INDEX

Table of Contents

1.0 INTRODUCERE

Programele de control al software-ului sunt de real interes in DoD si in practica industriala. Aceste programe rulează pe idei de scop si rezultat. Acestea suporta programe de implemetarea si managementul proceselor imbunatatite, cum ar fi cele bazate pe Software Engineering Institute's Capability Maturity Model (CMM) si pe NASA Goddard Space Flight Center Software Engineering Laboratory's Process Improvement Paradigm (PIP), dar si ofera support pentru managementul individual al proiectelor.

Metrica, totuşi, e un termen folosit vag. In unele cazuri, este folosit pentru a descrie măsurători specifice, cum ar fi complexitatea codului implementat. In alte cazuri, este folosit pentru a oferi indicatori de tendinţa, cunoscuţi sub numele de indicatori de management, cum ar fi documentaţia terminata după un anumit program. Oricare ar fi aplicaţia, puţine criterii acceptate ca fiind măsurabile sunt cunoscute si documentate pentru implementări specifice de software.

Scopul acestui raport este de a oferi informaţii de baza despre un set selectat de metrici, specific eficacităţii, complexităţii si fiabilităţii. Nu este o tratare comprehensiva a

metricii; intr-adevăr, acest subiect este tratat intr-un număr de carti si iniţiative DoD. Unele dintre aceste iniţiative includ programul Joint Logistics Commanders' Practical Software Measurement (PSM), programul SEI's Software Engineering Measurement and Analysis (SEMA) si metrica U.S. Army's Software Test and Evaluation Panel (STEP). Aceste iniţiative, oferă metodologii de implementare a metricii bazate pe concepte de indicatori de management. Toţi indicatorii de management se bazează pe o colecţie de date elementare pe care le numim, in acest raport, metrici.

Scopul unui program larg de măsurări este de a colecta date pentru a estima progresul unui proiect sau acceptabilitatea unui proiect. Intr-un proiect, obiectivul este de a estima procesul de dezvoltare si realizare a produselor proiectului pentru a evalua capacitatea proiectului de a-si îndeplini obiectivul. Intr-o organizaţie, programul de măsurători suporta aprecierea procesului de dezvoltare si întreţinere cu scopul de a imbunatati acest proces ca parte a unui program de imbunataţire a proceselor.

Când se planifica sau se implementează un proiect, se ridica întrebarea referitoare la care este valoarea acceptata pentru o anumita metrica. Un manager va trebui sa ştie care sa fie valorile aşteptate pentru o anumita complexitate a design-ului sau a codului, sau eficienta aşteptata pentru un ciclu de viata al produsului. In acest raport, prezentam metricile intr-un mod arborescent menţionat mai sus, eficienta, fiabilitatea si complexitatea. Explicam definiţiile lor si dam o serie de valori care trebuie sa ne aşteptam, bazate pe practicile curente. Exemplele prezentate sunt demonstrative pentru metricile acceptate in fiecare domeniu, dar nu sunt singurele pentru fiecare domeniu. In mod promiţător, pe măsura ce date adiţionale sunt colectate si făcute publice, acest raport poate fi lărgit pentru a forma o baza de norme pentru măsurători.

2.0 SPECIFICATII DE BAZA

Aceasta secţiune sumarizeaza câteva metrici de baza in software, bazate in principal pe literatura de inginerie software. Utilizatorii acestor rezultate trebuie sa fie conştienţi de existenta programelor de control al software-ului. Programele recente n-au produs inca date ce pot fi sumarizate in acest raport. Un nou val a apărut in controlul software-ului o data cu programele comerciale sau guvernamentale de control al software-ului ce au fost create recent. Programele influenţate de guvern includ:

- National Aeronautics and Space Administration Software Engineering Laboratory (NASA/SEL), un model timpuriu pentru o organizatie de control a software-ului;
- National Software Data and Information Repository (NSDIR), initiata de Mr. Lloyd Mosemann, II, Deputy Assistant Secretary of the Air Force, Communications, Computers, & Support Systems (Chruscicki 95)
- *Practical Software Measurement: A Guide to Objective Program Insight* (Version 2.1, 27 March 1996), sponsorizata de Joint Logistics Commanders, Joint Group on Systems Engineering
- Software Engineering Institute (SEI) Capability Maturity Model (CMM), care necesita control software la un nivel inalt (Paulk 95). Mai mult, SEI incepe un antrepozit metric si aduna date pe costul si beneficiile oferite de CMM.

- U. S. Air Force *Software Metrics Policy*, Acquisition Policy 93M-017, 16 February 1994
- U. S. Army Software Test and Evaluation Panel (STEP) metrics(DA 92)

2.1 EFICIENTA

Metrica: Eficienta
Măsurători: Ieşirea produsa pentru o unitate de intrare intr-o organizaţie software.
Metrici asociate: Mărimea ca fiind măsurata de funcţii calitate (FPs) sau Liniile de cod (SLOC), efort, plan.
Aplicaţii : Evaluarea eficientei unui proiect, noi tehnologii, etc. ; măsura estimata, efort, cost, si un orar in planificarea proiectului.
Definiţie: FPs pe persoana luna sau SLOC pe persoana luna. FP-urile sunt o suma ponderata a numărului de intrări, ieşiri, fişiere si interfeţe externe ale programului.

Gama:

- **Eficienta:** De la 2 la 23 FPs de persoana luna cu o medie de 5.6 FPs de persoana luna, sau de 80-400 SLOC de persoana luna.
- **Măsura:** De la 100 la 2300 FPs cu o medie de 993 FPs, sau de la 2 pana la 512 KSLOC
- **Conversie funcţii calitate :** De la 6 la 320 SLOC pe FP.

Note : Funcţiile calitate sunt aplicabile la Sistemele de management al informaţiei (MIS) si la alte aplicaţii in domeniul afacerilor.

Pentru mai multe informaţii:

International Function Points User Group (IFPUG)
5009-28 Pine Creek Drive
Westerville, OH 43081-4899
Voice: (614) 895-7130; FAX: (614) 895-3466
E-mail: 102214.2013@compuserve.com
URL: http://www.ifpug.org/ifpug

Barry W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

Capers Jones, *Programming Productivity*, McGraw-Hill, 1986.

Eficienta, ieşirea produsa pentru o unitate de intrare, este o caracteristica cruciala pentru organizaţiile software. Managementul nu poate estima valoarea eforturilor de imbunatatire a proceselor si impactul noilor tehnologii si metode fara un mecanism pentru masurarea eficientei, ca si a calităţii. Mărimea sistemului software si a eficientei sunt adesea

folositoare in planificarea proiectelor. Bugetele si planificările pot fi estimate conform măsurătorilor. Ideal, măsurarea mărimii ar trebui sa fie uşor estimata din fazele de cerinţe ale proiectului software.

Liniile sursa de cod (SLOC) este o metoda timpurie de măsura a mărimii software-ului, si un număr de modele de cost au fost dezvoltate pe acest considerent. Deşi este legata de perspectiva ingineriei software, aceasta măsura are un număr de paradoxuri. Sistemele echivalente funcţional pot sa varieze in SLOC, depinzând de cat de strâns este dezvoltat codul. Totuşi, măsurând productivitatea după SLOC pe persoana luna premiază codarea ineficienta si neglijenta. Expresivitatea codului sursa variază după nivelul limbajului. Software-ul poate cu uşurinţa sa fie codat la un nivel mai înalt cu mai puţin SLOC. Daca este folosit imprudent, SLOC-ul poate arata o eficienta scăzuta când, de fapt, eficienta a crescut (Jones 86). In încheiere, estimările SLOC-lui făcute devreme in timpul ciclului de viata pot expune o mare variabilitate si pot fi neadecvate ca principal conducător la un model de cost al software-ului.

Allan Albrecht (79), in colaborare cu John Gaffney, Jr. (Albrecht 83), au conceput funcţiile calitate ca fiind o măsura directa a functionalitatii capabile sa măsoare date accesibile inca din faza de cerinţe. Utilizarea FP-urilor a devenit mai răspândita, in special intre cei care creează modele de cost pentru software. Fp-urile sunt mai potrivite pentru sistemele informatice, dar au fost propuse modificări pentru a le adapta pe acestea la aplicaţii cu caracter mai general. De exemplu, funcţiile calitate ale lui Capers Jones pentru complexitatea algoritmica tipica sistemelor integrate.

Acest raport prezintă norme inginereşti pentru eficienta software sub doua forme. Secţiunea 2.1.1 prezintă eficienta măsurata cu funcţiile calitate. Secţiunea 2.1.2 utilizează tradiţionalele măsurători SLOC. Un număr de furnizori de modele de cost pentru software si consultanţi in măsurarea eficientei software menţin baze de date proprii. Appendix-ul B conţine informaţii cu cei mai cunoscuţi astfel de furnizori.

2.1.1 Funcţii calitate

Funcţiile calitate (FP) sunt o suma ponderata a numărului de intrări, ieşiri, fişiere si interfeţe ale sistemului. Tabela 2.1-1, adaptata de la (Albrecht 83), prezintă o hârtie de lucru sumarizand algoritmul FP. Cele mai recente reguli de măsurare sunt definite in Release 3.0 (1990) of *Function Point Counting Practices Manual*, by the International Function Points Users Group (IFPUG).

