

TEHNIKE MERENJA VELIČINE SOFTVERA

SOFTWARE SIZE MEASUREMENT TECHNIQUES

Jovan Popović
Gowi d.o.o.

Sadržaj – Ovaj rad predstavlja pregled tehnika koje se mogu koristiti radi merenja veličine softverskih projekata. Tehnike merenja veličine su osnova za procese analize i upravljanja projektima zato što daju objektivn uvid u stanje i veličinu projekta. Zbog toga je neophodno da članovi tima budu svesni tehnika merenja koje im mogu pomoći da donose bolje odluke tokom razvoja softvera.

Abstract – This paper gives overview of techniques that can be used to determine size of the software projects. These techniques are basis for process of analysis and management because they give objective insight into project status and size. Therefore it is crucial for team members to be aware of available measurement methods which can help them in better decision making during software development.

1. UVOD

Merenje veličine softvera je izuzetno značajna aktivnost u procesu razvoja softvera. Merenjem softvera se dobijaju informacije o veličini sistema koji je potrebno izgraditi, što je izuzetno važna informacija u procenama vremena potrebnog za izradu, cene i planiranje razvoja softvera.

Trenutno se u velikom broju softverskih organizacija procene vremena ne rade na osnovu rezultata merenja, već pretežno na osnovu iskustva ili poređenja sa sličnim projektima. Čest metod za procenu je dekompozicija sistema na manje delove čije se veličine lakše mogu proceniti pa se veličina sistema dobija kao suma veličina delova ili se nekom metodom kolektivne procene (npr. Delfi metodom) dolazi do procenjene veličine softvera. Ovakve empirijske metode, iako mogu biti veoma efikasne, nemaju objektivnu pozadinu i lako mogu dovesti do pogrešnih procena i zaključaka.

Iako su svesne problema sa empirijskim metodama i prednosti procena na osnovu mernih rezultata sistema, softverske organizacije se i dalje drže svojih metoda – uglavnom zbog nedovoljnog poznavanja praktične primene metoda merenja što često izaziva strah od greške. Pored toga, softverskim organizacijama je često teško da među brojnim metodama merenja odaberu jednu koja bi im najviše odgovarala.

Da bi se ovi problemi prevazišli, softverske organizacije moraju biti upoznate sa glavnim metodama merenja kako bi mogle da odaberu one koje im najviše odgovaraju. Cilj ovog rada je prikazivanje ti osnovnih metoda merenja u softveru.

Rad je organizovan na sledeći način: Sekcija 2 daje teorijsku osnovu merenja, Sekcije od 3 do 7 daju pregled najkorišćenijih metoda merenja, dok je u sekciji 8 dat zaključak.

2. DEFINICIJA MERA

Neka je (S, \geq) uređeni par gde je S proizvoljan skup objekata i \geq binarna tranzitivna relacija nad elementima skupa S . Relacija \geq predstavlja operator poređenja elemenata skupa S .

Neka je funkcija $f: S \rightarrow \mathbb{R}^+$ koja preslikava objekte iz skupa S u skup nenegativnih realnih brojeva takva da za svako s_1 i s_2 koji pripadaju skupu S važi:

$$\forall (s_1, s_2 \in S)(s_1 \geq s_2 \Leftrightarrow f(s_1) \geq f(s_2)) \quad (1)$$

Funkciju f nazivamo mernom funkcijom skupa S a brojeve u koje preslikava elemente skupa S merama objekata. Iz definicije funkcije f se može primetiti da ona preslikava objekte iz skupa S u odgovarajuće realne brojeve tako da ako je jedan objekat „veći“ od drugog u skupu S , tada će i mera prvog objekta biti veća od mere drugog objekta u skupu realnih brojeva.

Merna funkcija je značajna zato što omogućava preslikavanje skupa realnih podataka na apstraktni matematički model nad kojim se može vršiti dalja analiza postojećim matematičkim tehnikama.

U firmama koje se bave razvojem softvera postoje istorijski podaci o projektima koji su rađeni i može se odrediti neki odnos među projektima tako da se utvrdi međusobni poredak projekata po veličini. Odnos veličine projekata se može definisati uvidom u vreme potrebno za završetak projekta, ukupan broj linija koda u projektu ili na bilo koji drugi način koji može uporediti dva projekta.

