Aprecierea fiabilitatii folosind metrici software

Lect. Mihai POPESCU
Academia Tehnica Militara, Bucuresti

IEEE defineste fiabilitatea ca "abilitatea unei componente sau a unui sistem de a îndeplini functiile cerute în conditii date si pentru o perioada specificata de timp" [XXXX90].
Managerii de proiect, aproape în totalitate, identifica fiabilitatea cu corectitudinea, pentru ca ei au în vedere testarea si numarul de "bug-uri" gasite si fixate. În timpul descoperirii si fixarii bug-urilor din procesul de testare se asigura, de fapt, fiabilitatea produsului, testarea în toate fazele ciclului de viata fiind o modalitate recomandata pentru dezvoltarea unui produs robust, de înalta calitate si fiabilitate. Astfel, fiabilitatea codului livrat este direct proportionala cu calitatea tuturor proceselor si produselor dezvoltarii software (cu documentarea cerintelor, cu codul, planurile de test si testarea). În acest context, metricile software se utilizeaza pentru a îmbunatati fiabilitatea si calitatea produselor software. Fiabilitatea este un atribut al calitatii si calitatea produselor software poate fi masurata.
Cuvinte cheie: *metrica, software, fiabilitate, management.*

IEEE 982.1-1988 [XXXX88] defineste managementul fiabilitatii software ca fiind "procesul optimizarii fiabilitatii software printr-un program care accentueaza pe prevenirea, detectarea si eliminarea erorilor software si folosirea metricilor pentru a maximiza fiabilitatea în lumina constrângerilor proiect, cum ar fi resursele, planificarea, costul si performantele".
Pornind de la aceasta definitie, fiabilitatea software cuprinde trei activitati: prevenirea erorilor, detectarea si eliminarea erorilor, masuratori pentru a creste fiabilitatea, definirea si utilizarea de metrici specifici care sa sustina primele doua activitati.
În trecut s-a investit o munca laborioasa în masurarea fiabilitatii folosind timpul mediu între incidente si timpul mediu pâna la defectare [MUSA90]. Modelarea s-a folosit pentru a prezice ratele de defecte si fiabilitatea [MUSA90] [TRIA95] [WATE94]. Aceste activitati se refera la punctele 2 si 3 ale fiabilitatii (detectarea si masurarea), identificarea si eliminarea defectelor, astfel ca softul sa functioneze asa cum s-au formulat cerintele de fiabilitate.
Rata de defectare software este mai mare în timpul fazelor de integrare si testare. Pe masura ce software-ul se testeaza, erorile sunt identificate si eliminate. Eliminarea continua cu o rata mica pâna în faza operationala. Numarul de erori scade continuu, presupunând ca nu se introduc altele.
Software-ul nu are componente în miscare si nu se defecteaza ca hardware-ul, dar poate deveni nefolositor si învechit la un moment dat.
O importanta deosebita trebuie acordata prevenirii erorilor, pentru aceasta fiind necesar ca:
1. sa se defineasca cerintele, produsul dezvoltat având toate cerintele clar si precis specificate vizavi de functionalitatea finala;
2. sa se asigure faptul ca dezvoltarea de cod poate fi sustinuta de ingineria software (prin instrumente adecvate), fara a se introduce erori suplimentare;
3. sa se dezvolte un program de testare care sa verifice toate functiile specificate în cerinte.

2. Programul de construire a metricilor de fiabilitate software

Pentru a fi efectiva, masurarea trebuie sa fie o parte integrala a activitatii de management. Masurarea ajuta în atingerea obiectivelor de baza ale managementului

referitor la predictie, progres si îmbunatatirea proceselor.
De Marco [DEMA86] afirma ca "nu poti conduce ceea ce nu poti masura".
În contextul masurarii software sunt trei clase de entitati:

- *Procesele*, reprezentate prin orice activitate specifica, set de activitati sau timp în fabricarea unui produs sau dezvoltarea unui proiect. Exemple în acest sens sunt: specificarea cerintelor, proiectarea, codificarea si verificarea sau o anumita perioada de timp specifica, cum ar fi "primele trei luni ale proiectului X".
- *Produsele*, reprezentate prin orice rezultat al unei activitati sau document rezultat dintr-un proces. Exemple ar putea fi: codul sursa, o specificatie de proiectare, un plan de test si un manual de utilizare produs software.
- *Resursele*, reprezentate prin orice intrare într-un proces. Exemple: o persoana sau o echipa, un compilator sau un instrument de testare software.

