Universitatea Politehnică Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Inginerie Software

“ Evaluarea fiabilităţii produselor cu ajutorul metricilor software”

Profesor coordonator: prof. univ. dr. ing. Ştefan Stăncescu

Studenţi: Vică Daniel-Cătălin, Sipică Alin-Marian

Grupa: 441A (CTI)

Anul universitar

2014-2015

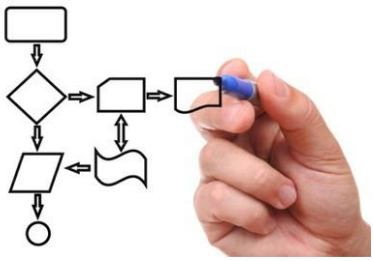
CUPRINS

INTRODUCERE

1.1 Notiuni generale despre metricii software

In lucrarea de fata, dupa cum se poate deduce cu usurinta din titlu, ne propunem scoaterea in evidenta a rolului de evaluator pe care il au metricele software asupra produselor aferente domeniului. Insa, precum in orice proiect de specializare, indiferent de pregatirea academica a cititorului, se impune o parcurgere sistematica, pornind de la principii de baza ce ne vor ajuta in cladirea ulterioara a mecanismelor complexe („Inainte de a putea alerga, trebuie mai intai sa stim a merge!”). Asadar, putem incepe acest capitol prin incercarea unei definitii cat mai clare asupra ingineriei software. Putem afirma faptul ca aceasta este o ramura a domeniului ingineresc, ce are in vedere toate aspectele fabricarii unui program (vom utiliza acest termen in scopul evitarii repetitive a cuvantului „software”), optimizat din punct de vedere atat functional, cat si economic. Ingineria software isi propune astfel cuantificarea unor notiuni abstracte in majoritatea cazurilor, spre utilizarea acestora in scopuri „umane” (reducerea costului, spre exemplu), adesea fiind privita de catre multi specialisti din domeniu sub ideea „aplicarii principiilor ingineriei de sunet spre a obtine un software pe care te poti baza in orice circumstante”.

Desi, la prima vedere, nu pare un domeniu foarte vast, ingineria software prezinta o multitudine de discipline, fiecare cu un rol binedefinit, precum proiectarea software (procesul de definire a arhitecturii, componentelor, interfetelor si altor parametrii ai sistemului, in urma caruia se pot pune bazele programului necesar in rezolvarea problemei in cauza), testare software (gandirea si aplicarea unor algoritmi ce doresc descoperirea si exploatarea punctelor slabe ale sistemului), si multe altele. Pe baza celor spuse pana in momentul actual, iar totodata cercetand documentatii de specialitate precum „Raportul Tehnic ISO/EIC 1979:2004” publicat de catre asociatia IEEE (acronim pentru „Institute of Electronical and Electrical Engineering), putem conclude faptul ca ingineria software prezinta o gama larga de oportunitati, bucurandu-se de un numar in crestere al practicantilor, insa totodata, si al criticilor („Precum economia este cunoscuta sub numele de „Stiinta redundanta”, ingineria software poate fi privita drept „Stiinta fara viitor” [...] fiind astfel un mod de acceptare al ideei „Cum sa programezi, daca nu cunosti.” – citat din Edsger W. Dijkstra, dupa perioada de „criza software” din 1972).



(Diagrame de functionalitate, des utilizate in proiectarea software)
(http://www.enisa.europa.eu/activities/ Resilience-and-CIIP/criticalapplications/ Software_design.jpg/ image_preview)

Avand in minte ideea centrala a disciplinei ce sta la baza acestei lucrari, putem reveni asupra definirii metricelor software, ce au luat nastere din necesitatea evaluarii produselor software. Dupa cum spune si denumirea („metrice” – „metric” – „metru” – „masuratoare”), metricele software reprezinta modele utilizate in masurarea cantitativa a anumitor caracteristici detinute de catre sisteme informatice (asemeni „bit”-ului care descrie cantitatea minima de informatie ce poate fi transmisa intr-un canal de comunicatie, metricele ajuta in exprimarea propietatilor de care dispune un program, din punct de vedere al ingineria software). Astfel, se pot efectua masuratori obiective, cuantificabile, ce pot fi reproduse, in scopul planificarii bugetului alocat proiectului, optimizarii software-ului, testarii acestuia sau asignarea sarcinilor catre multiplele departamente implicate in dezvoltare. [1]

Privind lucrurile intr-un mod practic, metrica software este un model matematic, caracterizat de o serie de ecuatii si inecuatii, totodata prezentand o serie de functii, alcatuind astfel un mecanism ce ajuta in descrierea sisteumului in cauza. Astfel, ajuta in analizarea software-ului, concentrandu-se asupra masurarii unei anumite caracteristici, in timp ce ia in calcul si factorii ce o influenteaza, realizand asadar o analiza complexa ce poate fi modelata spre diverse scopuri (planificare, proiectarea testelor de mentenanta sau enduranta, ierarhizarea produselor software etc.). Drept urmare, putem afirma faptul ca aplicarea metricii software are in prim plan modelarea parametrului cercetat, prin minimizarea sau maximizarea acestuia, comparativ cu doua praguri:

$$f(X) < \alpha \text{ sau } f(X) > \beta$$

Unde f(X) este reprezentarea matematica a metricii, X factorii ce influenteaza parametrul, iar α si β praguri mentionate anterior. Totodata, functia f(X) poate fi utilizata si ca o restrictie, daca modelarea se face sub forma:

$$\beta < f(X) < \alpha$$

Astfel, realizandu-se o controlare eficienta a parametrului in cauza (aplicabilitate in optimizare, spre exemplu, daca utilizam o metoda prin care vrem sa obtinem un cod cat mai scurt, putem folosi o astfel de modelare matematica).