Tabelul 2.1-1: Hartie de lucru a funtiilor calitate

		Factor de ponderare(c_i)			**Functii**
	Numar	**Simplu**	**Mediu**	**Complex**	**Calitate**
Numar de intrari	x_1	3	4	6	$c_1 x_1$
Numar de iesiri	x_2	4	5	7	$c_2 x_2$

Numar de cerinte ale userului	x_3	3	4	6	$c_3 x_3$
Numar de fisiere	x_4	7	10	15	$c_4 x_4$
Numar de interfete externe	x_5	5	7	10	$c_5 x_5$
	Total functii calitate				$\sum_{i=1}^{5} c_i x_i$

Care sunt mărimile tipice ale sistemelor in termeni FP? Doua mici seturi de date sunt din ce in ce mai referite de cercetătorii FP - Allan Albrecht & John Gaffney's (Albrecht 83) si Chris Kemerer's (Kemerer 87). Aceste seturi de date conţin in special aplicaţii din mediul afacerilor scrise in Cobol. Furnizorii de modele de cost si-au dezvoltat propriile baze de date, dar acestea sunt private.

Mărimile sistemelor examinate de Albrecht & Gaffney si Kemerer sunt de la diferite populaţii. Sistemele din datele lui Kemerer, care au fost colectate mai tarziu, tind sa fie mai mari decât cele de la Albrecht & Gaffney. Figura 2.1-1 arata distribuirea FP-urilor in proiectele Cobol ale lui Kemerer. Aceste proiecte sunt sisteme de marime medie, variind de la 39 la 450 mii de linii de cod (KSLOC). Valorile merice ale FP-urilor sunt citite de-a lungul axei X, in timp de inaltimea boxelor reprezintă proporţiile sistemelor cu FP intre intervalele indicate. Acest grafic ajuta sa înţelegem daca un sistem este mare sau mic, bazându-ne pe numărul de FP-uri din sistem. De exemplu, un sistem cu 1000 de FP-uri este de mărime medie.

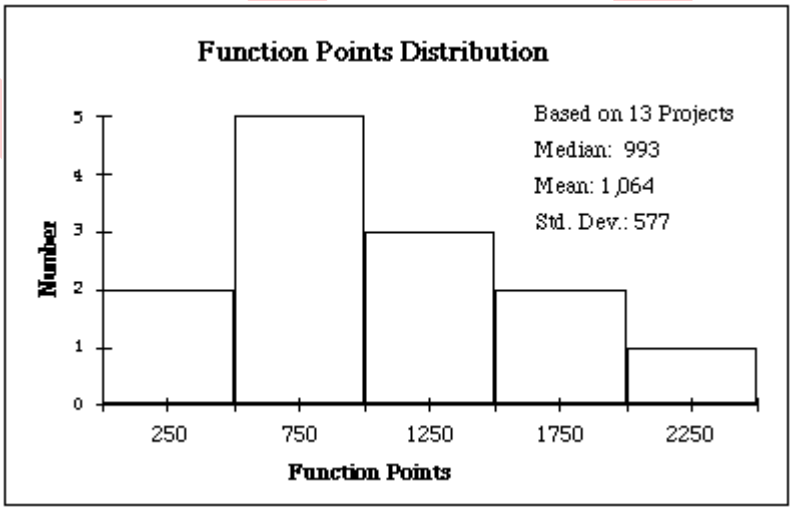

Figura 2.1-1: Datele Cobol ale lui Kemerer

Eficienta este măsurata după FP-uri pe persoana luna. Persoana luna poate fi prezisa din FP. Seturile de date ale lui Albrecht & Gaffney[4] si Kemerer pot fi folosite pentru a construi o relaţie pentru estimarea eforturilor conform FP-ului si pentru a examina distribuţia eficientei. Figura 2.1-2 arata o dependenta lineara pentru estimarea efortului din FP pentru proiectele Cobol. Figura 2.1-3 prezintă o histograma arătând distribuţia eficientei. Cititorii

acestui raport sunt încurajaţi sa dezvolte propriile lor baze de date pornind de la datele colectate in mediul particular.

Poate fi folositor sa ataşam FP-ul la SLOC. Cantitatea de SLOC reprezentata de un singur FP variază după limbaj, cu un nivel înalt necesitând mai puţin SLOC la fiecare FP. Tabela 2.1-2, după (Jones 86) estimează SLOC-ul per FP.

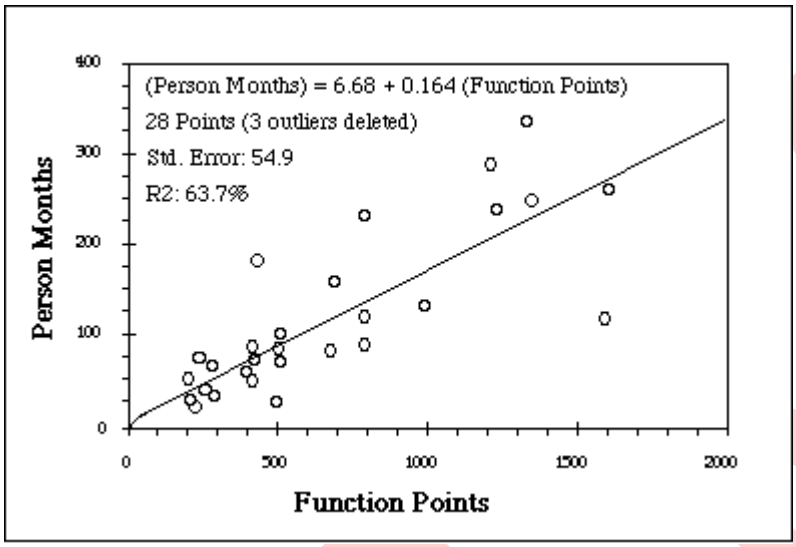

Figura 2.1-2: Efort versus Functii calitate

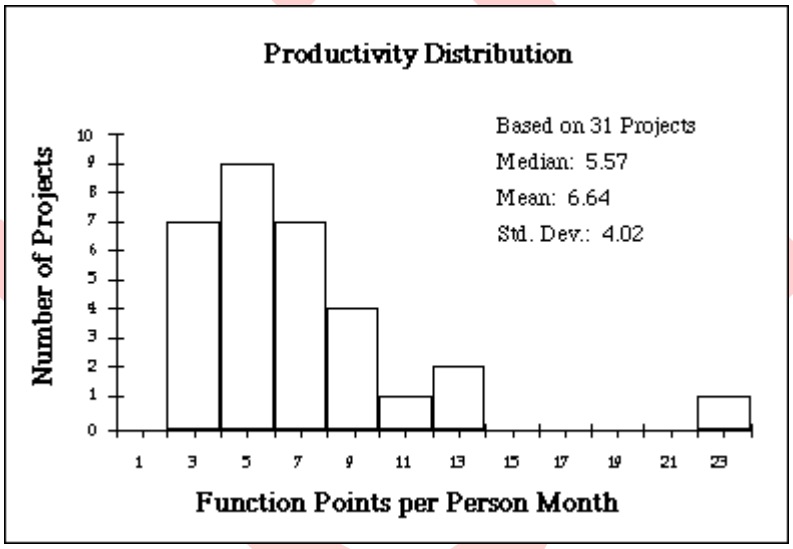

Figura 2.1-3: Eficienta in proiectele Cobol

Tabelul 2.1-2: Sloc Per FP dupa limbaje

Language	SLOC per FP
Assembler	320
Macro Assembler	213
C	150
Algol	106
Chill	106

Cobol	106
Fortran	106
Jovial	106
Pascal	91
RPG	80
PL/I	80
Modula-2	71
Ada	71
Prolog	64
Lisp	64
Forth	64
Basic	64
Logo	53
4th Generation Database	40
Strategem	35
APL	32
Objective - C	26
Smalltalk	21
Query Languages	1
Spreadsheet Languages	6

Liniile de cod sursa

Liniile de cod sursa (SLOC) oferă o baza mai tradiţionala pentru estimarea eficientei. Barry Boehm's Constructive Cost Model (COCOMO) este un model de cost acceptat care încapsulează relaţia dintre eficienta si SLOC (Boehm 81). Tabela 2.1-3 arata un SLOC tipic pentru o gama de mărimi de proiecte. Tendinţa este pentru organizaţii de a atinge sisteme mai mari in timp. Microsoft oferă un exemplu comercial. Microsoft Basic are 4KSLOC in 1975, in timp ce versiunea curenta se apropie de 500KSLOC. Microsoft Word 1.0 are 27 KSLOC; Word este acum de aproximativ 2000 KSLOC (Brand 95).

Tabelul 2.1-3: Game de marimi de baza COCOMO

Size	**KSLOC**
Small	2
Intermediate	8
Medium	32
Large	128
Very Large	512

COCOMO descrie trei “moduri” in dezvoltarea software – organic, semidetasat si integrat.