Odata ce proiectul software avanseaza si echipa de dezvoltare proiecteaza module si scrie cod, efortul planificare si specificare a metricilor de fiabilitate (colectare date, analiza si înregistrare) da rezultate.
Rezultatele de baza pentru un plan de folosire a metricilor de fiabilitate sunt:

- metricii trebuie sa fie usor de înteles. De exemplu, liniile de cod (LOC, KLOC) si punctele functionale sunt cele mai obisnuite masuri ale dimensiunii software (potential si ale complexitatii), fiind familiare inginerilor software;
- metricii trebuie sa fie ieftini pentru a putea fi cumparati; studiile arata ca aproximativ 5% - 10% din costurile totale de dezvoltare se pot datora metricilor;
- metricii trebuie sa fie testati înainte de a se utiliza; pot avea o baza teoretica solida, dar nu au aplicare sau evaluare practica;
- metricii trebuie sa fie un ajutor managerial pentru îmbunatatirea proceselor;
- metricii sa fie disponibili la timp; daca un metric nu e disponibil decât atunci când sunt probleme, de exemplu daca rata de defectare n-a fost masurata timpuriu, poate fi nefolositor mai târziu [YOUR92];
- metricii trebuie sa stimuleze îmbunatatirile de proces; nu se vor utiliza metrici pentru a judeca echipa sau performanta individuala;
- metricii trebuie sa fie folositori la mai multe niveluri: sa serveasca atât managementului cât si echipei tehnice pentru îmbunatatirea proceselor de dezvoltare.

3. Analiza datelor

Metricii de fiabilitate folosesc date obiective si subiective:
- *datele obiective* sunt contoare ale unor caracteristici (KLOC, puncte functionale, componente, functii testate, unitati codificate, modificari, erori) ce se pot verifica independent;
- *datele subiective* sunt aprecieri individuale ale unei caracteristici sau conditii (nivelul dificultatii problemei, gradul noii tehnologii implicate, stabilitatea cerintelor). Ele aduc informatie critica în interpretarea si validarea datelor obiective.

Pe masura ce se utilizeaza metricii si datele disponibile se înmultesc, se creeaza baze de date cu informatii mai precise si mai utile.
Folosind aceste baze de date se poate vedea daca masuratorile au tendinte diferite fata de proiecte similare anteriore si fata de performanta optima a programului data de modelele implementate în software.
Bazele de date create contin trei tipuri de date:
- *date de cost*, care reflecta eforturile facute;
- *date despre procese*, care reflecta informatia despre programe (metodologie, instrumente si tehnici utilizate) si experienta/instruirea personalului;
- *date despre produs*, includ dimensiunea, date despre modificari, defecte si rezultatele analizelor statistice asupra codului livrat.

Figura 1 ilustreaza posibilitatea comparării, bazata pe metrici, cu proiecte aflate în baza de date istorica.
Comparatiile sunt o componenta a sistemului de feedback si control, care permite revizii în planul de management, fapt ce va produce îmbunatatiri în procesul de dezvoltare care, la rândul lor, vor duce la masurari ajustate în pasul urmator al comparatiei.