Asadar, metricile software ofera parametrii si indicatori ce ajuta in dezvoltarea programelor, prin obtinerea unor rezultate de performanta mult mai bune decat in lipsa acestora, atat in etapele incipiente, cat si in cele de finisare a produsului. De pilda, se poate efectua o serie de masuratori in faza de modelare a programului, ulterior acestea fiind re-utilizate si imbunatatie pentru o etapa de testare (spre exemplu) aplicata dupa aducerea software-ului intr-o faza finala.

Metricele ar trebui sa fie de incredere (termen tradus in engleza drept “reliable”), concentradu-se atat pe descrierea parametrilor, cat si prezicerea anumitor evenimente referitoare la acestea. Astfel, se impune ca o masuratoare sa fie evaluata din punct de vedere obiectiv, sa

prezinte un cost redus si o modalitate de implementare usoara (sub ideea ca o complexitate sporita poate duce la obtinerea unor rezultate eronate daca unul sau mai multi parametrii cheie sunt determinati gresit, marind astfel probabilitatea evaluarii defectuoase). Totodata, scopul oricarei masuratori, desi ar trebui sa fie binestabilit de dinainte, se bazeaza pe obtinerea unor rezultate ce doresc, principial, ilustrarea performantei, functionalitatii, timpului de raspuns, costului sau vitezei sistemului evaluat. Drept urmare, metricile software s-au bazat pe textul sursa, graful pe baza caruia a fost proiectat programul si comportamentul acestuia in timpul executiei (evaluand in mod continuu parametrii doriti si simtetizarea rezultatelor obtinute).

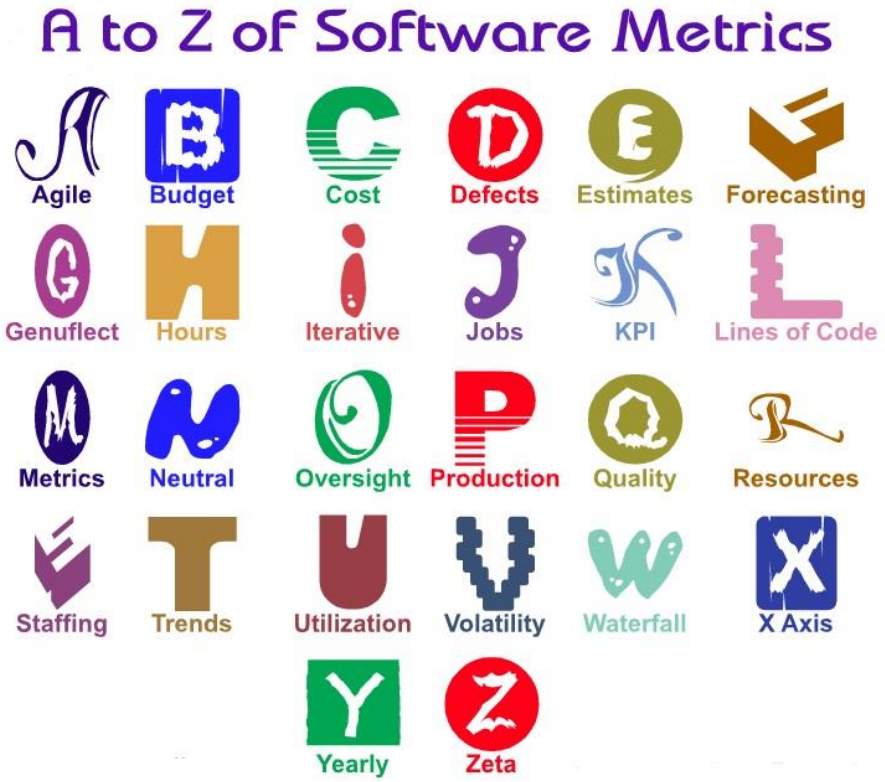
1.2 Functiile metricilor software

In continuare, vom discuta despre functiile indeplinite de catre metricele software, astfel incarcand sa ne apropiem de ideea “fiabilitatii” (termen ce poate fi inteles ca proces de prevenire, detectare si eliminare a erorilor de tip software). Astfel, in proiectarea unei metode de masurare, trebuie sa avem in vedere anumite functii pe care le urmareste aceasta, precum: [1]

- **Functia de masurare:** este obiectivul principal pe baza caruia au fost create metricele initial, avand la baza modelarea matematica descrisa anterior, urmarind cuatificarea caracteristicilor unui produs software. Totul pleaca de la parametrii simplu si impliciti ai programului, precum linii de cod sursa, numar de erori, durata executiei unui ciclu masina etc., acestea fiind utile atat in faza de proiectare a produsului, cat si ulterior in verificarea pe loturi compuse din programul initial.
- **Functia de comparare:** se refera la analiza software-ului din punct de vedere al compararii sale cu alte produse similare, aparute pe piata sau inca aflat in stagiu de dezvoltare, sau cu el insusi (sub ideea suportul teoretic comparativ cu programul practic). Aceasta comparare se realizeaza prin diferenta intre doua rezultate exprimate in unitati de masura similare.
- **Functia de analiza:** rezulta din necesitatea interpretarii rezultatelor obtinute in urma aplicarii metricilor software, astfel putand insusi produsului software atribute caractestice disciplinei de „Calitate si Asigurarea Fiabilitatii”, si anume, viteza de executie, fiabilitate, scalabilitate si cost.
- **Functia de sinteza:** face ca analiza produsului sa fie mai usoara din punct de vedere al lotului de esantioane cu care se doreste compararea, astfel incat, se aleg programe ale caror scop si insusiri seamana foarte mult cu cele ale software-ului in dezvoltare (spre exemplu, daca facem un program in Java, ar fi imposibila compararea sa cu toate celelalte existente sub acelasi limbaj de programare).
- **Functia de estimare:** se refera la caracteristica metricelor de a actiona precum niste indicatori statistici, acestia avand un prag de comparatie.

