation/methodology.

INTRODUCERE

Dezvoltarea fără precedent a tehnologiei în ultimele decenii şi accesul tot mai larg al populației la Internet şi la dispozitive inteligente, face ca experiența utilizatorului în interacțiune să devină factor decisiv în succesul sau eşecul unui produs [11, 12, 15, 16].

Utilizabilitatea, parte integrantă şi determinantă a experienței utilizator, se află în centrul atenției cercetărilor din domeniul HCI de o bună perioadă de timp. Cu toate acestea, nu există abordări uniform acceptate la nivelul definiției acesteia, însă ca o componentă comună a definițiilor propuse se remarcă faptul că utilizabilitatea este asociată atributelor funcționale ale sistemelor interactive.

Evaluarea utilizabilității unui produs interactiv este o componentă fundamentală a proiectării centrate pe utilizator, dar rămâne în continuare o activitate caracterizată drept costisitoare.

Automatizarea evaluării utilizabilității ar contribui în mod determinant la scăderea costurilor de evaluare, iar posibilitatea reutilizării modulelor de evaluare automată a utilizabilității ar conduce în mod evident la sporirea eficienței unei astfel de soluții. În acest articol propunem arhitectura unui cadru pentru evaluarea unui set de metrici ale utilizabilității dezvoltat folosind paradigma orientată pe aspecte. Articolul este organizat după cum urmează: o secțiune dedicată utilizabilității în contextul experienței utilizator şi metodelor de evaluare a acesteia, cu accent pe metodele automate, o secțiune dedicată paradigmei de programare orientată pe aspecte, o secțiune dedicată arhitecturii de evaluare automată a utilizabilității dezvoltat folosind AOP şi o secțiune de concluzii şi direcții ulterioare de cercetare.

UTILIZABILITATEA

Utilizabilitatea a fost definită în numeroase forme de-a lungul timpului. De o largă acceptabilitate se bucură definiția ISO [9] care stabileşte criteriile utilizabilității a fi eficiența, eficacitatea şi satisfacția utilizatorului.

Viziunea tradițională asupra utilizabilității se referă la aspectele interfeței care uşureaza utilizarea produsului [3]. Nielsen [14] defineşte utilizabilitatea prin următoarele atribute ale unui sistem: uşurința de învățare, eficiența, memorabilitatea, erorile şi satisfacția în utilizare. Dix [6] defineşte utilizabilitatea în termeni de învățare, flexibilitate şi robustețe. Astfel, putem remarca faptul că definițiile larg acceptate şi vehiculate în literatura de specialitate surprind caracteristici ale sistemului, fapt care le face abordabile dintr-o perspectivă măsurabilă, care poate implica soluții automatizate parțial.

Evaluarea utilizabilității se realizează prin diverse metode, cele mai eficiente şi în acelaşi timp cele mai costisitoare, sunt metodele care folosesc testarea cu utilizatori reali. Tipic, la un test de utilizabilitate evaluatorul propune un scenariu de utilizare a sistemului informatic. Sarcinile incluse în scenariu au scopul de a testa anumite aspecte ale interfeței utilizator. În timpul testului de utilizabilitate specialistul în utilizabilitate observă modul în care utilizatorul execută sarcinile, comportamentul său şi comentariile pe care le face. La sfârşitul testului are loc o discuție privind modul de realizare a sarcinilor şi gradul de satisfacție al utilizatorului în raport cu sistemul folosit.

Deoarece producătorii de sisteme informatice sunt interesați de dezvoltarea de sisteme cu un grad cât mai sporit de utilizabilitate, pentru aceştia este nevoie de dezvoltarea de metode mai puțin costisitoare pentru realizarea evaluării utilizabilității unui sistem informatic.

Una din metodele cele mai uşor de aplicat în mod automat este evaluarea unor metrici ale utilizabilității precum numărul de sarcini realizate cu succes, numărul de sarcini eşuate, timpul de execuție a unor sarcini, numărul şi frecvența erorilor, frecvența de utilizare a unor

funcţionalităţi ale aplicaţiei, frecvenţa de accesare a documentaţiei de utilizare, etc.