In modul organic, echipe relativ mici de dezvoltători creează software intr-un mediu familiar, in-house. Majoritatea oamenilor conectaţi la proiect au experienţa in cadrul organizaţiei, si înţeleg cum sistemul aflat sub dezvoltarea lor contribuie la obiectivele organizatiei... Un proiect in modul organic este relaxat din punct de vedere al faptului cum sunt îndeplinite cerinţele si specificaţiile de interfaţa.

Modul semidetasat reprezintă un stagiu intermediar intre modurile organic si integrat. “Intermediar” înseamnă unul din aceste doua lucruri:

1. Un nivel intermediar ale caracteristicilor proiectului
2. Un amestec de caracteristici ale modurilor organic si integrat

Factorul de distincţie majora al proiectului in mod integrat este nevoia de a opera in constrângeri stricte. Produsul trebuie sa opereze intr-un complex bine definit de hardware, software, reguli si proceduri operaţionale, cum ar fi transferuri electronice de fonduri sau sistemul de control al zborurilor aeriene. In general, costurile schimbării unor parti ale acestui complex sunt atât de mari incat caracteristicile lor sunt considerate definitive, si software-ul se aşteaptă a fi in conformitate cu aceste specificaţii. Proiectul dezvoltat in mod integrat tinde a se dezvolta mai bine decât cel in mod organic. O data ce in proiectul dezvoltat in mod integrat a fost definitivat design-ul produsului, cea mai buna strategie este de a aduce o echipa foarte mare de programatori pentru a dezvolta un design detaliat, codare si testarea unitatilor in paralel. (Boehm 81)

Figura 2.1-4 arata efortul necesar pentru fiecare mod in funcţie de KSLOC. Acestea sunt curbe exponenţiale in COCOMO, si nu linii drepte. Figura 2.1-5 arata programarea in funcţie de KSLOC, iar Figura 2.1-6 arata eficienta. COCOMO modifica aceste curbe cu anumiţi “multiplicatori” ce reflecta produsul, platforma, personalul si atributele proiectului.

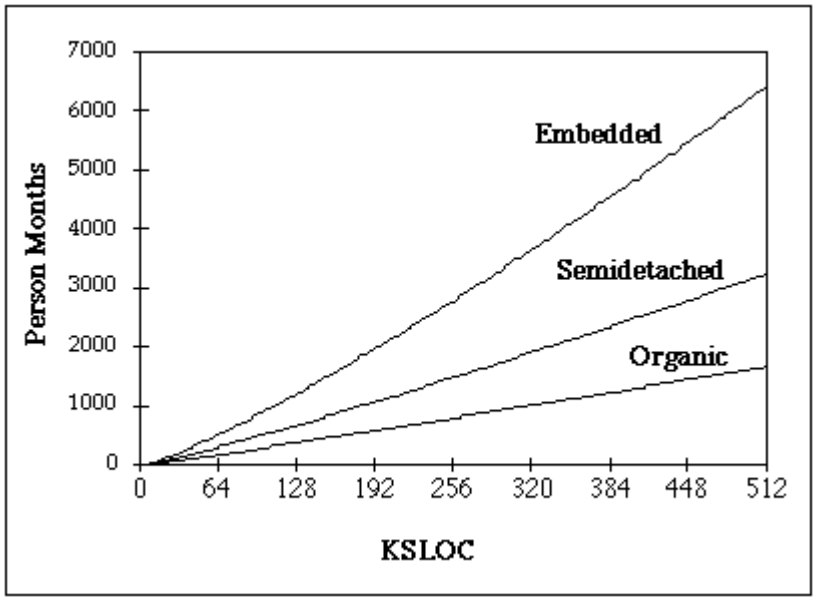

Figura 2.1-4: Efortul in functie de KSLOC

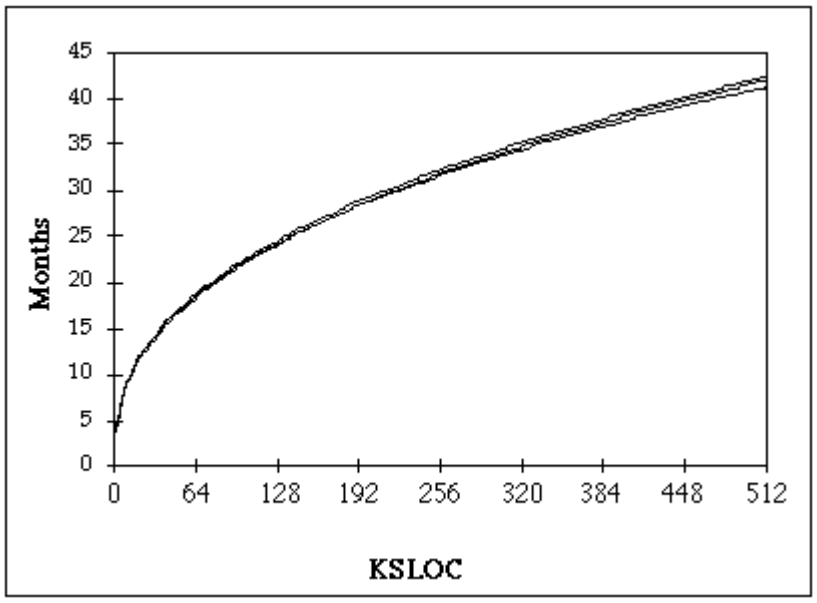

Figura 2.1-5: Programarea in functie de KSLOC

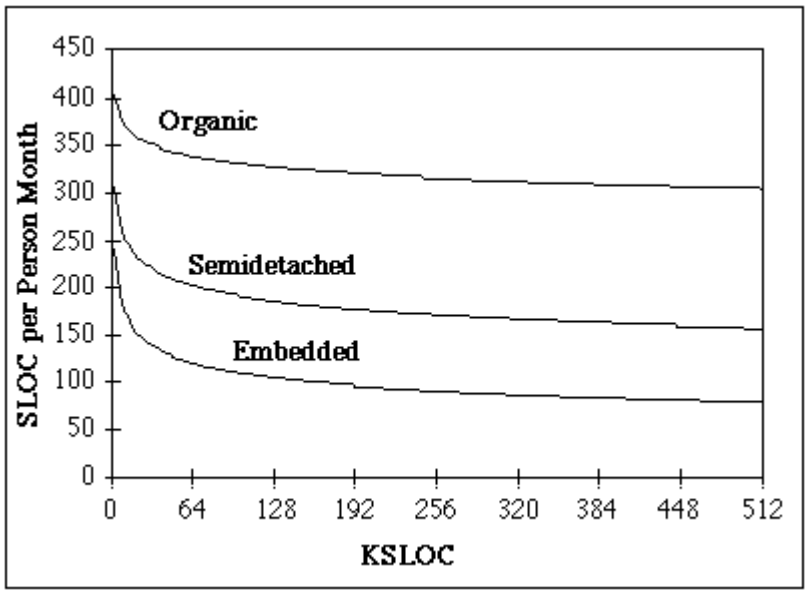

Figura 2.1-6: Eficienta in functie de KSLOC

2.2 Fiabilitatea

Metrici: Fiabilitatea

Sursa: Sursele originale apartin lui Zygmund Jelinski, Paul Moranda (1972) si Martin Shooman (1972).

Măsurători: Calitatea orientata pe user cum este arătata de eşecul unui system.

Metrici asociate: Mean Time Between Failure (MTBF), Intensitatea de eşec, Rata de Esec, Densitatea de eşec

Aplicaţii: Asigurarea calităţii, planificarea si monitorizarea testării sistemelor, operaţii si planificarea întreţinerii.

Definiţie: Probabilitatea ca un software sa nu cauzeze eşecul unui sistem pentru un timp specificat si in condiţii specifice. Probabilitatea este o funcţie a intrărilor in sistem, la fel ca si o funcţie a existentei eşecurilor in software. Intrările in sistem determina daca existenta eşecurilor este determinata.

Gama: Eşecul operaţional se încadrează intre 3x10-6 si 55x10-6 Esecuri per CPU Second. MTBF se întinde de la 5.1 pana la 92.6 CPU hours.

Densitatea eşecurilor la startul Sistemului Test variază de la 1 la 10 Esecuri pe KSLOC, cu o medie de 6 eşecuri pe KSLOC. KSLOC este considerat ca o linie sursa executabila, excluzând comentariile, declaraţii de date, cod refolosit, etc.

Număr de greşeli eliminat pe eşec: 0.995 Faults

Pentru mai multe informaţii :

John D. Musa, Anthony Iannino, and Kazuhira Okumoto, *Software Reliability: Measurement, Prediction*, Application, McGraw-Hill, 1987.

American National Standard Recommended Practice for Software Reliability, American Institute of Aeronautics and Astronautics, ANSI/AIAA R-013-1992; February 23, 1993.