- **Functia de verificare:** metricele totodata sunt folosite si pentru a dovedi ipoteze statistice, spre exemplu, in cazul in care aproximam din punct de vedere teoretic faptul ca unu din o suta de produse software vor avea defecte de tipul logic.

Avand in vedere functiile caracteristice ale metricelor de tip software, putem continua in capitolul viitor cu prezentarea celor mai proeminente metode de masurare dezvoltate pe baza principiilor ilustrate anterior. Vom observa faptul ca fiecare metrica software deserveste un scop predefinit, rezultatele contribuind in diverse analize ale fazelor dezvoltarii software-ului.



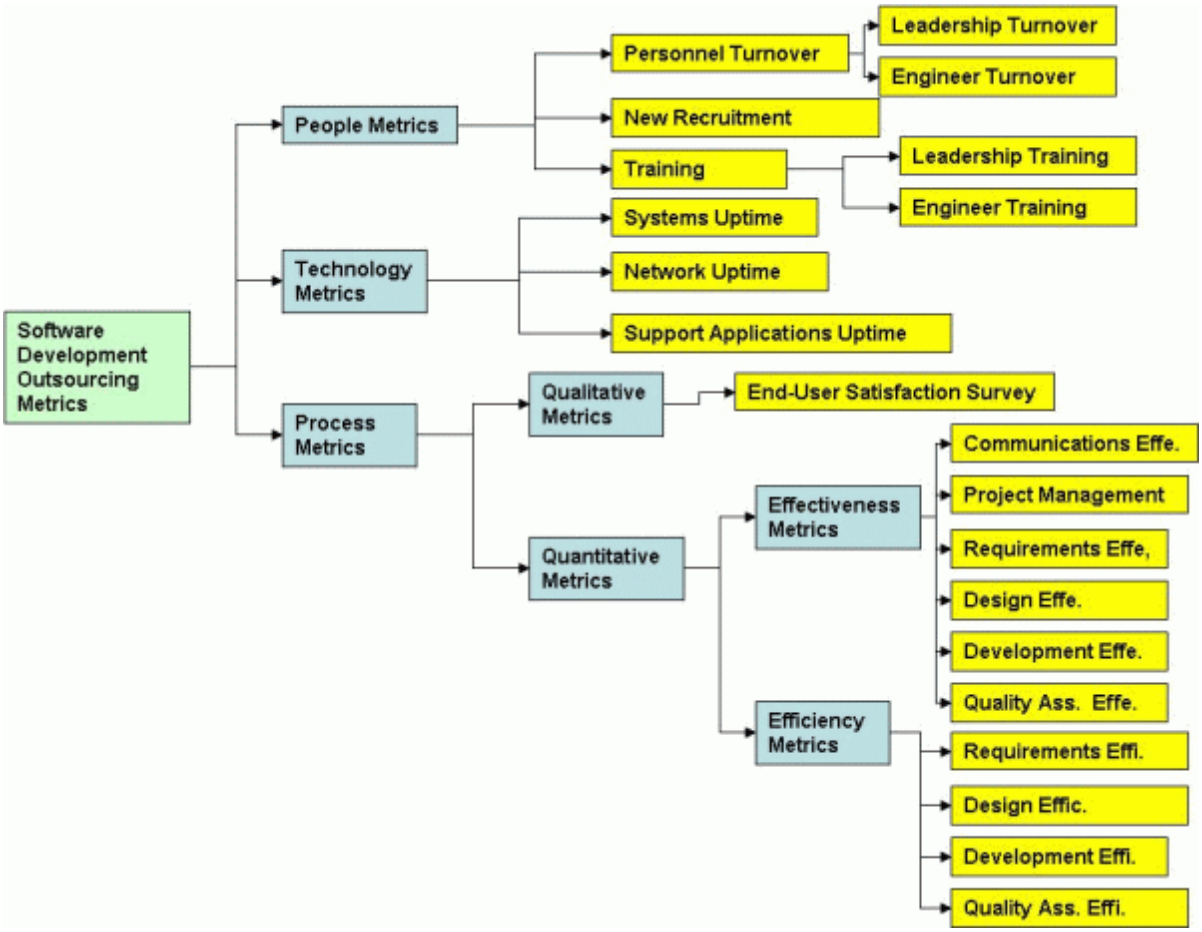
(Ilustrarea caracteristicilor urmarite prin aplicarea metricelor software)
(http://teamqualitypro.com/wp-content/uploads/b-AtoZ-of-software-metrics.png)

1.3 Metode de masurare a calitatii software

Metricile software pot fi de mai multe tipuri, in functie de parametrul in cauza, astfel existand cateva “metode de masura” (vom utiliza acest termen in evitarea repetitiei cuvantului “metrici”), dupa cum urmeaza: [2]