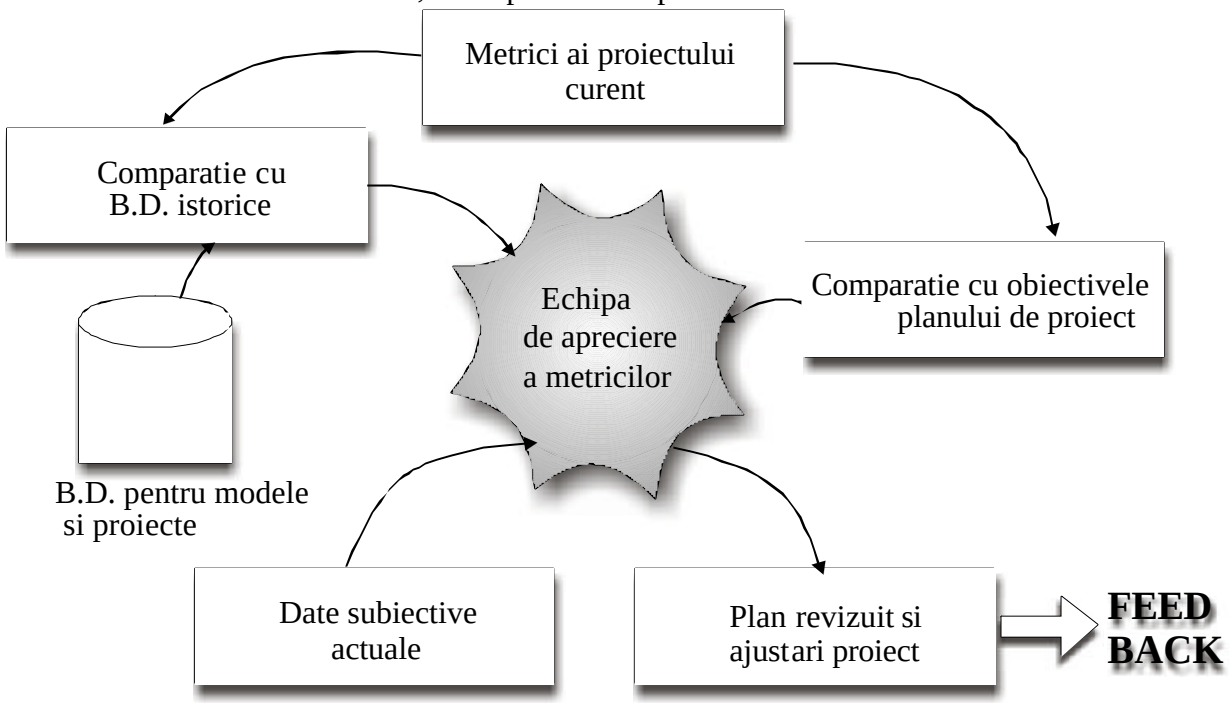

Fig. 1. Procesul de management utilizând metrici

4. Identificarea scopurilor metricilor de fiabilitate

Un program de metrici e o activitate planificata a utilizarii masuratorilor pentru a vedea daca s-a atins un anumit obiectiv.
Programul de elaborare metrici este o succesiune de activitati:

Scopuri Þ **Metrici** Þ **Date** Þ **Realitati/tendinte** Þ **Decizii** Þ **Actiuni**

Deoarece exista o multitudine de scopuri tehnice care pot fi sustinute de metricii de fiabilitate, nu exista un program de metrici universal, care sa fie adoptat de toate companiile software. Metricii de fiabilitate software graviteaza în jurul procesului de testare. Multe programe de metrici esueaza pentru ca nu sunt definite corespunzator sau din lipsa obiectivelor.
Pentru a rezolva acest neajuns, Vic Basili si echipa sa de la Universitatea Maryland au dezvoltat o conceptie riguroasa de masurare orientata pe scop, numita GQM (***G**oal **Q**uestion **M**etrics*). Ideea de baza este aceea ca managerii urmeaza urmatoarele trei etape (figura 2):
1. stabilesc obiective specifice necesitatilor în termeni ai scopului, perspectivei si mediului;
2. rafineaza scopurile în întrebari cuantificabile care sunt usor de manevrat;
3. deduc metricii si datele ce trebuie colectate pentru a raspunde la întrebarile stabilite.

În practica este imposibil de folosit toti metricii care rezulta dintr-o strategie de nivel înalt. De exemplu, se stie ca *dimensiunea* (în termenii functionalitatii) si *complexitatea* (în termenii dificultatii) unui modul sunt factori certi ai numarului de defecte care vor fi depistate.
Totusi, poate fi imposibil de dat masuri precise ale functionalitatii si complexitatii (unii pot fi multumiti cu numarul de instructiuni KLOC si numarul de ramuri de

cod). Asemanator, se stie ca folosirea operationala a unui modul este un factor esential în aparitia incidentelor. Totusi, se poate întâmpla sa nu se stie aceste date, fiind sarcina celui care implementeaza un astfel de plan s-o faca într-un mod realist.