Deşi mulţi factori sunt folosiţi pentru a descrie calitatea software – de exemplu portabilitatea, usabilitatea si mentenanta – fiabilitatea este cel mai des folosit. Deoarece se bazează pe apariţia unor probleme observabile sau eşecuri in produs, se situează printre cei mai uşori factori de calitate de măsurat. Este de asemenea un concept usor pentru utilizator de a se lega – acesta asteapta ca produsul sa fie fara erori, deci foarte fiabil. Mentananta si accesibilitatea sunt doi alti factori de calitate strans legati de fiabilitate. Mentenanta este măsurata prin Mean Time To Repair (MTTR) si accesibilitatea prin rata Mean Time Between Failures (MTBF) raportata la suma dintre MTBF si MTTR. Calcularea acestora se bazează din nou pe urmărirea eşecurilor.

Fiabilitatea este printre cele mai vechi si cunoscute metrici. Managerii de software au urmărit bug-uri in software inainte de a aparea conceptele de fiabilitate. Colectarea si urmărirea aşa-ziselor Software Problem Reports (SPRs) sau Software Trouble Reports (STRs) este inca o procedura utilizata in monitorizarea procesului de dezvoltare software pentru a atinge telurile fiabilitatii.(Section 2.2.1). In mod adiţional, multa munca incepand cu anii ’70 a fost depusa pentru a clarifica conceptele si modelarea fiabilităţii software. Aceasta a produs o viziune comuna a procesului de esec software, care se reflecta in definiţiile din Tabela 2.2-1. In esenţa, eşecurile, observate in timpul operării software-ului, sunt cauzate când software-ul intra intr-o stare in care un defect schimba rezultatul ieşirii. Intrările schimba starea

sistemului, dar nu localizazea problemele. Aceasta înţelegere conceptuala oferă fundaţiile unei tehnologii inginereşti pentru modelarea si controlul fiabilităţii software (Section 2.2.2), dar si procedurile pentru prezicerea fiabilităţii (Section 2.2.3). Aceasta tehnologie poate fi prezentata foarte pe scurt aici. Appendix C oferă resurse adiţionale pentru Ingineria Software a Fiabilităţii.

2.2.1 Profiluri de Defecte

Fiabilitatea a fost monitorizata iniţial prin urmărirea eşecurilor in timpul ciclului de viata. Eşecurile sunt colectate si catalogate in forme cunoscute sub nume variate, cum ar fi Software Trouble Reports (STRs) sau Software Problem Reports (SPRs). Diferite grafice ale datelor SPR pot oferi o privire in procesul dezvoltării de software. De exemplu, Figurile 2.1-1 si 2.2-2 arata doua din cinci grafice definite de United States Army's Software Test and Evaluation Panel (STEP) metrici ale “Profilelor defecte”.

Tabelul 2.2-1: Definiţii pentru Fiabilitatea Software

Error (1) O discrepanta intre o valoare observata, măsurata sau calculate sau condiţie si adevăr, specificat sau valoare teoretica corecta sau condiţie.

(2) Acţiunea umana care rezulta intr-un produs software care conţine o eroare. De exemplu omiterea sau interpretarea greşită a cerinţelor utilizatorului într-o aplicaţie software şi transpunerea incorectă sau omiterea unei cerinţe din specificaţiile de proiectare. Acesta nu este modul de folosire preferat.

Eşec (Failure) (1) Incapacitatea unui sistem sau a componentei unui sistem de a realiza o anumită funcţie cu limitările specificate. Un eşec poate să se producă la întâlnirea unei erori, ceea ce duce la pierderea serviciului aşteptat de către utilizator.

(2) Pierderea de către o unitate funcţională a capacităţii de a-şi desfăşura funcţia cerută.

(3) O depărtare a funcţionării programului faţă de cerinţele sale.

Rata eşecului (Failure Rate) (1) Procentul numărului de eşecuri de o anumită categorie sau severitate, într-o anumită perioadă de timp; de exemplu eşecuri pe lună. Este sinonim cu **intensitatea eşecului (failure intensity)**.

(2) Procentul numărului de eşecuri la o anumită unitate de măsură; de exemplu, eşecuri pe o anumită unitate de timp, eşecuri pe numărul de tranzacţii, eşecuri pe numărul de calculatoare care rulează.

Eroare (Fault) (1) Un defect în cadrul codului care poate să fie cauza a unuia sau a mai multor eşecuri.

(2) O condiţie accidentală care cauzează eşecul unei unităţi funcţionale în a-şi realiza funcţia cerută. Sinonim cu bug.

Calitate Totalitatea caracteristicilor şi trăsăturilor unui produs sau a unui serviciu care se susţine pe capacitatea sa de a satisface anumite nevoi precizate.

Calitatea software-ului (1) Totalitatea caracteristicilor şi trăsăturilor unui produs software care se susţine pe capacitatea sa de a satisface anumite nevoi precizate; de exemplu de a se conforma specificaţiilor.

(2) Gradul în care software-ul are a anumită combinaţie de atribute dorite.

(3) Gradul în care un client sau un utilizator percepe că software-ul îndeplineşte aşteptările sale.

(4) Caracteristicile compuse ale software-ului care determină gradul în care software-ul în timpul folosirii sale va îndeplini aşteptările clienţilor.

Siguranţa Softwareului (Software Reliability) (1) Probabilitatea că software-ul nu va cauza un eşec al sistemului de-a lungul unei anumite perioade de timp şi în conformitate cu anumite condiţii. Această probabilitate este funcţie de intrări şi funcţie de utilizarea sistemului, precum şi o funcţie a existenţei erorilor în cadrul software-ului. Intrările sistemului determină dacă erorile existente (dacă sunt) vor fi şi activate.

(2) Capacitatea unui program de a îndeplini o funcţie cerută în anumite condiţii şi pentru o perioadă de timp precizată.

Definiţii preluate din (AIAA 92)

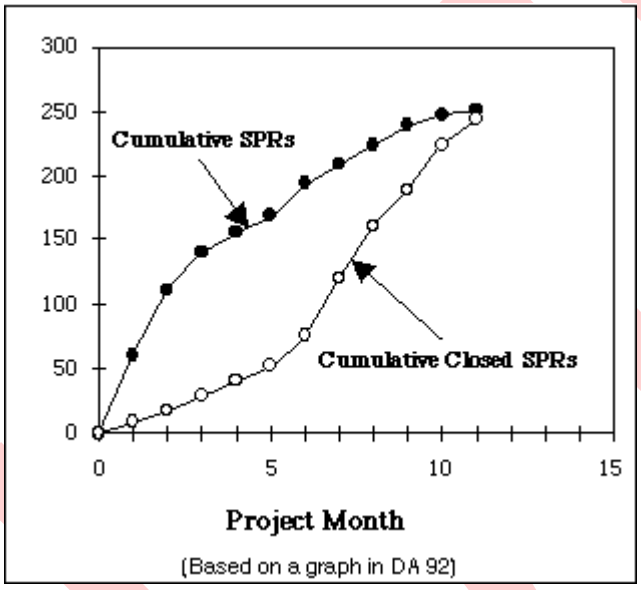

Figura 2.2-1: Cumulative Software Problem Reports

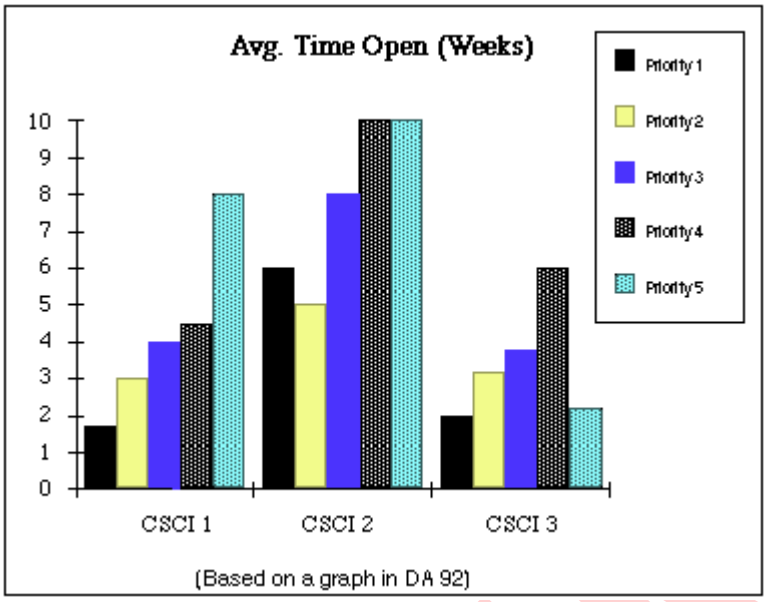

Figura 2.2-2: Average SPR Age

Astfel de grafice pot să fie examinate de-a lungul unui proiect software pentru a detecta tendinţe, pentru a certifica progresul dezvoltării şi pentru a putea face comparaţii cu proiecte asemănătoare. Adesea, erorile sunt sumarizate prin normalizare relativ la dimensiunea produsului. De exemplu, . arată numărul de erori pe mii de linii sursă de cod (Thousand Source Lines Of Code - KSLOC). Cititorul poate să folosească acest tabel ca sursă pentru valorile obişnuite ale densităţii erorilor în absenţa datelor viitoare din mediul său de lucru.