- **Strategia echilibrata** ("Balanced scorecard – BSC"): reprezinta o metoda de administrare si impartire a sarcinilor intre departamente si angajati de catre manageri pe baza metricilor aplicate asupra evolutiei prograumul pe care trebuie sa il dezvolte, astfel ajutand in decizionarea asupra unei strategii de lucru optime.
- **Erori per linie de cod** ("Bugs per line of code"): metode dezvoltate in evaluarea liniilor de cod, cu scopul gasirii erorilor de sintaxa, ce ulterior pot induce o logica defectuoasa a software-ului.
- **Caracteristica de testabilitate a codului** ("Code coverage"): o metoda de masurare a testelor ce pot fi aplicate asupra unui cod sursa, astfel incat o valoare mare a acesteia va duce la reducrea probabilitatii de existare a erorilor de sintaxa sau logica.
- **Consistenta** ("Cohesion"): masuratoare ce determina consistenta sistemului prin evaluarea apartenentei componentelor din care este alcatuit, vrand sa se determine daca acestea conlucreaza in parametrii optimi (spre exemplu: unui bloc de cod ce realizeaza conversia datelor din format analogic in digital nu va functiona optim daca ar avea un alt bloc alaturat ce numara granulele de nisip dintr-o cutie, tinzandu-se astfel spre o proiectare optima cu scop predeterminat).
- **Complexitatea ciclomatica** ("Cyclomatic complexity"): metrica ce ajuta in determinarea complexitatii unui software prin masurarea numarului de cai liniar independente din codul sursa (vom dezvolta acest aspect in capitolele viitoare).
- **Indexul de calitate al structurii** ("Design Structure Quality Index – DSQI"): metoda ce se axeaza pe arhitectura programului dezvoltat.
- **Complexitatea Halstead** ("Halstead complexity"): o abordare ce doreste ilustrarea implementarii si expresiei algorimilor in diferite limbaje, dar si independenta executiei acestora pe platforme specifice (vom dezvolta acest aspect in capitolele viitoare).
- **Sursa liniilor de cod** ("Source Lines of Code – SLOC"): cunoscut si sub numele de "Lines of Code – LOC", este o masuratoare destinata determinarii dimensiunii programului prin numararea liniilor din codul sursa (vom dezvolta acest aspect in capitolele viitoare).
- **Timpul de executie** ("Run time"): metrica ce se concentreaza pe modificare parametrilor in timpul rularii programului, precum timpul de compilare, de legatura dintre blocurile de cod sau de incarcare a acestora.
- **Masurarea Micro Functiilor** ("Weighted Micro Funtion Points" – WMFP): este o metoda moderna, inventata de "Logical Solutions", ce are in prim plan verificarea dimensiunii algoritmului folosit, fiind un succesor puternic al metricelor "COCOMO" (inventat de Barry Boehm) si "COSYSMO" (realizat de Richard Valerdi), combinand caracteristici ale metodelor complexe prezentate mai sus (Halstead si ciclomatica).

Din cadrul acestor metode, putem deduce faptul ca metricele software au fost dezvoltate de-a lungul timpului din necesitatea evaluarii produsului software din mai multe puncte de vedere. De pilda, metoda WMFP face referire la dimensiunile codului sursa, cel de detectare a erorilor se axeaza pe sintaxa si logica din spatele limbajului utilizat, iar consistenta este concentrata asupra conlucrarii si compatibilitatii blocurilor operationale din cadrul sistemului evaluat.



(Schema logica a aplicarii diferitelor metrici software asupra unui produs software)
(http://www.sourcingmag.com/wp-content/uploads/graphics/Framework_for_outsourced_software_development_metrics_and_examples.gif)

Drept urmare a celor discutate pana in momentul de fata, ne putem face o idee generala asupra importantei metricilor software in ingineria de dezvoltare a programelor. Asadar, vom dezvolta cateva din metodele ilustrate anterior in capitolele urmatoare, pentru a intelege si mai bine mecanismul din spatele metricilor software si influenta acestuia in obtinerea unui program cu o fiabilitate ridicata.

EVALUAREA FIABILITATII CU AJUTORUL METRICILOR SOFTWARE

2.1 Notiunea de fiabilitate

Dezvoltarea unui produs software presupune atat proiectarea acestuia cat si testarea capabilitatilor aferente. Astfel, utilizand notiunile dobandite in capitolul anterior despre metrici software, functiile urmarite de catre acestea si metode dezvoltate pe parcursul anilor, ne dorim in continuare a aprofunda cateva dintre aceste tipuri de masuratori, evidentiind totodata implicarea acestora in determinarea fiabilitatii unui program. Pentru inceput, se impune o definitie a termenului de “fiabilitate”, acesta ilustrand capacitatea unui sistem de a indeplini in mod corect functiunile prevazute, in conditii impuse de exploatare, pe parcursul unei durate de timp. Tehnic vorbind, este un aspect al calitatii, ce are in prim plan studiul defectiunilor (si combaterea acestora), aprecierea calitativa a sistemelor in timp, descoperirea tehnologiilor ce ajuta intr-o exploatare mai eficienta a sistemelor si analiza erorilor aparute pe parcursul functionarii.

Deoarece fiabilitatea poate fi studiata atat din punct de vedere calitativ, cat si cantitativ, metricii software au fost modelati astfel incat sa descrie modificarea parametrilor software-ului din ambele perspective. Asadar, in capitolele ce vor urma, dorim a prezenta cei mai proeminenti metrici software ce pot contribui in dezvoltarea eficace a unui produs software, pe mai multe planuri, precum lungimea liniilor de cod, aparitia erorilor de sintaxa si logice sau consistenta programului.

2.2 Complexitatea ciclomatica

Complexitatea ciclomatica (cunoscuta in limba engleza sub numele de “cyclomatic complexity”) este o metrica software inventata de catre Thomas McCabe Sr. in anul 1976, in scopul masurarii dificultatii de intelegere a programului cu ajutorul instructiunilor sale ce arata caile liniar independente ale codului sursa. Mecanismul din spatele unei astfel de metrici se bazeaza pe complexitatea grafului de control al fluxului de informatii, nodurile fiind vazute drept grupuri indivizibile de comenzi ale programului, iar arcele in diferite moduri (de pilda, un “arc” conecteaza doua noduri daca urmatoarea comanda poate fi executata imediat dupa terminarea celei dintai), astfel putand fi aplicata si asupra unui singur bloc operational de cod (o clasa, o functie sau o metoda). Asadar, putem afirma faptul ca instructiunile unui program sau modul al acestuia determina fiabilitatea si dificultatea in testare a programului sau a unui modul din cadrul acestuia.