Evaluarea utilizabilităţii

Argumentele pentru evaluarea utilizabilităţii sunt întotdeauna legate de dorinţa de îmbunătăţire [18, 21]. Scopul acestei activităţi este de a permite comparaţia cu competitorii, precum şi identificarea aspectelor problematice care necesită concentrarea eforturilor proiectanţilor.

Conceptul de utilizabilitate pare a fi unul complex şi greu de măsurat, totuşi autori precum Tullis [21] , Seffah [18] sau Sauro [17] susţin că utilizabilitatea poate fi evaluată prin intermediul metricilor, chiar dacă există unele aspecte care sunt mai dificil de evaluat. Metrica este definită ca o funcţie care se aplică pe date legate de soft şi al cărei rezultat este o valoare numerică ce poate fi interpretată ca fiind gradul în care softul deţine un atribut cu efect asupra calităţii sale.

Metricile de utilizabilitate sintetizează informaţii despre interacţiunea dintre utilizator şi sistem care ţin de eficienţă (consumul de resurse în raport cu rezultatele aşteptate), eficacitate (capacitatea de a realiza o sarcină) sau satisfacţie (măsura în care utilizatorul e fericit cu experienţa sa de interacţiune).

Câteva din metricile sugestive pentru calitatea interacţiunii dintre utilizator şi un sistem sunt: numărul de sarcini realizate cu succes, durata execuţiei unei sarcini, numărul de clickuri, numărul tastelor activate, frustrarea sau satisfacţia în utilizare [21]. Aceastea oferă informaţii despre caracteristici ale utilizabilităţii precum eficienţa şi satisfacţia menţionate atât în standardul ISO [9] cât şi în modelul QUIM asupra calităţii sistemelor soft [18].

Evaluarea utilizabilităţii este o componentă majoră din dezvoltarea de interfeţe utilizator care încearcă identificarea problemelor de utilizabilitate. Indiferent de abordarea adoptată în evaluarea utilizabilităţii, sunt efectuate următoarele trei activităţi:

- achiziţia – se referă la culegerea de date de utilizabilitate, precum timpul de efectuare a unei sarcini, încălcări ale regulilor de proiectare, erori;
- analiza – se referă la interpretarea datelor de utilizabilitate pentru identificarea problemelor;
- propunerea de sugestii de îmbunătăţire a interfeţei utilizator [16, 21].

Primele două activităţi din cele amintite mai sus pot fi supuse unui proces de automatizare.

Avantajele pe care le aduce automatizarea evaluării utilizabilităţii sunt: reducerea necesarului de resurse umane, reducerea necesarului de expertiză umană, completitudine şi posibilitatea comparării rezultatelor testelor de utilizabilitate.

Automatizarea procesului de evaluare a interfeţei utilizator necesită jurnalizarea evenimentelor din interacţiunea dintre utilizator şi sistemul supus analizei. Jurnalizarea şi analiza evenimentelor care apar la nivelul interfeţei utilizator au fost recunoscute a fi surse valoroase de determinare a problemelor de utilizabilitate. Prin jurnalizarea evenimentelor din interfaţa utilizator se pot obţine informaţii precise despre ce face utilizatorul, ce date foloseşte, ce funcţionalităţi accesează, în ce ordine execută acţiunile. Astfel, metrici ale utilizabilităţii precum timpul de execuţie a unei sarcini, frecvenţa de folosire a unei funcţionalităţi sau numărul de erori pot fi calculate din fişierele de jurnalizare.

Desigur, există şi alte modalităţi de evaluare a acestor metrici (înregistrarea video a sesiunii de testare, notarea manuală), dar folosind jurnalizarea evenimentelor acest lucru se poate realiza automat. Datele înregistrate în fişierele de jurnalizare au un format specific care poate fi transformat în alte formate, operaţie urmată de supunerea datelor spre analiză.

Jurnalizarea automată a evenimentelor din interfaţa utilizator are şi neajunsuri, unul din acestea fiind imposibilitatea surprinderii reacţiilor utilizatorului. În mod obişnuit jurnalizarea se realizează prin inserarea unor linii de cod în cadrul codului aplicaţiei pentru a se surprinde elementele de interes din cadrul interacţiunii. Această abordare necesită o foarte bună cunoaştere a structurii aplicaţiei şi acces la codul sursă al aplicaţiei. Decizia asupra locaţiilor în care să se facă instrumentarea trebuie să aparţină unui expert în utilizabilitate care să colaboreze cu dezvoltatorii sistemului. În secţiunile următoare propunem o arhitectură pentru evaluarea automată a metricilor de execuţie dezvoltat folosind programarea orientată pe aspecte.