Scop:	Identificarea în avans a modulelor pasibile la defectari		
Întrebari:	Sunt defectele descoperite anterior livrarii predictive pentru incidentele ulterioare ?	Care factori ai procesului de baza pot influenta defectele si incidentele ?	Care factori ai produsului de baza pot influenta defectele si incidentele ?
Metrici:	Datele de defectare pentru fiecare modul: • numarul de defecte depistate în faza de testare; • numarul de incidente aparute în modul si severitatea lor.	Date despre efortul depus pentru fiecare modul: • durata de testare din faza de testare; • efortul de dezvoltare • experienta programatorilor; • folosirea operationala actuala.	Date despre dimensiunea/complexitatea fiecarui modul: • KLOC sau puncte functionale; • metrici de complexitate

Fig. 2. Modelul "Scop - Întrebari - Metrici"

Un asemenea document ar putea contine urmatoarele informatii de baza:

- justificarea faptului ca metricii propusi corespund scopului tehnic;
- ce metrici se vor colecta, cum vor fi definiti si cum vor fi analizati;
- cine va face colectarea de date, cine le va analiza si cine va vedea rezultatele;
- cum vor fi obtinuti: instrumente, tehnici si practici utilizate pentru a ajuta colectarea datelor metricilor si analiza acestora;
- când si cum cor fi colectati si analizati metricii în proces;
- unde se vor memora datele.

În cazul companiei Alpha, la aceste puncte s-ar putea raspunde astfel:

De ce? Prin colectarea datelor despre defecte se vor depista modulele cu un numar disproportionat de defecte anterior livrarii si care, la rândul lor, vor cauza un numar disproportionat de incidente în faza operationala. Transpunând acest lucru în metrici care ofera date explicarii posibile (cum ar fi efortul de testare si dimensiunea modulului) vom putea identifica modulele problema pre si post livrare.

Care metrici? Pentru fiecare modul se vor colecta metricii din figura 3 (modificati în raport cu realitatea practica); de asemenea se vor calcula metrici derivati: defecte/ KLOC (densitatea de defecte); incidente/zi de folosire operationala; defecte depistate într-o ora de testare.

Metricii se vor colecta pentru proiecte în dezvoltare (de exemplu x si y) si pentru proiecte abia livrate (de exemplu z).

Cine? Testerii vor raspunde de înregistrarea informatiilor despre defectele gasite la prelivrare.

O persoana desemnata se va ocupa, în cadrul suportului tehnic acordat, de înregistrarea incidentelor operationale raportate de utilizatori.

Managerul de proiect va raspunde de colectarea metricilor specifici unui modul, cum ar fi *metricii de proces* (efortul de testare) si *metricii de produs* (dimensiunea).

Cum? Informatia despre defecte si incidente se va înregistra într-o baza de date relationala (figura 4). Metricii de proces (cum ar fi efortul de dezvoltare si testare pe modul) se vor baza pe foi de raportari în timp standard.
Când? Pentru proiectele x si y metricii pre livrare vor fi adunati si raportati formal la sfârsitul fiecarei etape de testare majora. Pentru proiectul z, pe cât posibil, informatia despre metricii pre livrare se va acoperi retrospectiv din înregistrarile de proiect.
Pentru toate proiectele, metricii operationali vor fi strânsi si raportati formal la anumite intervale (de exemplu, 6 luni).
Und? Datele se vor memora într-o baza de date accesibila angajatilor companiei pe o retea intranet.

5. Structura bazei de date

Multe companii fac greseala de a trata defectele si incidentele la fel ca în sistemul de raportare, completând acelasi gen de informatii despre amândoua.
Propun o baza de date relationala care contine informatii despre urmatoarele caracteristici: incidente (caderi); defecte; cereri de schimbare (modificare) de cod.
Utilizatorii raporteaza incidentele când sistemul e operational. Defectele sunt gasite de testeri înainte de livrare sau atunci când se depisteaza cauza unui incident raportat. Cererile de schimbare (corectiva sau o noua functie) pot fi facute de utilizatori sau dezvoltatori. Incidentele au atribute precum:

- *localizare*: calculatorul pe care s-a observat;
- *timpul*: calendaristic, sau de executie;
- *efectul*: ce s-a observat la aparitia lor;
- *mecanismul*: ce secventa de actiuni/intrari conduce la incident;
- *cauza*: defectul care a cauzat incidentul; va fi necunoscut la prima raportare.