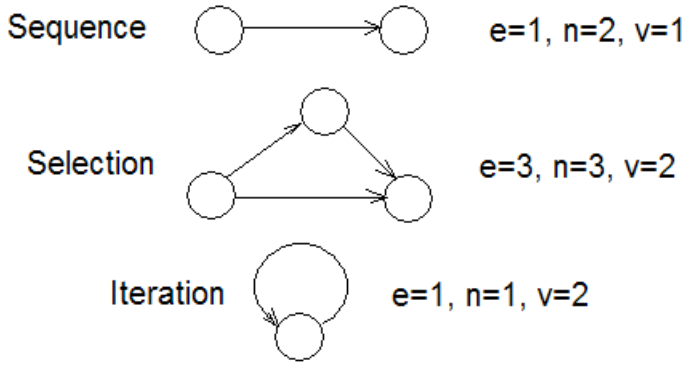
In prima faza, vom discuta despre efectul si metoda de functionare a acestei metrici asupra unui modul din program (instructiune simpla sau un ciclu de tipul “for”, “while” etc.),

ulterior discutand despre influenta acesteia asupra intregului software. Astfel, vom discuta despre **secvente** (blocuri indivizibile de instructiuni, ce se executa mereu in aceeasi ordine), **selectii** (conditiile blocurilor operationale) si **iteratii** (ciclurile secventelor). [4]

Complexitatea ciclomatica a unui modul din program, raportata la un graf de control al fluxului, este modelata de formula matematica:

$$V = e - n + 2$$

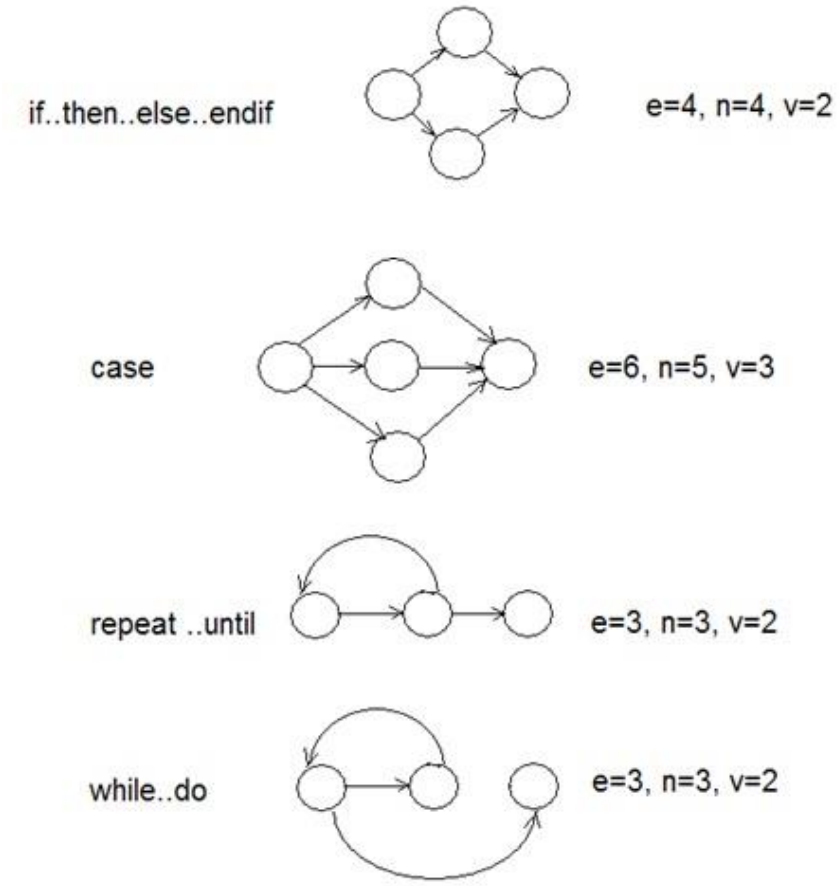
Unde "V" este notatia utilizata pentru o secventa, "e" reprezinta numarul de arce, iar "n" numarul de noduri ale grafului. Trebuie precizat faptul ca o simpla secventa, formata dintr-un arc si doua noduri, ar avea e=1, n=2 si v=1, pentru o instructiune decizionala ar avea e=3, n=3 si v=2, iar pentru o iteratie, ar fi e=1, n=1 si v=2.



(Grafuri de control pentru secventa, selectie si iteratie)

Daca avem o secventa, vom avea V=1, fiind necesara doar o executie de test pentru a parcurge fiecare instructiune din cadrul acesteia. Trebuie mentionat faptul ca orice salt la un alt modul va creste complexitatea acestuia cu 1, necesitand inca o executie de testare. Deci, putem afirma faptul ca metrica de complexitate ciclomatica pentru un modul returneaza numarul minim de executii a testului, in scopul acoperirii tuturor arcelor din graful orientat.