PROGRAMAREA ORIENTATĂ PE ASPECTE

Programarea orientată pe aspecte (AOP) este o paradigmă recent dezvoltată care se adresează situaţiilor de cerinţe transversale (o caracteristică a unui sistem a cărei implementare este distribuită în tot sistemul) [10]. Pentru implementarea unor astfel de cerinţe AOP introduce patru noţiuni noi :

- join-point - punct bine definit în execuţia programului;
- pointcut (ro. punct de tăietură)- selectează o mulţime de join-pointuri şi expune anumite valori din contextul de execuţie al join-point-urilor;
- advice - secvenţă de cod care se execută la fiecare join-point dintr-un pointcut;
- aspect - este un tip care încapsulează pointcut-uri, advice-uri şi proprietăţi statice [10].

Un aspect este unitatea de modularizare a AOP. Aspectele sunt integrate în sistem folosind un instrument special numit weaver. Există extensii AOP pentru limbaje de programare bine-cunoscute precum Java [2] sau C++ [1].

Activitatea de culegere de date din evaluarea utilizabilităţii constă în jurnalizarea evenimentelor care apar în timpul interacţiunii utilizator-sistem precum apăsarea unui buton, traversarea unui meniu, alegerea unei comenzi, începerea execuţiei unei sarcini.

Inserarea codului necesar jurnalizării în sistemele deja dezvoltate necesită modificarea tuturor modulelor interfeţei grafice, dar folosind AOP este necesară doar scrierea unor aspecte şi integrarea acestora folosind weaver-ul. Activitatea de analiză constă în evaluarea unor metrici de utilizabilitate. Folosind AOP calculul metricilor de utilizabilitate poate fi realizat în timpul interacţiunii.

Avantajele pe care le aduce folosirea AOP în evaluarea utilizabilității sunt:

- unele metrici pot fi calculate automat scăzând efortul necesar analizei;
- evenimentele sunt jurnalizate cu uşurință pentru analiza detaliată ulterioară;

Avantajele utilizării programării orientate pe aspecte în jurnalizarea interacțiunii sunt susținute de folosirea tot mai frecventă a acesteia pentru instrumentarea codului, în defavoarea instrumentării manuale.

În [8], autorii abordează problema instrumentării automate a codului cu scopul de a completa datele primare de interacțiune (clickuri, apăsari de taste) cu informații contextuale care să ajute la interpretarea modului în care este utilizat sistemul. Autorii însă nu își orientează cercetarea înspre o anumită metodă de evaluare a utilizabilității în care informațiile de interacțiune reconstituite să fie folosite.

Majoritatea aplicațiilor cu interfețe grafice respectă şablonul MVC, astfel că în [12] este prezentat un exemplu de instrumentare a codului în vederea culegerii de date

- dacă se consideră necesară tratarea unor noi situații, sunt necesare doar modificări la nivelul aspectelor.

ABORDĂRI SIMILARE (SITUAȚIA CURENTĂ)

pentru evaluarea utilizabilității. Autorii menționează că sunt culese informații legate de evenimente relative sarcinilor, dar nu oferă nici o indicație asupra modului în care datele culese sunt relaționate sarcinilor sau aspecte relative sarcinilor care pot fi măsurate pornind de la aceste date. Bateman at al. [3] abordează problema evaluării automate a utilizabilității folosind AOP pentru construirea unui instrument interactiv de desemnare a componentelor din interfața grafică si a evenimentelor care vor fi supuse jurnalizării. Datele colectate sunt analizate la nivel primar însă, limitându-se la număr de clickuri per componentă sau frecvența cu care evenimentele pe o altă componentă din interfața grafică succed evenimentelor pe o componentă supusă analizei.