Application	**Systems**	**KSLOC**	**Average Faults/KSLOC**	**Standard Deviation**
Airborne	7	541	12.8	9.4
Strategic	21	1,794	9.2	14.0
Tactical	5	88	7.8	6.1
Process Control	2	140	1.8	0.3
Production Center	12	2,575	8.5	9.5
Developmental	4	97	12.3	9.3
Total/Average	**51**	**5,236**	**9.4**	**11.0**

Tabelul 2.2-2: Densitatea erorilor (Reprodus după McCall 87)

2.2.2 Software Reliability Engineering

Deşi urmărirea raportărilor de probleme ale software-ului este o metodologie folositoare în monitorizarea procesului de dezvoltare software, ea nu foloseşte în stabilirea unei metrici predictive de încredere, orientată pe utilizator. Modelele de siguranţa softwareului, care relaţionează datele despre erori cu un model statistic a procesului de eşec a software-ului, şi sunt folosite pentru a specifica, a prezice şi a estima siguranţa sistemelor software.

Disciplina Software Reliability Engineering a evoluat de la dezvoltarea şi aplicarea acestor modele probabilistice a procesului de eşec a software-ului.

Multe dintre modele sunt folosite cu rezultate bune. În mod particularizat, aceste modele sunt apicate în timpul testării sistemului sau în timpul înreţinerii sale, prin colectarea datelor de eşec, încadrarea în model şi actualizarea rezultatelor pe baza datelor adiţionale. Dacă potrivirea este bună, modelul poate să fie folosit relativ cu încredere pentru a oferi o evaluare a siguranţei curente sau pentru a predicţiona comportamentul de eşec viitor. De exemplu, dacă un model s-a dovedit a fi precis în timp şi pentru creşteri anterioare ale produsului software, atunci folosirea sa duce la a avea încredere în noile rezultate. Pentru a fi expliciţi, acest raport descrie Modelul de Bază a Execuţiei în Timp a Musa, unul dintre modelele de siguranţa software-ului cele mai mult acceptate.

Să presupunem că un sistem software operează într-un mediu cu un profil operaţional care nu se schimbă. Cu alte cuvinte, distribuţia tipurilor de utilizatori cere sau necesită ca capabilităţile sistemului să nu varieze în timp (Musa 93). În plus, să presupunem că în timpul operaţiunilor nu se aduce nici o schimbare software-ului. Atunci software-ul ar putea să fie modelat cu o rată constantă de eşec λ. Atunci, numărul aşteptat de eşecuri dintr-un interval de timp este proporţional cu lungimea intervalului de timp considerat. Probabilitatea ca software-ul să opereze fără eşec, siguranţa R(τ), devine cu atât mai mică, cu cât perioada de timp luată în considerare este mai mare. Figura 2.2-3 arată aceată relaţie, în care siguranţa descreşte exponenţial cu timpul de execuţie τ. Pentru sistemele cu o rată de eşec constantă, Timpul Mediu Între Eşecuri, calculat ca inversul ratei de eşec, este adesea folosit pentru a sumariza siguranţa

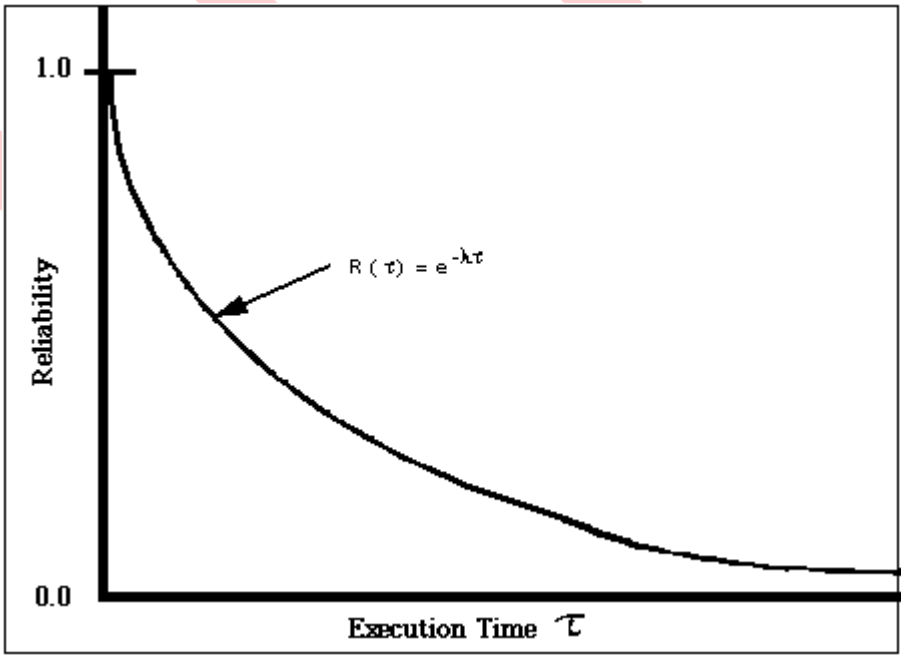

Figura 2.2-3: Reliability Function

Software-ul va fi schimbat în timpul testării sitemului pentru a îndepărta erorile descoperite de către sistemul de operare. Îndepărtarea erorilor observabile va duce la creşterea siguranţei software-ului. De aceea, testarea sitemului este modelată pentru a arăta creşterea siguranţei.

Modelul de Bază a Execuţiei în Timp Musa se bazează pe presupunerea că toate erorile contribuie egal la rata de eşec. Astfel, rata de eşec este o funcţie liniară descrescătoare a numărului aşteptat de erori $\mu(\tau)$ (Figura 2.2-4). Parametrii de bază ai modelului Musa sunt ν_0, numărul aşteptat de eşecuri necesare pentru a îndepărta roate erorile, şi β, descreşterea aşteptată a ratei de eşecuri pe eşec. Aceşti parametrii definesc şi alte funcţii de interes. Figura 2.2-5 arată numărul aşteptat cumulat de eşecuri, iar Figura 2.2-6 arată modul în care rata de eşec variază în timp.

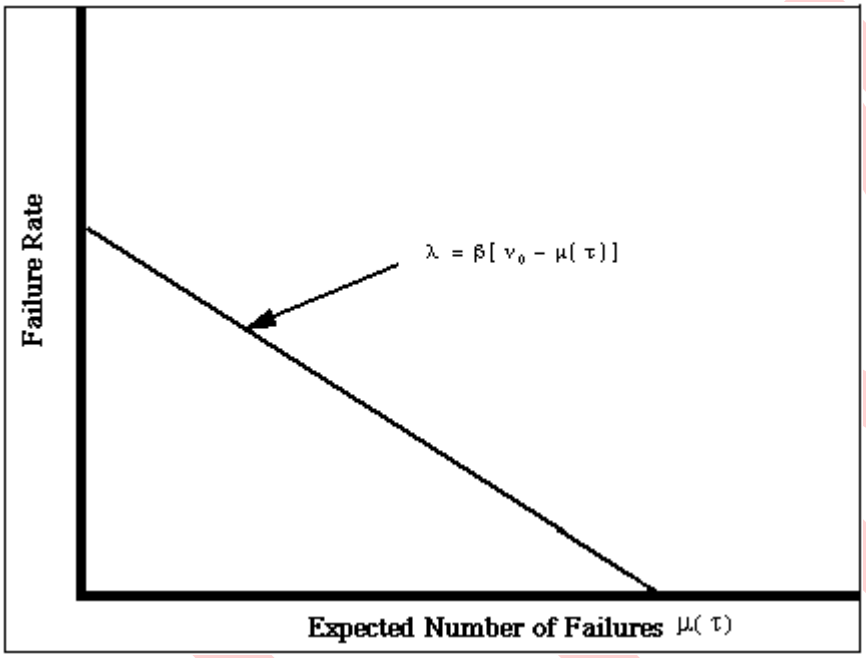

Figura 2.2-4: Rata de eşec vs. Eşecurile aşteptate

Fiind daţi parametrii modelului, β şi ν_0, se poate determina cât timp trebuie să se desfăşoare procesul de testare astfel încât orice obiectiv în privinţa siguranţei este atins. Graniţele de încredere ar trebui să fie folosite pentru o regulă de oprire a siguranţei, în loc de estimările făcute. De exmplu, testarea ar putea să se oprească atunci când 90% din graniţa de încredere superioară relativ la numărul de eşecuri rămase este mai jos de o graniţă ţintă. Ca alternativă, testarea ar putea să se oprească atunci când costul total al ciclului de viaţă este minimizat. Costul unui eşec este mai mare pe teren, decât la testarea sistemului. Beneficiul marginal al testării pentru o incrementare a timpului de execuţie este descreşterea aşteptată a costului eşecului, care se socoteşte pentru numărul aşteptat de eşecuri din acea incrementare. Costul marginal al testării constă în resursele necesare pentru a testa pentru o incrementare a timpului de execuţie. Pentru a minimiza timpul total, testarea ar trebui să continue până când beneficiul marginal cade dedesubtul costului marginal (Vienneau 91).