In cazul unor module de instructiuni compuse (de exemplu, un "if" cu doua cazuri), acestea trebuie despartite in mai multe decizii de tip simplu, precum cele prezentate in pictogramele de mai sus (if "decizia1" do "actiunea1", else if "decizia1" do "actiunea2"). Trebuie precizat faptul ca, in cadrul unei testari structurale, complexitatea ciclomatica drept rezultat numerit ar fi indicat sa nu depaseasca valoarea de 10, deoarece probabilitatea de aparitie a erorilor creste exponential o data cu depasirea acestui prag impus (factor ce influenteaza fiabilitatea programului, aceasta metrica prevenind greselile de tip logic). De asemenea, in cadrul proiectarii detaliate a modulelor (adaugarea de cazuri pentru o structura "do while", de pilda), limitarea nu trebuie sa depaseasca 7 ca si rezultat al complexitatii, deoarece exista din nou sansa aparitiei de erori nedorite, astfel incat, acestea trebuie reproiectate.



(Exemple de grafuri orientate pentru diverse module)
(https://geekdetected.files.wordpress.com/2013/03/untitled.jpg)

Complexitatea ciclomatica a proiectarii unui modul consta in masurarea efectului individual al fiecarui modul in parte asupra programului dezvoltat. Se bazeaza tot pe principiul grafurilor orientate, insa intervine problema comunicarii intre module, fiind necesara astfel si marcarea nodurilor ce contin apeluri catre modulele externe. Asadar, nodurile ce fac legatura intre module vor fi marcate, urmand apoi simplificarea fiecarui graf de control in parte dupa un algoritm bine stabilit: in primul rand, nodurile marcate nu pot fi eliminate, apoi se elimina nodurile nemarcate ce nu contin decizii, totodata eliminandu-se si arcele care intorc secventa la inceputul unui ciclu (si nu contine noduri marcate). Drept urmare, se obtine complexitatea grafului redus, dupa o formula asemanatoare celei anterioare:

$$IV = e - n + 2$$

Se observa faptul ca, in proiectarea modulului, complexitatea ciclomatica ignora arcele acoperite de catre testare ce nu contin legaturi catre module externe.

In continuare, putem defini **complexitatea ciclomatica totala a unui program**, cunoscand notiunile discutate pana la momentul actual, ce insumeaza toate ciclomaticele modulelor aferente acestuia, dupa formula:

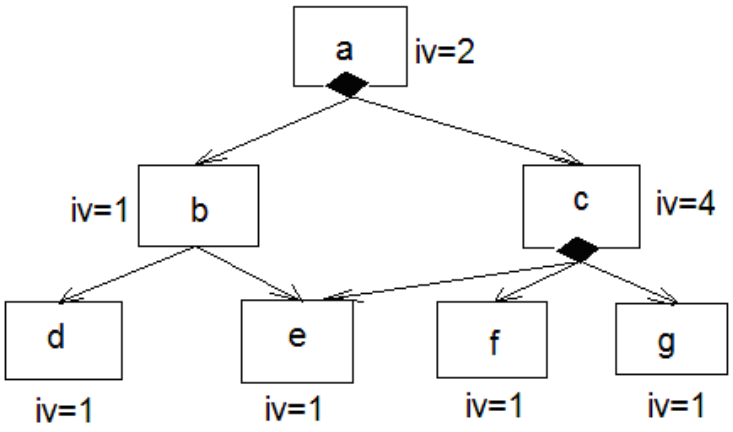
$$V = e - n + 2p$$

Unde "p" este numarul de module, "e" numarul de arce din toate modulele, iar "n" numarul de noduri al tuturor modulelor. Din punct de vedere al fiabilitatii, poate fi la indemana proiectantului sa realizeze un singur modul imens, alcatuit la baza din toate modulele aferente programului, deoarece, conform formulei de mai sus, ar rezulta o complexitate ciclomatica totala mai mica, insa lucrul cu module separate este utila in mentenanta logica a software-ului, chiar daca are ca repercusiune crestea complexitatii totale. [4]

Complexitatea ciclomatica a integrarii este un alt aspect al metricii prezentate in acest capitol, aceasta facand referire la un ansamblu de module, modelandu-se cu ajutorul formulei:

$$S_1 = S_0 - N + 1$$

Unde "S_0" reprezinta suma tuturor complexitatilor ciclomatice ale fiecarui modul in parte, iar "N" numarul de module. Aceasta metoda functioneaza pe principiul diagramei de structura, astfel ca, un program ce nu prezinta ramificatii va avea complexitatea egala cu 1.



S0=11, S1=11-7+1=5

(Diagrama de structura a unui program si calcularea complexitatii ciclomatice a integrarii)

Se impune explicarea diagramei de mai sus, dreptunghiurile fiind modulele, iar romburile componente decizionale (fie modulul b, fie modulul c, spre exemplu).

1.3 Complexitatea Halstead

Complexitatea Halstead este o metrica software de o importanta deosebita, introdusa de catre Maurice Howard Halstead in anul 1977 drept o parte din contributia sa in stabilirea unui model empiric in ingineria software. Aceasta metoda reflecta ideea ca masuratorile ar trebui sa reflecta implementarea si expresia algoritmilor in diferite limbaje de programare, insa rezultatele ar trebui sa fie invariante platformei pe care acestea sunt executate. Astfel, s-a dorit identificarea propietatilor masurabile si a relatiilor dintre acestea (similar relatiei dintre volum, masa si presiunea gazului, cunoscute din fizica clasica), ceea ce face ca aceasta metrica sa nu fie neaparat "complexa" (insa vom pastra denumirea, dupa voia cercetatorului). [5]

Metrica lui Halstead se bazeaza pe un model matematic invariant limbajului de programare sau a platformei pe care acesta ruleaza, incercand sa modele volumul programului scris, dificultatea de intelegere, efortul depus de catre inginer, timpul necesar scrierii si numarul de erori aparute. Astfel, avem dupa cum urmeaza:

n_1 = numarul de operatori diferiti
n_2 = numarul de operanzi diferiti
N_1 = numarul total de operatori
N_2 = numarul total de operanzi