Defectele au atribute precum localizare si timp, dar localizarea se refera la modulul unde s-a produs, iar timpul la faza de testare.
Toate caracteristicile (incidente, defecte, modificari) trebuie sa aiba un atribut de criticalitate si altul de severitate. Criticalitatea se refera la importanta caracteristicii, iar severitatea mai mult la amploare. De exemplu, criticalitatea unei defectari poate fi majora daca întrerupe functionarea întregului sistem, în timp ce severitatea este majora daca poate afecta foarte multi utilizatori.
Astfel, se poate construi o baza de date relationala (figura 3), unde incidentele se relationeaza cu defectele care le-au cauzat si modificarile se relationeaza cu defectele pe care le fixeaza sau cu cererile de modificare.
Baza de date are doua entitati de baza carora li se pot asocia doua formuri (ferestre pentru introducerea de date):

- *Caracteristici* care contine atribute dependente de tipul caracteristicii (incident, defect sau cerere de modificare).
- *Formul modificari* care detaliaza modificarile care se impun ca raspuns la caracteristici.

Pentru ca procesul de colectare a datelor sa functioneze bine, trebuie sa existe un comitet al managerului de proiect care sa fie plasat unde se raporteaza si se înregistreaza toate caracteristicile. De asemenea, trebuie sa existe o persoana desemnata pentru filtrarea intrarilor pentru a ne asigura ca se introduc în baza de date numai caracteristici noi. Altfel, pot aparea numarari multiple ale acelorasi incidente si defecte, repetând inutil o munca (o serie de activitati).
Baza de date ne ajuta sa obtinem date ca în tabelul 1.