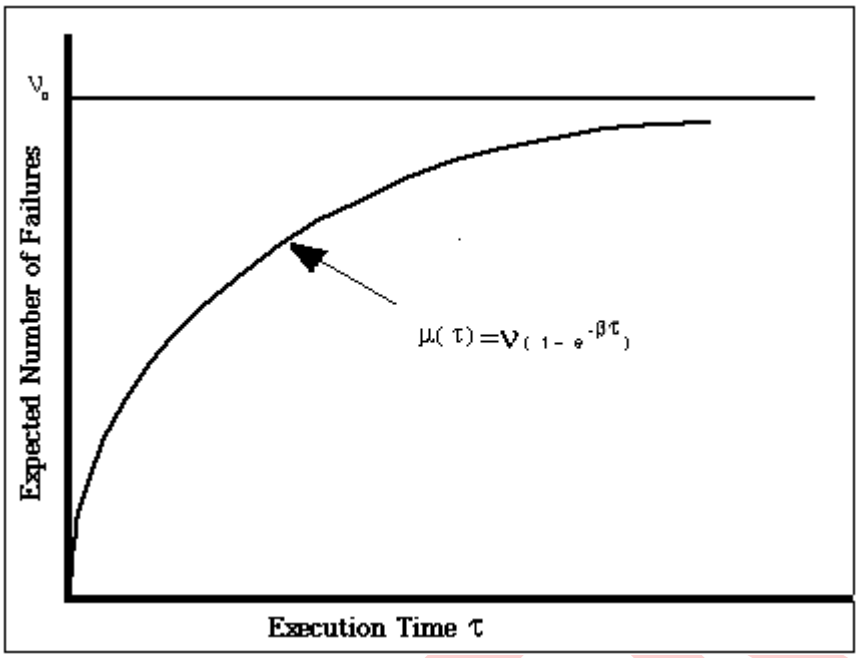

Figura 2.2-5: Eşecuri aşteptate vs. Timp

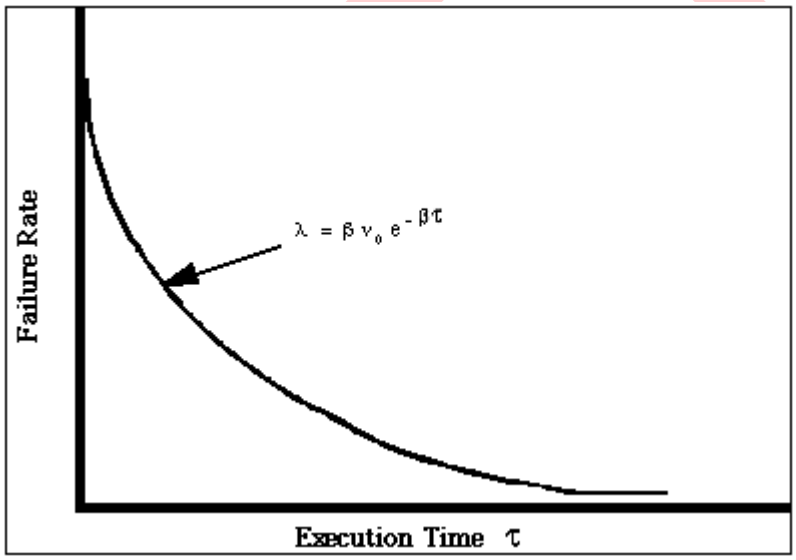

Figura 2.2-6: Rata eşecului vs. Timp

2.2.3 Predicţia siguranţei

Modelul Musa al Execuţiei de Bază în Timp descrie comportamentul eşecului unui sistem în timpul operării şi a testării sitemului, fiind daţi parametrii modelului β şi ν_0, descrşterea aşteptată a ratei eşecului pe eşec şi respective numărul eşecurilor necesare pentru a înlătura toate erorile. Siguranţa poate să fie predicţionată prin estimarea acestor parametrii pornind de la o serie de alţi parametrii cunoscuţi mai devreme în ciclul de viaţă. Un standard comercial recent, şi anume American Standard Recommended Practice for Software Reliability (AIAA 93), şi un standard military schiţat, *Military Handbook Hardware/Software Reliability Assurance and Control* (DRAFT) (RL 91), conţin proceduri similare de utilizare a modelului

Musa pentru predicţionarea siguranţei. Aceste proceduri de predicţionare se bazează pe un set de trei parametrii, din care parametrii modelului Musa pot să fie derivaţi:

- Rata de eşec iniţială, λ_0
- Numărul de erori din program la începutul testării sistemului, ω_0
- Factorul de reducere al erorilor, B

Rata eşecului pe eroare, ϕ, este definit funcţie de aceşti parametrii:

$$\phi = \frac{\lambda_0}{\omega_0} \qquad (2.2\text{-}1)$$

Rata eşecului pe eroare, ϕ, diferă de rata eşecului per eşec, β, din cauza greşelilor care se fac în timpul debuggingului. Uneori, debuggingul după un eşec nu reduce numărul de erori. Factorul de reducere al erorilor, B , modelează debuggingul imperfect. Factorul de reducere al erorilor este numărul aşteptat de erori care au fost îndepărtate pe eşec. Factorul de reducere al erorilor este, în general, subunitar. Ecuaţiile 2.2-2 şi 2.2-3 arată unele relaţii dintre aceşti parametrii:

$$\beta = B\ \phi = \frac{\lambda_0}{\omega_0} \qquad (2.2\text{-}2)$$

$$\nu_0 = \frac{\omega_0}{\beta} \qquad (2.2\text{-}3)$$

Standardul militar recomandă utilizarea pentru B a valorii de 0.955 erori pe eşec.

Numărul de erori moştenite, ω_0, este predicţionat ca fiind produsul dintre dimensiunea software-ului şi densitatea aşteptată a erorilor. Dimensiunea este măsurată în KSLOC, cu excluderea codului sursă reutilizat şi neexecutabil. Densitatea erorilor este numărul de erori per KSLOC. O gamă obişnuită a densităţii erorilor la începerea testării sistemului este între 1 eroare per KSLOC şi 10 erori per KSLOC. Valoarea mai mică a intervalului este pentru programatorii cu experienţă extinsă relative la aplicaţia respectivă şi într/un mediu foarte disciplinat (AIAA 92). Standardul militar recomandă utilizarea valorii de 6 erori pe KSLOC (RL 91). Tabelul 2.2-2 (din secţiunea 2.2.1) prezintă unele date empirice despre densitatea erorilor, bazate pe date operaţionale din mai multe sisteme.

Standardul militar include şi procedurile de predicţie pentru rata de eşec iniţială λ_0. Aceste proceduri devin din ce în ce mai detaliate, odată cu desfăşurarea ciclului de viaţă. Rata de eşec iniţială este estimată în timpul fazei cerinţelor:

$$\lambda_0 = K\ \omega_0\ r \qquad (2.2\text{-}4)$$

unde ω_0 este numărul de erori moştenite predicţionat mai sus, r este viteza medie a procesorului, considerată în instrucţiuni pe secundă, I este numărul de instrucţiuni obiect, iar K este o constantă de proporţionalitate, “rata de expunere a erorilor”. Viteza medie a procesorului, o caracteristică a calculatorului şi a mix-ului de instrucţiuni, poate să fie găsită prin benchmarking. Numărul de instrucţiuni obiect este găsit prin înmulţirea numărului de

SLOC cu o rată de expansiune a codului. Rate de expansiune a codului representative sunt prezentate în tabelul 2.2-3. În absenţa datelor istoric pentru un anumit mediu, ar trebui să se folosească 4.29*10-7 eşecuri per eroare, ca valoare pentru K . Valorile recomandate pentru constantele utilizate în predicţia parametrilor modelului Musa sunt rezumate în Tabelul 2.2-4. Tabelul 2.2-5 prezintă un exemplu despre cum să combini aceste rezultate în vederea predicţionării MTBF-ului unui system software de dimensiune medie.

Tabelul 2.2-3: Ratele de expansiune a codului

Programming Language	Expansion Ratio
Assembler	1
Macro Assembler	1.5
C	2.5
Cobol	3
FORTRAN	3
Jovial	3
Ada	4.5

Din (RL 91)

Tabelul 2.2-4: Câţiva parametrii pentru predicţia siguranţei

Parameter	Expected Value	Range
Fault Reduction Factor, B (Average Number of faults corrected per failure)	0.955 Faults per Failure	
Fault Density	6 Faults per KSLOC	1 to 10
Fault Exposure Ratio, K	4.20×10^{-7} Failures per Fault	1.41×10^{-7} to 10.6×10^{-7}

Tabelul 2.2-5: Exemplu de predicţie

System Characteristics	
System size:	50 KSLOC
Fault Density:	6 Faults per KSLOC (Average)
Average Processor Speed:	r = 100 x 10 Instructions per Second
Code Expansion Ratio:	3 Object Instructions per SLOC (Cobol)
Fault Exposure Ratio:	K = 4.20 x 10^{-7} Failures per Fault
Fault Reduction Factor:	B = 0.955 Faults per Failure
System Test Period:	1000 CPU Hours
Calculations	
Object Instructions:	I = 50,000 x 3 = 150,000 Instructions
Inherent Faults:	ω_0 = 50 x 6 = 300 Faults

Initial Failure Rate:	$\lambda_0 = (4.20 \times 10^{-7})(300)(100 \times 10^6)/(150{,}000)$ $\lambda_0 = 0.084$ Failures per Second $\lambda_0 = 2.3 \times 10^{-5}$ Failures per Hour
Failure Rate per Fault:	$\Phi = (2.3 \times 10^{-5})/(300)$ $\Phi = 7.8 \times 10^{-8}$ Failures per Hour per Fault
Musa Basic Execution Time Model Parameters	
Expected Number of Failures:	$\nu_0 = (300)/(0.955) = 314$ Failures
Failure Rate per Failure:	$\beta = (0.955)(7.8 \times 10^{-8})$ $\beta = 7.4 \times 10^{-8}$ Failures per Hour per Failure
Failure Rate at Release:	$\lambda = (7.4 \times 10^{-8})(314)$ e-0.000074 $\lambda = 2.3 \times 10^{-5}$ Failures per CPU Hour
MTBF at Release:	43,000 CPU Hours

2.3 Complexitatea

Metrica: Complexitatea

Măsoară: Dimensiunea, fluxul de control, structurile de date, structurile intermodulare, etc.