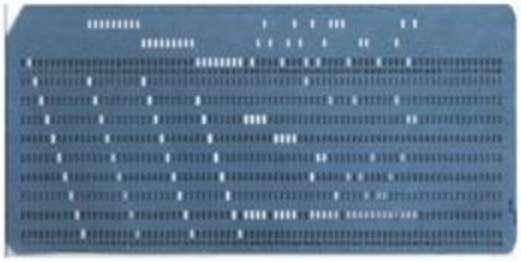
Asadar, pe baza acestor termeni putem efectua calcule, in determinarea urmatorilor parametrii:

- Vocabularul programului: $n = n_1 + n_2$
- Lungimea programului: $N = N_1 + N_2$
- Lungimea calculata a programului: $N = n_1 \log_2 n_1 + n_2 \log_2 n_2$
- Volumul: $V = N \times \log_2 n$
- Dificultatea programului: $D = \frac{n_1}{2} \cdot \frac{N_2}{n_2}$
- Efortul depus: $E = D \cdot V$
- Timpul necesar scrierii: $T = \frac{E}{18}$ secunde
- Numarul de erori aparute: $B = \frac{E^{\frac{2}{3}}}{3000}$

Drept urmare, se poate observa caracteristica specifica a acestei metrici si influenta sa asupra fiabilitatii unui produs software, fiind mai mult un instrument de calcul statistic, ce poate fi utilizat inaintea efectuarii masuratorilor efective si luat drept model teoretic de referinta.

2.4 Metrica liniilor de cod (“Lines of Code”)

“Lines of Code” (abreviat “LOC”), cunoscut deseori si sub numele de “Source Lines of Code” (prescurtat “SLOC”) este o metrica software ce are ca scop estimarea productivitatii din punct de vedere al programarii si asigurarea mentenantei software-ului, prin numararea liniilor de cod ale unui unui acestuia. Acesta a luat nastere pe vremea cand limbajele de programare cele mai raspandite erau FORTRAN si Assembler-ul, incercand sa preia modelul sistemelor de calcul ce functionau pe baza cartelelor perforate (cunoscute sub numele de “Punched Cards” sau “IBM cards”, fiind dezvoltate initial de catre colosul industrial mentionat). Ideea consta in faptul ca, o astfel de cartela reprezenta o linie de cod, ceea ce facea usoara indexarea si numerotarea acestora, principiu ce a dezvoltat necesitatea unei astfel de metrici. Eficienta acestei masuratori isi face simtita prezenta atunci cand dorim a compara software-uri de 50.000 de linii cu altele de 200.000 de linii, nefiind foarte folositor in comparatia numerelor cu discrepante neglijabile, deoarece valoarea acestei masuratori, mai departe, poate fi transformata in diversi parametrii cu insemnatate proprie (de exemplu, poate fi tradus drept “man-hours”, adica munca depusa de un om intr-o ora).



(Cartela perforata – “Punched Card”)

(http://upload.wikimedia.org/wikipedia/commons/4/4c/Blue-punch-card-front-horiz.png)

Metrica “LOC” este alcatuita din doua componente: SLOC de tip fizic (referit simplu drept “LOC”) si SLOC de tip logic (referit ca “LLOC”, prin adaugarea cuvantului “logical”). Cel dintai tip este o implementare ce numara fiecare linie de text a programului sursa, inclusiv cele de comentariu si liniile goale lasate intentionat drept delimitare a blocurilor operationale, ce se iau in considerare doar daca depasesc un procentaj de 25% din totalitatea programului. LLOC-ul pe de alta parte, are in vedere numararea cuvintelor cheie caracteristic fiecarui limbaj (“for” si “while” pentru C/C++ sau “loop” pentru Assembler, de pilda). Pentru o ilustrare mai buna asupra modului de functionare, vom utiliza urmatorul exemplu: [6]

```
for (i=0; i<10; i++) printf(“Inginerie Software”); /*Comentariu*/
```

In cadrul acestui exemplu putem vedea faptul ca avem o linie de LOC, doua linii de LLOC si o linie de comentariu. Insa, putem obtine si un alt rezultat daca exemplul este modificat:

```
/*Comentariu*/
for (i=0; i<10; i++)
{
printf(“Inginerie Software”);
}
```

Putem observa acum faptul ca avem cinci linii de LOC, doua linii de LLOC si una de comentariu, astfel incat se poate deduce ca obiectivitatea programatorului de a aseza liniile de cod in pagina pot afecta in mare parte doar LOC-ul de nivel fizic.

Desi, o metrica aparent utila si intuitiv de utilizat, "Lines of Code" a starnit controverse in randul programatorilor de pretutindeni, adesea considerandu-se faptul ca se investeste prea mult timp si resurse intr-o asemenea masuratoare, deoarece nu prezinta uniformizare a valorilor obtinute, neputand fii astfel comparate programe scrise in limbaje diferite. Totodata, LOC-ul este o metoda ineficienta de a monitoriza progresul angajatilor din cadrul unei firme de programare, deoarece un program scris in o suta de linii poate fi optimizat intr-o masura mult mai redusa, iar astfel s-ar crea discrepante uriase intre eforturile depuse de catre acestia (asemeni declaratiei lui Steve Ballmer, directorul executiv al Microsoft Corporation, adresata companiei rivale IBM: "Daca un LOC mai mare inseamna mai multi bani, insa noi avem un software de dimensiuni restranse, asta inseamna ca trebuie sa avem un profit mai mic decat al vostru?"). De asemenea, programatorii fata experienta pot recurge la repetarea blocurilor de cod, astfel rezultand intr-o valoare imensa a acestei metrici, insa pot creste considerabil timpul de executie (din nou, la o diferenta sesizabila de linii) si probabilitatea aparitiei erorilor de sintaxa. [6]