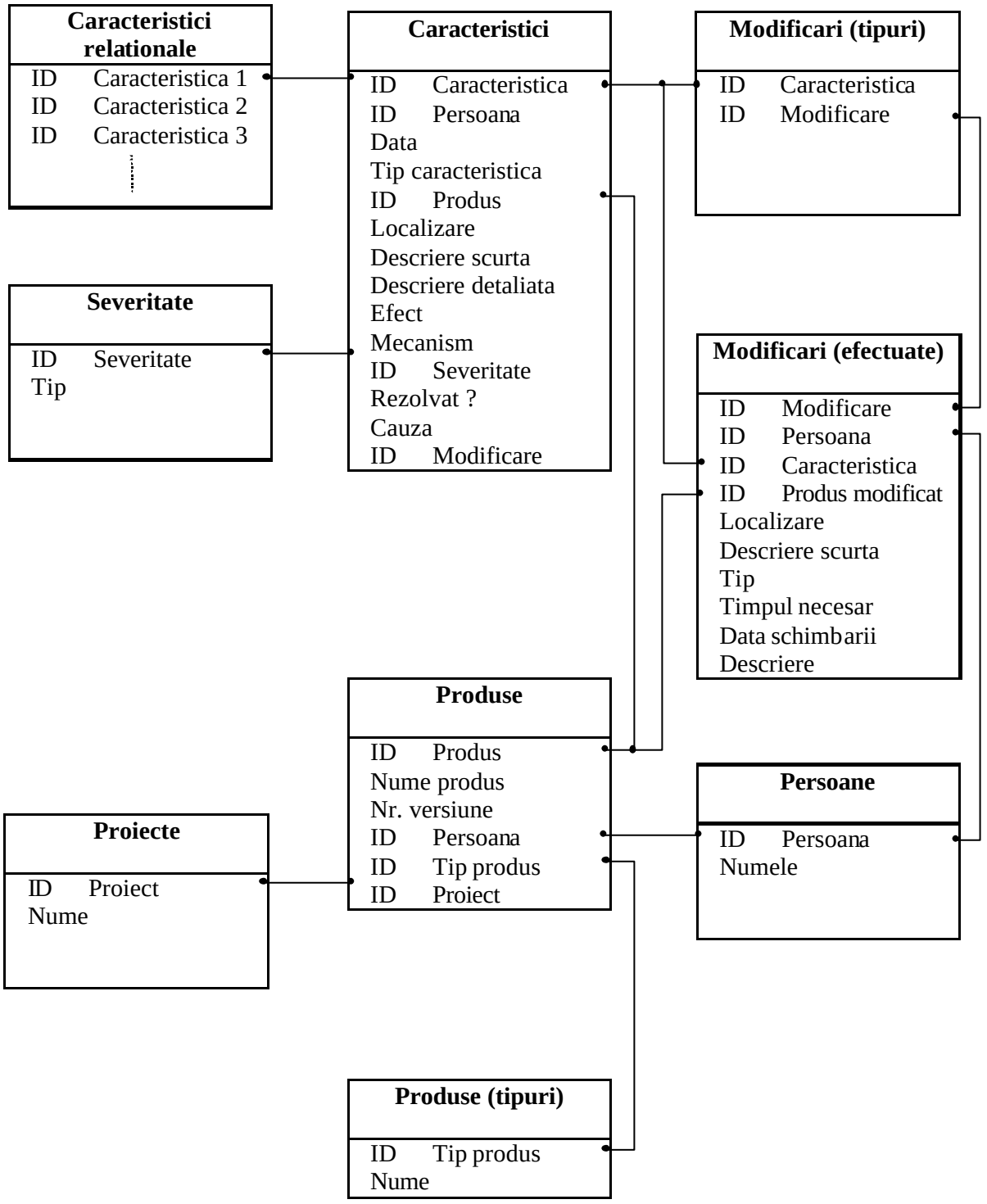

Fig. 3. Baza de date relationata folosita pentru colectarea datelor despre defecte, incidente si modificari

Tabelul 1

Modul (subprogram compilat separat)	Defecte totale (înainte de livrare si 6 luni dupa livrare)	Dimensiune (KLOC)	Defecte pre livrare	Defecte post livrare	Densitate de defecte post livrare (defecte/KLOC)
M1	16	1	16	0	0
M2	4	2	1	3	1,5
M3	16	3	11	5	1,7
M4	11	2	0	11	5,5
M5	50	6	15	35	5,8
M6	16	4	0	16	4

Aceste date ne permit sa observam factori si tendinte precum:

– densitatea de defecte pre-livrare este un predictor necorespunzator densitatii de defecte post-livrare;

– modulele cu o densitate de defecte pre-livrare scazuta pot sa aiba multe probleme dupa livrare.

Totusi, în planul de metrici pot exista factori predictivi extrasi din baza de date. De exemplu, se poate aplica un filtru pentru a restrictiona densitatea de defecte doar pentru cele mai critice defecte, pentru a vedea daca se schimba cele afirmate.

Pentru modulele cu densitate de defecte pre-livrare foarte mica se poate verifica efortul de testare si se compara cu cel mai mic, mediu si cel mai mare efort de testare pentru toate modulele. Astfel s-ar putea gasi ca modulele M4 si M6, care au înregistrat zero defecte pre-livrare, s-au testat insuficient. Astfel de informatie poate duce la decizii de genul: modulele M4 si M6 necesita testare suplimentara sau modulele M4 si M6 trebuie reproiectate complet înainte de livrare.

Aceste decizii pot conduce la actiuni asociate, cum ar fi utilizarea de proceduri noi pentru asigurarea calitatii (de exemplu, impunerea unor niveluri minime de testare).

În concluzie putem evidentia urmatoarele aspecte:

– factorii cauzali si explicatorii sunt esentiali în utilizarea metricilor ca un instrument de management real în reducerea riscului;

– densitatea de defecte este nerelevanta daca stim ca un numar de module n-au fost testate;

– esentiala în folosirea metricilor nu este taria acestora, ci combinarea lor în vederea obtinerii unei predictii reale prin tehnici statistice avansate - retelele bayesiene sunt un exemple de îmbunatatire a predictiilor fiabilitatii software, luând în calcul factorii cauzali si o strategie de reducere a riscului adecvata.

6. Capabilitati si limitari ale metricilor de baza în fiabilitatea software

Ne vom îndrepta atentia asupra metricilor de defecte, dimensiune si complexitate. Trebuie facuta distinctie între diferite concepte:

Defecte descoperite în diferite faze ale ciclului de viata:

- defecte depistate în faza operationala (numite incidente sau caderi);
- defecte descoperite în timpul dezvoltarii, referite ca defecte pentru ca pot conduce la incidente în operare.

Criticalitatea diferitelor categorii de defecte.

Unul dintre obiectivele majore ale metricilor software este acela de a da predictii timpurii bune ale fiabilitatii (frecventa defectarilor operationale).