Metrice înrudite: Linii sursă de cod (SLOC), Complexitatea Ciclomatică a lui McCabe, metricele Ştiinţei Software a lui Halstead, metrica Fluxului de Informaţii a lui Henry şi Kafura, metricele de acoperire a testării.

Aplicaţii: Asigurarea calităţii în timpul dezvoltării, predicţia caracteristicilor operaţionale, planificarea testării.

Definiţie: Complexitatea Ciclomatică este complexitatea teoretică a grafului fluxului unui program al unui calculator. Cu alte cuvinte, numărul de arii inchise de schiţa fluxului.

Gamă: De obicei este determinată de decizia de management. Complexitatea ciclomatică trebuie să fie pozitivă, cu o limită maximă recomandată la valoarea de 10.

Observaţii: Este cel mai bine dezvoltată pentru aplicaţii la subrutine individuale.

Productivitatea este o preocupare a managementului, în timp de siguranţa este un factor de calitate direct visibil pentru utilizatorii sistemelor software. Aceste atribute vizibile din exterior ale proceselor şi produselor software sunt puternic influenţate de atributele de inginerie ale software-ului, cum ar fi complexitatea. Software-ul bine proiectat prezintă un minim de complexitate care nu este necesară. Complexitatea negestionată duce la un software dificil de utilizat, de întreţinut şi de modificat. Ea cauzează costuri de dezvoltare crescute şi duce la depăşirea termenelor. Dar unele elemente ale software-ului sunt complexe prin moştenire – conformarea la interfeţe externe arbitrare, presiunea de adaptare la cerinţele schimbătoare ale utilizatorilor, lipsa reprezentării evidente a vizualizării softwrae-ului (Brooks 87).

Controlarea şi măsurarea complexităţii este o problemă provocatoare, atât pentru management, cât şi pentru inginerie şi pentru cercetare. Au fost create metrici pentru a măsura variatele aspecte ale complexităţii, cum ar fi dimensiunea absolută, fluxul de contol, structurile de date şi structura intermodulară. Cele mai bine acceptate sunt probabil SLOC (Secţiunea 2.1.2) şi complexitatea ciclomatică (McCabe 76). Metricele complexităţii, de altfel nediscutate aici, include metricele Ştiinţei Software a lui Halstead şi metrica Fluxului Informaţional a lui Henry şi Kafura. Etricele lui Halstead sunt controversate. Metrica lui Henry şi Kafura este una dintre primele dintr-o familie care se adresează complexităţii proiectării, complexităţii presupuse de conexiunile dintre module. Altfel importantă, măsurarea acestui aspect a complexităţii, spre deosebire de măsurarea complexităţii unui singur modul, este încă o problemă de cercetare.

Complexitatea ciclomatică, V(G), aplică teoria grafurilor pentru a măsura complexitatea fluxului de control. Intuitiv, metrica este numărul de regiuni conectate dintr-un flowchart, pentru o subrutină cu o intrare şi o ieşire. Figura 2.3-1 arată un flowchart pentru care metrica McCabe este 3. Metrica McCabe poate să fie calculată şi din numărul de muchii şi de noduri din flowchart. Din motive teoretice, se adauga mai întâi o muchie imaginară pornind de la nodul terminal şi până la nodul iniţial. Cu această modificare, metrica McCabe se caalculează astfel:

$$V(G) = (\#\ Edges) - (\#\ Nodes) + 1 \qquad (2.3\text{-}1)$$

Sunt 7 muchii şi 5 noduri în exemplu. Deci, încă o dată, complexitatea ciclomatică este 3.

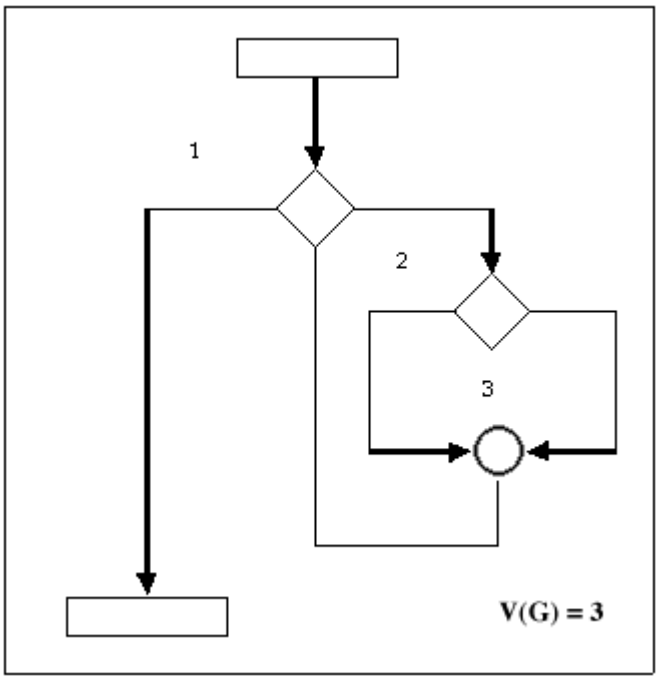

Figura 2.3-1: A Flow Chart Example

Metricile de complexitate sunt colectate de către un analizor static din codul sursă (Appendix D). Ele pot să fie colectate din proiecte, dacă proiectele sunt scrise într-un limbaj formal, adică în Program design Language (PDL). Aceste metrici sunt în principal folosite în asigurarea calităţii. Parametrii de ghidaj cuantificabili pot să fie forţaţi pentru toate modulele.

De exemplu, unele organizaţii mandatează că toate subrutinele conţin mai puţine linii de cod decât ar încăpea pe o pagină şi că au o metrică Ciclomatică mai mică decât 10, fiind necesare proceduri speciale pentru a obţine un waiver. De exemplu, un waiver poate fi acordat unei subrutine care este one large & statement. Dl. George Stark et. al. (94) descrie o aplicaţie a complexităţii Ciclomatice în care rutinelor individuale li se permite să depăşească o valoare a metrii de 10, dar distribuţia metricii la toate subrutinele dintr-un sistem trebuie să se încadreze în anumite linii de ghidare. Figura 2.3-2 afişază grafic aceste linii de ghidare. Se poate determina fracţiunea subrutinelor care sunt sub o anumită valoare a metricii ca funcţie de valoarea metricii. Liniile de ghidare cer ca această Funcţie de Distribuţie Cumulativă (Cumulative Distribution Function (CDF)) să exceadă curba cea mai de jos din Figura 2.3-2, şi este recomandat ca să exceadă curba care separă regiunile "Straightforward" şi "Standard".

Figura 2.3-2 ilustrează aceste linii de ghidare prin aplicarea lor la Common Ada Missile Packages (CAMP). CAMP este o colecţie de pachete Ada reutilizabile care constă în peste 100 000 de linii sursă de cod, organizată în 2 507 pachete Ada cuprinzând 10 categorii (CAMP 87). Curba pentru CAMP din Figura 2.3-2 este derivată din măsurătorile a 1 474 de proceduri, funcţii sau task-uri Ada din CAMP (Staff 92). Trebuie să se aibă în vedere că marea majoritate a subprogramelor CAMP au avut o complexitate ciclomatică mai mică decât limita superioară recomandată de 10, şi că curba CAMP se află în regiunea "Straightforward" pentru toate valorile complexităţii ciclomatice, cu execpţia celei mai mici. Deci, după această măsurare, CAMP este un sistem direct. Această curbă CAMP nu poate fi considerată ca fiind reprezentativă pentru toate sistemele, din moment ce distribuţia complexităţii ciclomatice variază semnificativ odată cu sitemul considerat, chiar şi dacă sistemele folosesc acelaşi limbaj de programare (Staff 92).