C

```c
# include <stdio.h>

int main() {
    printf("\nHello world\n");
}
```

Lines of code: 4
(excluding whitespace)

COBOL

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLOWORLD.
000300
000400*
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400     DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500     DISPLAY "Hello world!" LINE 15 POSITION 10.
100600     STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800     EXIT.
```

Lines of code: 17
(excluding whitespace)

(Comparatie a programul "Hello world!" scris in limbajele C si COBOL)
(http://en.wikipedia.org/wiki/Source_lines_of_code)

In consecinta, aceasta metrica software nu are foarte mare aplicabilitate, insa ar putea ajuta in reducerea timpului de executie a programului si optimizarea stocarii datelor pe suportul fizic, ceea ce ar duce la o fiabilitate sporita a programului in cauza.

1.5 Metrica WMFP (Weighted Micro Function Points)

Metrica "Weighted Micro Function Points" (abreviata "WMFP") este o metoda moderna de masurare a dimensiunii software-ului dezvoltat, fiind inventata in anul 2009 de catre compania Logical Solutions, pe baza algoritmilor clasici precum COCOMO, COSYSMO si cele doua complexitati discutate in capitolele anterioare. Spre deosebire de predecesorul sau in domeniu (metrica SLOC), aceasta metoda de masurare accepta o pregatire slab academica din partea utilizatorului, impartind codul sursa in mai multe "micro functii" a caror valoare este cuantificata si interpolata in mod dinamic, oferind asadar o acuratete sporita in estimare. [7]

WFMP-ul este alcatuit dintr-o serie de metrice cu o dimensiune redusa, ceea ce face ca rezultatul final sa fie o combinatie a datelor obtinute din acestea. Practic, valoarea returnata de catre aceste "micro metrici" (daca le putem numi astfel) sunt deduse prin aplicarea algoritmilor specifici WFMP asupra codului sursa, fiind ulterior translatate in timp:

- **Complexitatea fluxului** ("Flow complexity" – FC): masoara complexitatea programului utilizand caile de control a fluxului de date dintr-un program, asemeni metodei ciclomatice discutata in capitolele anterioare, insa prezinta o imbunatatire printr-o precizie ridicata obtinuta din introducerea unor relatii intre calculele efectuate.
- **Vocabularul de obiecte** ("Object vocabulary" – OV): masoara cantitatea unica de informatie continuta de catre programul in cauza, asemeni complexitatii Halstead, insa utilizeaza un limbaj de tip dinamic.
- **Crearea de obiecte** ("Object conjuration" – OC): estimeaza cantitatea de resurse utilizate de catre programul sursa in crearea si utilizarea obiectelor.
- **Intrigare aritmetica** ("Arithmetic intricracy" – AI): masoara complexitatea calculelor aritmetice realizate de catre software.
- **Transferul de date** ("Data transfer" – DT): cuantifica manipularea structurilor de date din cadrul programului.
- **Structura codului** ("Code structure" – CS): masoara efortul depus in scrierea codului sursa.
- **Comentarii** ("Comments" – CM): masoara efortul depus in scrierea comentariilor din program.
- **Date de initiere** ("Initiate data" – ID): estimeaza efortul depus in urma imbricarii datelor pe un suport fizic.

De asemenea, trebuie impus faptul ca, algoritmul WFMP este compus din trei stagii, primul fiind analizarea functiei, apoi transformarea de tip "APPW", ulterior facandu-se translatia in domeniul timp, plecandu-se de la formula de baza $\Sigma(W_i \cdot N_I)^{D \cdot q}$, unde M reprezinta sursa valorii metircelor masurate in stadiul incipient, W ponderea adjustata in realizarea modelului APPW, D costul iar q costul per iteratie.

CONCLUZII

In incheiere, putem conclude faptul ca metricele de tip software joaca un rol foarte important in dezvoltarea unui program, ajutand in asigurarea mentenantei acestuia, contribuind la realizarea fiabilitatii optime si a costului redus, prin ideea lor generala de a cuantifica si exprima numeric parametrii aferenti, spre a putea fi procesati si interpretati ultetior. Astfel, prin intermediul metricelor prezentate in lucrarea noastra, am avut in vedere demonstrarea evaluarii fiabilitatii unui produs software prin aportul adus de catre notiunea abstracta de “metrica”, vazand asadar cum parametrii simpli (precum numararea liniilor de cod) pot conduce la o intreaga analiza asupra complexitatii procesului de dezvoltare. Totodata, putem sustine faptul ca metricele software ajuta si in crearea modelelor teoretice, ce pot fi utilizate drept sablon de comparatie pentru rezultatul final, astfel sporind imbunatatirea acestuia in versiunile viitoare, astfel contribuind deschizand noi oportunitati catre obtinerea unei fiabilitati cat mai ridicate.

BIBLIOGRAFIE

1. http://www.software-metrics.ase.ro/articole/METRICI%20%20SOFTWARE.htm
2. "Integration Watch: Using metrics effectively", Binstock Andrew
3. Joan C. Miller, Clifford J. Maloney (February 1963). "Systematic mistake analysis of digital computer programs". Communications of the ACM
4. McCabe (December 1976). "A Complexity Measure". IEEE Transactions on Software Engineering: 308–320
5. Halstead, Maurice H. (1977). Elements of Software Science
6. IEEE Standard for a Software Quality Metrics Methodology
7. Jones, C. and Bonsignour O. The Economics of Software Quality