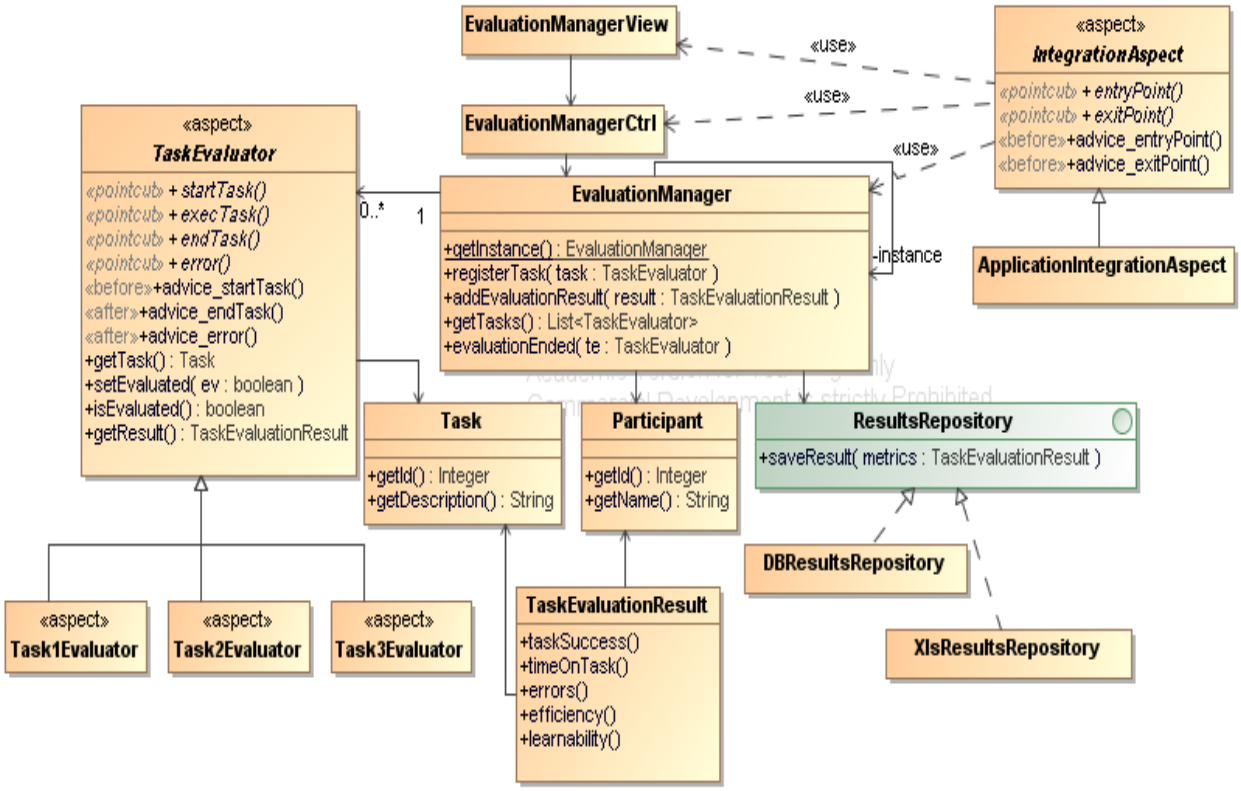

Figura 4. Arhitectura cadrului de evaluare bazat pe AOP

Spre deosebire de abordarea prezentată în [5], care propune un framework de evaluarea automată a utilizabilității unor situri web bazată pe evaluarea respectării unui set de ghiduri de utilizabilitate, propunerea noastră se referă la aplicații desktop, întrucât nu există până în prezent extensii AOP pentru limbajele de programare pentru web, iar evaluarea utilizabilității, realizată din perspectiva metricilor de execuție, se face pe baza unor date din interacțiunea utilizatorului cu aplicația.

În [18] autorii propun un model de evaluare a utilizabilității care încearcă unificarea abordărilor anterioare referitoare la factorii, criteriile şi metricile care caracterizează utilizabilitatea şi oferă un instrument care ajută dezvoltatorii în crearea unui plan de evaluare a utilizabilității pornind de la modelul descris. Autorii evidențiază importanța posibilității automatizării

procesului de evaluare pornind de la modelul de măsurare a utilizabilității. Abordarea propusă în această lucrare se referă strict la un subset de metrici ale utilizabilității, şi anume cele bazate pe date din timpul execuției sarcinilor utilizatorului cu sistemul. Metricile sunt evalaute automat în timpul execuției, iar soluția pe care o propunem poate fi integrată mai multor sisteme pentru care se doreşte evaluarea metricilor de execuție pentru care este oferit suport.