Se stie ca modelele de crestere a fiabilitatii stocastice pot da predictii precise ale fia-

bilitatii unui sistem software, daca se colecteaza suficiente date despre defectari într-un mediu operational reprezentativ [BROC92]. De multe ori, însa, facem predictii înainte ca softul sa fie operational. În [FENT98] se face un studiu de caz pornind de la doua versiuni de software de dimensiuni mari pentru a testa o serie de ipoteze despre metricii de baza folositi în estimarea fiabilitatii. Acest studiu l-am completat cu informatii si observatii obtinute din activitatea de depanare si proiectare software pe care am desfasurat-o ca inginer de sistem în Academia Tehnica Militara în perioada 1983-1995.

Sistemul s-a divizat în sute de module, iar datele metricilor s-au colectat la nivel de modul: numarul de defecte gasite la patru faze de testare diferite (modul, de integrare, sistem si în faza operationala); LOC (*Lines Of Code)*; date de complexitate ce-au inclus si complexitatea ciclomatica.

Ipotezele si rezultatele obtinute se prezinta în tabelul 2.

Tabelul 2

Nr. crt.	Ipoteza	Rezultate obtinute
1.1	Un numar mic de module contin majoritatea defectelor descoperite în testarea pre-livrare	Da, s-a confirmat
1.2	Daca un numar de module contine majoritatea defectelor descoperite în timpul testarii pre-livrare, atunci ele dau codul majoritar.	Nu s-a confirmat
2.1	Un numar mic de module contine majoritatea defectelor operationale.	Da, e adevarat
2.2	Daca un numar mic de module da majoritatea defectelor operationale, ele constituie majoritatea codului.	Nu, infirmat puternic
3	Modulele cu un numar mare de defecte descoperite anterior livrarii este probabil sa se comporte la fel si în faza operationala.	Nu, respinsa puternic
4.1	Modulele mai mici sunt mai putin predispuse la defectari decât cele mari.	Nu s-a confirmat
4.2	Metricii de dimensiune sunt buni predictori ai densitatii de defecte pre si post livrare într-un modul.	Nu s-a confirmat
5	Metricii de complexitate sunt predictori mai buni decât cei de dimensiune simpli ai modulelor pasibile la defecte si incidente.	Slab confirmata
6	Densitatile de defecte, la faze analoge de testare si operationale, ramân în general constante între versiuni majore ulterioare ale unui sistem software.	Da
7	Sistemele software produse în medii similare au densitati de defecte apropiate în faze de testare si operationale similare.	Da

7. Concluzii

- Dimensiunea nu explica în orice situatie numarul mare de defecte; multi înclina sa creada (ipotezele 1.2 si 2.2) ca daca un numar mic de module contine majoritatea defectelor, se explica prin faptul ca sunt mari si dau "grosul" dimensiunii sistemului, lucru neconfirmat întotdeauna.
- Complexitatea nu este semnificativ mai buna în prezicerea modulelor pasibile la defecte si defectari decât metricii de dimensiuni simpli (ipoteza 5).
- Nu este obligatoriu ca modulele care contin multe defecte înainte de livrare sa fie aceleasi si în faza operationala (ipoteza 3). Aceasta ipoteza nu numai ca s-a infirmat, dar a evidentiat ca în ambele versiuni defecte numeroase descoperite în testarea pre-livrare au aparut în module care ulterior n-au mai avut probleme în faza operationala.

- Raportat la densitatea de defecte, putem asertiona ca la nivel de modul o valoare mare în faza operationala este mai degraba un indicator al testarii insuficiente decât un nivel de calitate scazut (modulele au densitatile de defecte pre si post-livrare invers proportionale);
- Se confirma faptul ca metricii de complexitate sunt strâns legati de metricii de dimensiune precum LOC, fiind predictori rezonabili ai numarului absolut, dar modesti pentru densitatea de defecte;
- Pentru a face o predictie adecvata, trebuie înteleasa bine relatia cauza si efect. În acest context, relatia dintre dimensiunea unui modul si numarul de defecte gasite nu este o evidenta a relatiei cauzale. În schimb, între profunzimea testarii si nivelul de fiabilitate atins exista o asemenea relatie. Ipoteza 4.1 este infirmata si de studii mai recente [HATT97], unde se afirma ca modulele mari au o densitate de defecte mai mica decât modulele mici, fiind mai putin criptice si complicate.
- Foarte importante sunt efortul de testare si folosirea operationala; daca nu se testeaza sau nu se utilizeaza un modul, nu se pot observa defectele sau incidentele asociate lui.