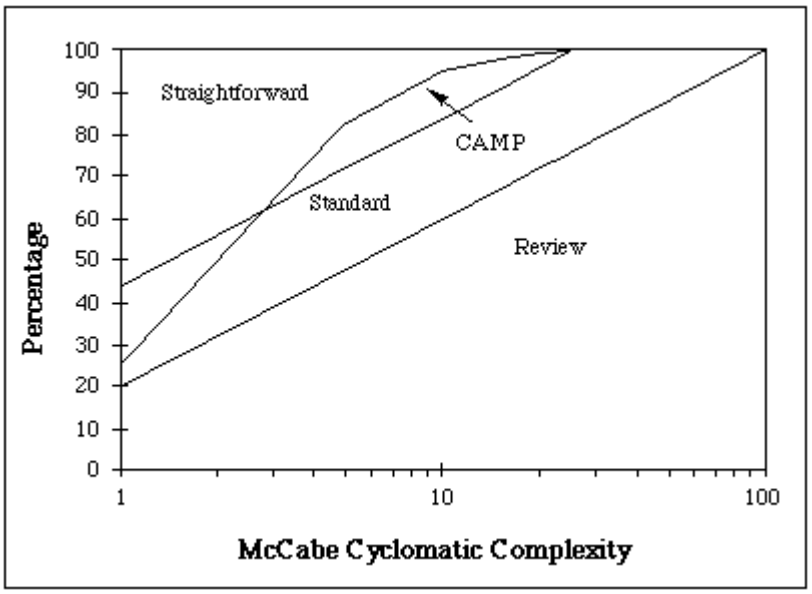

Figura 2.3-2: Distribuţia Complexităţii Ciclomatice

3.0 REFERENCES

(AFSC 87) *Software Quality Indicators*, AFSC Pamphlet 800-14, Air Force Systems Command, January 1987.

(AFSC 88) *Software Management Indicators*, AFSC Pamphlet 800-43, Air Force Systems Command, June 1988.

(AIAA 93) *American National Standard Recommended Practice for Software Reliability*, American Institute of Aeronautics and Astronautics, ANSI/AIAA R-013-1992; February 23, 1993.

(Albrecht 79) Allan J. Albrecht, "Measuring Application Development Productivity," *Proceedings of the IBM Application Development Symposium*, Monterey, California, October 1979, pp. 83-92.

(Albrecht 83) Allan J. Albrecht and John E. Gaffney, Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering*, Volume SE-9, Number 6, November 1993, pp. 639-648.

(Brand 95) Stewart Brand, "The Physicist," *Wired* 3.09, September 1995.

(Boehm 81) Barry W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

(Brooks 87) Frederick P. Brooks, Jr., "No Silver Bullet," *Computer*, April 1987, pp. 10-19.

(Carleton 92) A. Carleton et al., *Software Measurement for DOD Systems: Recommendations for Initial Core Measures*, Software Engineering Institute, CMU/SEI-92-TR-19, 1992.

(CAMP 87) *User's Guide for the Missile Software Parts of the Common Ada Missile Packages (CAMP) Project*, McDonnell Douglas Astronautics Company, 30 October 1987.

(Chruscicki 95) Andrew J. Chruscicki and John J. Marciniak, *National Software Initiative: Report of the Cooperstown Workshops*, Data & Analysis Center for Software, 7 June 1995.

(DA 92) "Software Test and Evaluation Guidelines," *Test and Evaluation Procedures and Guidelines*, Part Seven, (Draft), DA Pamphlet 73-1, Headquarters, Department of the Army, 30 September 1992.

(Grady 87) Robert B. Grady and Deborah L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.

(Grady 92) Robert B. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, 1992.

(IEEE 88a) *IEEE Standard Dictionary of Measures to Produce Reliable Software*, Institute of Electrical and Electronics Engineers, IEEE Standard 981.1-1988.

(IEEE 88b) *Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software*, Institute of Electrical and Electronics Engineers, IEEE 981.2-1988.

(Jelinski 72) Z. Jelinski and P. B. Moranda, "Software Reliability Research," *Statistical Computer Performance Evaluation*, Editor: W. Freidberger, Academic Press, 1972.

(Jones 86) Capers Jones, *Programming Productivity*, McGraw-Hill, 1986.

(Kemerer 87) Chris F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, Volume 30, Number 5, May 1987, pp. 416-429.

(McCabe 76) Thomas McCabe, "A Software Complexity Measure," IEEE Transactions on Software Engineering, Volume SE-2, December 1976, pp. 308-320.

(McCall 87) J. McCall et. al., Methodology for Software Reliability Prediction, (2 volumes), Rome Air Development Center, RADC-TR-87-171, November 1987.

(Musa 87) John D. Musa, Anthony Iannino, and Kazuhira Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, 1987.

(Musa 93) John D. Musa, "Operational Profiles in Software-Reliability Engineering," *IEEE Software*, March 1993, pp. 14-32.

(Paulk 95) Mark C. Paulk et. al., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1995.

(RL 91) *Military Handbook Hardware/Software Reliability Assurance and Control*, (DRAFT), Rome Laboratory, MIL-HDBK-XXX, 6 December 1991.

(Shooman 72) Martin L. Shooman, "Probabilistic Models for Software Reliability Prediction", *Statistical Computer Performance Evaluation*, Editor: W. Freidberger, Academic Press, 1972.

(Springsteen 94) Beth Springsteen et al., *Survey of Software Metrics in the Department of Defense and Industry*, IDA Paper P-2996, Institute for Defense Analysis, April 1994.

(Staff 92) Staff, *Ada Usage Profiles*, Kaman Sciences Corporation, October 30, 1992.

(Stark 94) George Stark, Robert C. Durst, and C. W. Vowell, "Using Metrics in Management Decision Making," *Computer*, September 1994, pp. 42-48.

(Vienneau 91) Robert L. Vienneau, "The Cost of Testing Software," *Annual Reliability and Maintainability Symposium*; Orlando, FL; January 29-31, 1991.

(Vienneau 95) Robert L. Vienneau, "The Present Value of Software Maintenance," *Journal of Parametrics*, Volume XV, Number 1, April 1995, pp. 18-36.

APPENDIX A: ACRONYMS

ANOVA - Analysis Of Variance

CAMP - Common Ada Missile Packages

CDF - Cumulative Distribution Function

CMM - Capability Maturity Model

COCOMO - Constructive Cost Model

CPU - Central Processing Unit

CSC - Computer Sciences Corporation

CSCI - Computer Software Configuration Item

CSU - Computer Software Unit

DoD - Department of Defense

FP - Function Point

GSFC - Goddard Space Flight Center

IFPUG - International Function Points User's Group

KSLOC - Thousands of Source Lines of Code

MIS - Management Information System

MTBF - Mean Time Between Failures

MTTR - Mean Time To Repair

NASA/SEL - National Aeronautics and Space Administration Software Engineering Laboratory

NSDIR - National Software Data and Information Repository

PDL - Program Design Language

PIP - Process Improvement Paradigm

PSM - Practical Software Measurement

SEI - Software Engineering Institute

SEMA - Software Engineering Measurement and Analysis

SPR - Software Problem Report

SRE - Software Reliability Engineering

STEP - Software Test and Evaluation Panel

STR - Software Trouble Report

SLOC - Source Lines of Code

APPENDIX C: REFERENCES FOR SOFTWARE RELIABILITY

Standards

American National Standard Recommended Practice for Software Reliability (AIAA 92)

Military Handbook Hardware/Software Reliability Assurance and Control (RL 91)

IEEE Standard Dictionary of Measures to Produce Reliable Software (IEEE 88a)

Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software (IEEE 88b)

Textbooks

Software Reliability: Measurement, Prediction, Application (Musa 87)

Software Tools

Computer Aided Software Reliability Estimation (CASRE) Available from:

COSMIC
University of Georgia
382 East Broad Street
Athens, GA 30602-4272
Phone: (706) 542-3265; Fax: (706) 542-4807
Internet: service@cossack.cosmic.uga.edu
Bitnet: COSMIC@UGA
URL: http://www.cosmic.uga.edu/

APPENDIX D: SOME VENDORS OF COMPLEXITY MEASUREMENT TOOLS

AdaMat
Dynamic Research Corporation
60 Frontage Road

Andover, MA 01810
Voice: (800) 522-7321; Voice: (508) 475-9090
FAX: (508) 475-2157
E-mail: cmcquire@s1.drc.com

Amadeus
Amadeus Software Research, Inc.
President: Richard W. Selby
10 Young Court
Irvine, CA 92715
Voice: (714) 725-6400
FAX: (714) 725-6411
E-mail: selby@amadeus.com

AdaQuest
General Research Corporation
PO Box 6770
5383 Hollister Ave
Santa Barbara, CA 93160-6770
Voice: (805) 964-7724

McCabe & Associates, Inc.
Mr. Thomas McCabe
Twin Knolls Professional Park
5501 Twin Knolls Road, Suite 111
Columbia, Maryland 21045
Voice: (800) 638-6316
FAX: (410) 995-1528
URL: http://www.mccabe.com

SET Laboratories Incorporated
PO Box 868
Mulino, OR 97042
Voice: (503) 289-4758
FAX: (503) 829-7220
URL: http://www.molalla.net/~setlabs/

Logiscope
Verilog, Inc.
3010 LBJ Freeway, Suite 900
Dallas, Texas 75234
Voice: (214) 241-6595
FAX: (214) 241-6594
E-mail: info@logtech.com