Tahir şi Rodina [19] folosesc programarea orientată pe aspecte pentru a evalua calitatea sistemelor software prin evaluarea unor metrici care reflectă uşurința de întreținere a codului. Evaluarea se face pe baza identificării relațiilor de cuplare între clase.

In lucrarea [20] autorii prezintă o abordare de jurnalizare a evenimentelor din interfața utilizator folosind paradigma AOP. Asupra datelor de jurnalizare se realizeaza o procesare care are ca scop identificarea sarcinilor care au fost executate, pe baza unor şabloane.

Această lucrare se constituie într-o continuare a rezultatelor obținute în direcția evaluării automate a utilizabilității prezentate în [13] şi [22]. Îmbunătățirea majoră pe care o propune această lucrare este soluția de integrare a modulelor de evaluare automată bazată pe metrici de execuție oricărui sistem care este dezvoltat într-un limbaj pentru care există o extensie AOP. Arhitectura pe care o propunem permite configurarea sarcinilor care vor fi supuse evaluării, a metricilor care se doresc a fi evaluate într-o sesiune de testare şi integrarea cu multiple sisteme care se doresc a fi evaluate.

ARHITECTURA CADRULUI DE EVALUARE A UTILIZABILITĂȚII BAZAT PE AOP

În proiectarea cadrului de evaluare bazat pe AOP am încercat să respectăm câteva cerințe importante, cum ar fi: posibilitatea alegerii sarcinilor care urmează a fi evaluate, asocierea sarcinilor evaluate cu participantul, salvarea automată a rezultatelor obținute şi integrarea uşoară a modulului de evaluare cu aplicația ce urmează a fi evaluată. In versiunea curentă, cadrul poate fi folosit pentru evaluarea a trei metrici de execuție: succesul, timpul de execuție şi erorile care au apărut în timpul execuției.

În Figura 1 este prezentată arhitectura cadrului de evaluare. Pentru reprezentarea conceptelor specifice programării orientate pe aspecte am folosit următoarele notații: un aspect este definit ca şi un stereotip "aspect" asociat unei clase, un advice este o metodă având asociat un stereotip corespunzător tipului de advice ("before", "after" sau "around"), iar punctul de tăietură este reprezentat ca o metodă care are asociat stereotipul "pointcut".

Cadrul propus este compus din:

- un aspect abstract **`TaskEvaluator`** care declară puncte de tăietură abstracte pentru începerea execuției (**`startTask`**), execuția (**`execTask`**) şi sfârşitul execuției unei sarcini (**`endTask`**), şi pentru apariția unei erori în timpul execuției unei sarcini (**`error`**). Pentru a determina execuția cu succes a unei sarcini, **`TaskEvaluator`** defineşte un **`after`** advice pentru punctul de tăietură **`endTask`**. Pentru a calcula timpul de execuție **`TaskEvaluator`** defineşte un before advice pentru **`startTask`**() şi after advice pentru **`endTask`**(), iar pentru a memora/înregistra erorile ce au apărut în timpul execuției unei sarcini, s-a definit un after advice pentru punctul de tăietură **`error`**(). Pentru fiecare sarcină ce se vrea a fi evaluată trebuie definit un nou aspect care moştenește din **`TaskEvaluator`** şi defineşte punctele de tăietură abstracte. Fiecare **`TaskEvaluator`** are asociată o sarcină ce conține un identificator unic şi o descriere a sarcinii care o evaluează (clasa **`Task`**).
- un **`EvaluationManager`** care păstrează toate sarcinile ce pot fi evaluate (pentru care există definit un **`TaskEvaluator`**). Fiecare **`TaskEvaluator`** definit se înregistrează la **`EvaluationManager`** în momentul creării. Deoarece pentru fiecare instanță a aplicației evaluate este nevoie de un singur **`EvaluationManager`**, aceasta clasa este definită ca şi Singleton [7]. Definirea ca singleton permite si accesul uşor a fiecărui **`TaskEvaluator`** la **`EvaluationManager`** apelând metoda **`getInstance`**(). Managerul păstrează şi informațiile despre participantul care participă la evaluare: identificatorul unic şi numele (clasa **`Participant`**).
- Pentru fiecare sarcină executată de către participant se va crea o instanță de tip **`TaskEvaluationResult`**. Dacă sarcina este executată cu succes, **`TaskEvaluator`**-ul asociat va adăuga rezultate la **`EvaluationManager`** (apelând metoda **`addEvaluationResult`**). Pentru sarcinile care sunt abandonate sau nu se încheie cu succes, înainte de închiderea aplicației, **`EvaluationManager`**-ul cere **`TaskEvaluator`**-ului corespunzător rezultatele evaluării.
- un depozit **`ResultsRepository`** care permite salvarea unui nou rezultat. Rezultatele pot fi salvate în diferite formate: într-o bază de date, într-un fişier Excel, etc. Cadrul permite configurarea, respectiv particularizarea modului în care se vor salva datele.
- o interfață grafică pentru **`EvaluationManager`** care permite înregistrarea datelor despre participantul care va executa sarcinile şi selectarea sarcinilor care vor fi evaluate. Clasele **`EvaluationManagerView`**, respectiv **`EvaluationManagerCtrl`** sunt folosite pentru construirea interfeței grafice.
- Un aspect abstract **`IntegrationAspect`** care permite integrarea modulului de evaluare cu aplicația ce se va evalua. Acesta conține două puncte de tăietură abstracte: **`entryPoint`**() care selectează începutul execuţiei aplicației (de exemplu funcţia main din Java) şi **`exitPoint`**() care selectează sfârşitul execuției aplicației sau închiderea aplicației. Pentru fiecare aplicație care urmează a fi evaluată se va defini un nou aspect (cum ar fi, **`ApplicationIntegrationAspect`**) ce

moşteneşte din **`IntegrationAspect`** şi defineşte cele două puncte de tăietură.

In timpul unei sesiuni de evaluare nu sunt executate toate sarcinile odată. Din aceasta cauză ar fi recomandată încărcarea la cerere a sarcinilor ce vor fi evaluate. Folosind soluția bazată pe aspecte acest lucru nu este posibil, deoarece modificarea aplicației ce se evaluează (folosind weaverul) se poate face doar în timpul compilării, dupa compilare, sau în timpul încărcării claselor (in funcție de limbajul orientat pe aspecte folosit). Din aceasta cauză, pentru a permite selectarea sarcinilor ce se doresc a fi evaluate, fiecare sarcină se înregistrează la **`EvaluationManager`**, iar acesta, prin intermediul interfeței grafice, permite alegerea lor. După alegerea sarcinilor, managerul setează (marchează) pentru fiecare **`TaskEvaluator`** existent dacă sarcina asociată se va evalua, folosind metoda **`setEvaluated`**(). Dacă un **`TaskEvaluator`** nu a fost marcat ca fiind evaluat, pentru sarcina asociată nu se vor calcula metricile.

Starea unui **`TaskEvaluator`** (de a fi evaluat sau nu) este folosită de **`EvaluationManager`** şi pentru a determina sarcinile care au fost abandonate sau care au eşuat.

CONCLUZII

In acest articol am prezentat o arhitectură bazată pe paradigma orientată pe aspecte care permite evaluarea metricilor de execuție.

Avantajele folosirii acestei paradigme sunt:

- Lipsa instrumentării codului sistemului ce urmează a fi evaluat.
- Posibilitatea reutilizării cadrului pentru diferite sisteme (pentru un sistem nou trebuie doar definite câteva aspecte noi specifice sistemului).
- Obținerea rapidă a valorilor metricilor. După terminarea sesiunii de evaluare de către participant rezultatele vor fi salvate automat în depozitul de date ales.
- Nu mai este necesară folosirea jurnalelor şi parcurgerea lor pentru a obține valorile metricilor.

Există şi câteva dezavantaje în folosirea acestei paradigme pentru dezvoltarea cadrului:

- Pentru a defini aspectele este necesară prezența unui expert în modul de construire a interfeței grafice.
- Nu există extensii orientate pe aspecte pentru orice limbaj de programare. In astfel de situații această soluție nu poate fi folosită.
- Punctele de tăietura ce trebuie particularizate pentru fiecare sistem ce urmează a fi evaluat nu sunt întotdeauna uşor de definit. Abordările similare propuse pană acum se referă doar la interfețele grafice construite folosind limbajul Java şi biblioteca Swing de componente grafice.