Este nevoie de modele de fiabilitate care sa poata manevra: procese si produse diverse; relatii cauza si efect adevarate; incertitudinea; informatia incompleta.

Modelele nu trebuie sa introduca metrici suplimentari, sofisticati si care necesita date numeroase de colectat. În acest sens, în [FENT98] se apreciaza ca retelele Bayesiane sunt cea mai buna solutie pentru problemele care au un suport decizional incert, fiind sprijinite de instrumente software care le-au implementat. Un astfel de instrument este HUGIN care a fost adus de autor via Internet, fiind instalat si utilizat pe calculator. Cea mai recenta folosire a fost de catre Microsoft în wizardurile de help din Microsoft Office, dar si în aplicatii care necesita luarea deciziei într-un timp critic pentru sistemele de propulsie în spatiul aerian [HORV95].

Bibliografie

[AKIY71] Akiyama, F., "An Example of Software Design Debugging", Inf Processing 71", pg. 353-379, 1971.

[ALBR79] Albrecht, A., J., "Measuring Application Development", Proceedings of IBM Applications Development Joint SHARE/GUIDE Symposium, Monterey CA, pg. 83-92, 1979.

[BOEH81] Boehm, B., W., "Software Engineering Economics", Prentice-Hall, New York, 1981.

[BROC92] Brocklehurst, S., Littlewood, B., "New ways to get accurate software reliability modeling", IEEE Softwre, pg. 34-42, July 1992.

[DEMA86] De Marco, T.,"Controlling Software Projects", Yourdon Press, New York, 1986.

[FENT98] Fenton, N., E., Littlewood, B., s.a., "Assesing Dependability of Safety Critical Systems Using Diverse Evidence", IEE Proceedings Software Engineering, 145(1), pg. 35-39, 1998.

[GILB76] Gilb, T., "Software Metrics", Chartwell-Bratt, 1976.

[GRAD92] Grady, R., B., "Work-Product Analysis:The Philosopher's Stone of Software?", IEEE Software, pg. 26-34, March 1992.

[HALS77] Halstead, M., H., "Elements of Software Science", Elsevier-North Holland, Amsterdam, 1977.

[HATT97] Hatton, L., "Reexamining the fault density-component size connection", IEEE Software, 14(2), pg. 89-97, 1997.

[HORV95] Horvitz, E., Barry, M., "Display of information for time-critical decision making", Proc of 11th Conf in AI, Montreal, August 1995.

[IVAN96] Ivan, I., Popescu, M., "Metrici software", Revista Byte, vol. 2, nr. 5, mai, 1996, pag. 73 – 82.

[MCAB76] McCabe, T., J., "A Complexity Measure", IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pg. 308-320, December 1976.

[MUSA90]Musa, J., D., Okumoto, K., "Software Reliability: Measurement, Prediction, Application", Professional Edition:Software Engineering Series, McGraw-Hill, New York, 1990.

[ROSE98] Rosenberg, L., Hammer, T., "Metrics for Quality Assurance and Risk Assessment", Proc. Eleventh International Software Quality Week, San Francisco, 1998.

[SYMO91] Symons, C., R., "Software Sizing & Estimating:Mark II Function Point Analysis", John Willey & Sons, 1991.

[TRIA95] Triantafyllos, G., Vassiliadis, S., Kobrosly, W., "On the Prediction of Computer Implementation Faults Via Static Error Prediction Models", Journal of Systems and Software, vol. 28, no. 2, pg. 129-142, February 1995.

[WATE94] Waterman, R., E., Hyatt, L., E., "Testing-When Do I Stop?", International Testing and Evaluation Conference, Washington, DC, October 1994.

[YOUR92] Yourdon, E., "Decline and Fall of the American Programmer", Yourdon Press, Englewood Cliffs, New Jersey, 1992.

[XXXX88] IEEE Std. 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software.

[XXXX90] IEEE Standard 610.12-1990, Glossary of Software Engineering Terminology.