Ca şi direcții de cercetare viitoare menționăm:

- Implementarea arhitecturii propuse folosind diferite limbaje orientate pe aspecte (AspectJ, AspectC++, etc).
- Folosirea cadrului propus pentru evaluarea diferitor sisteme.
- Investigarea posibilității de a folosi această paradigmă pentru evaluarea siturilor web.

REFERINȚE

1. AspectC++ Homepage. http://www.aspectc.org/.
2. AspectJ Project. http://eclipse.org/aspectj/.
3. S. Bateman, C. Gutwin, N. Osgood, G. McCalla, (2009), *Interactive usability instrumentation.* In Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems (EICS '09). ACM, New York, NY, USA, 45-54.
4. N. Bevan, (2008). *A framework for selecting the most appropriate usability measures.* In COST 294-MAUSE Workshop: Critiquing Automated Usability Evaluation Methods.
5. A. Dingli, J. Mifsud, (2011) *USEFul: A Framework to Mainstream Web Site Usability through Automated Evaluation* Intl Journal of Human-Computer Interaction 2(1), 10-30, 2011
6. A. Dix, J. E. Finlay, G. D. Abowd, R. Beale, (2004). *Human-Computer Interaction*. Pearson Education.
7. E. Gamma, R. Helm, R.E. Johnson, J. Vlissides. (1994) *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994
8. G.S. Hartman, L. Bass, (2005), *Logging Events Crossing Architectural Boundaries*, Lecture Notes in Computer Science, 2005, 823-834
9. ISO 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) Part 11: Guidance on usability. ISO
10. G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, (1997) *Aspect-Oriented Programming*. In Proceedings European Conference onObject-Oriented Programming, volume 1241, pages 220–242. Springer-Verlag, 1997.
11. M. Kuniavsky, *Observing the user experience.* A practitioner's guide to user research. Elsevier, San Francisco.
12. O. Mishali, S. Katz, (2009), *The HighspectJ framework*. In Proceedings of the 8th workshop on Aspects, components, and patterns for infrastructure software (ACP4IS '09). ACM, New York, NY, USA, 19-24.
13. G.S. Moldovan, A. M.Tarța, (2006) *Developing an Usability Evaluation Module Using AOP.* In International Conference on Computers, Communications & Control, ICCCC, pages 320–325, Felix Spa, România, 2006.
14. J., Nielsen, (1993). *Usability Engineering*. Morgan Kaufmann, Elsevier, San Francisco
15. D.A. Norman, (1988) *The design of every day things*. Reprint: Originally published: The psychology of everyday things: New York: Basic Books (1988).
16. J. Rubin, D.Chisnell, J. Spool. (2008), Handbook of Usability Testing: Howto Plan, Design, and Conduct Effective Tests. O'Reilly Media, 2008
17. J. Sauro, A Practical Guide to Measuring Usability: 72 Answers to the Most Common Questions about Quantifying the Usability of Websites and Software,

18. A. Seffah, M. Donyaee, R. B. Kline, H.K. Padda, (2006), *Usability measurement and metrics: A consolidated model.* Software Quality Journal, 14(2), 159-178, 2006.
19. A. Tahir, R. Ahmad, *An AOP-Based Approach for Collecting Software Maintainability Dynamic Metrics*, Computer Research and Development, International Conference on, pp. 168-172, 2010 Second International Conference on Computer Research and Development, 2010
20. Y. Tao, (2008) *Automated Data Collection for Usability Evaluation in Early Stages of Application Development,* 7th WSEAS Int. Conf. on APPLIED COMPUTER & APPLIED COMPUTATIONAL SCIENCE (ACACOS '08), Hangzhou, China, April 6
21. T. Tullis, W.Albert, (2008). *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
22. A. M. Tarta, & G. S. Moldovan, (2006) Automatic usability evaluation using aop. In IEEE International Conference on Automation, Quality and Testing, Robotics (Vol. 2, pp. 84-89), 2006