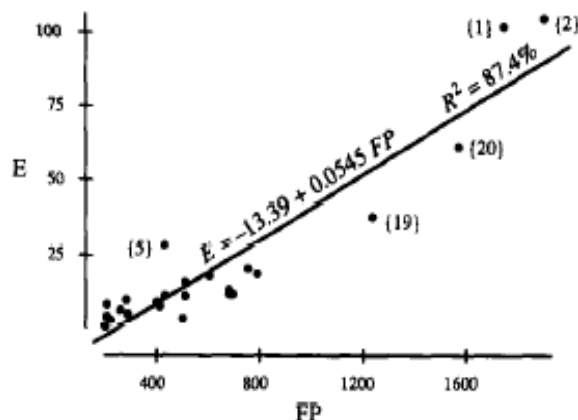
Na ovaj način skup S postaje skup projekata softverske organizacije dok relacija \geq postaje relacija „većih“ projekata. U realnosti je teško odrediti funkciju koja će projekte direktno preslikati u skup realnih brojeva, tako da se funkcija mere određuje indirektno. Projekti se predstavljaju modelima koji predstavljaju apstrakciju samih projekata i nad tim modelima se definišu funkcije mere. Preciznije definicije se mogu naći u [1].

U sledećim sekcijama su predstavljene metode koje se najčešće koriste za merenje veličine projekata na osnovu modela koji reprezentuju projekat.

3. FUNKCIONALNE TAČKE

Merenje veličine projekata metodom funkcionalnih tačaka (FP) se zasniva na posmatranju funkcionalnog modela sistema [2]. Funkcionalni model sistema opisuje funkcije koje sistem pruža korisniku kao i podatke koji se koriste u sistemu. Verovatno najpogodniji model za opisivanje funkcionalnog modela je metod dijagrama toka kontrole i dijagrama toka podataka. Ovakvim modelom se opisuje koji su tokovi komunikacije između korisnika, eksternih sistema i sistema koji se meri. Tokovi se modeluju tako da se lako može utvrditi odakle je tok iniciran, sa kojim procesima komunicira, koje podatke sadrži i koje podatke unutar sistema koristi.

Prvi pokušaj definisanja metrike kojom bi se iskazala veličina sistema na osnovu funkcionalnog modela je izveo A. Albreht osamdesetih godina prošlog veka [3]. On je prikupio podatke o 24 projekata rađenih u IBM-u, modelovao njihove funkcionalnosti, merio veličinu funkcionalnog modela i pokušao da nađe funkcionalnu zavisnost između mere veličine sistema i vremena potrošenog na izradu. Slika 1 prikazuje rezultate njegove analize.

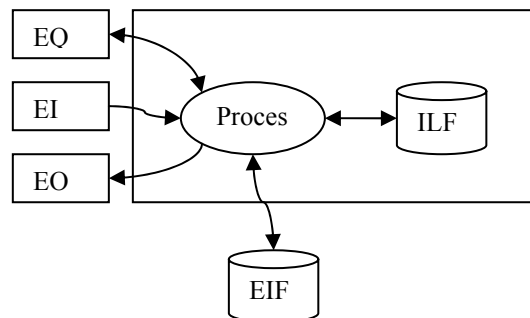


Slika 1. Zavisnost vremena potrebnog za izradu projekta od veličine iskazane u funkcionalnim tačkama

Albrehtova metoda je idejno bila ispravna ali usled relativno malog broja projekata kojima je raspolagao nije mogao da dobije preterano dobre rezultate. Njegova ideja je dalje unapređivana i validirana na osnovu velikog broja podataka o projektima iz celog sveta.

Danas, internacionalna grupa IFPUG održava i unapređuje FP metod merenja. IFPUG se sastoji od više grupa podeljenih po zemljama kao što su Engleska, Brazil, Australija i koje prikupljaju i analiziraju podatke o softverskim projektima u svojim zemljama, i konstantno unapređuju FP standard. Trenutno je FP standard toliko unapređen da je definisan i ISO standard [4] kojim se definiše proces merenja metodom funkcionalnih tačaka.

Slika 2 prikazuje meta-model funkcionalnog modela sistema koji se koristi prilikom merenja FP metodom. Ovaj model prikazuje osnovne elemente koji se koriste pri analizi funkcionalnog modela.



Slika 2. Meta-model funkcionalnih tačaka

Po FP meta-modelu [2] korisnici i eksterni sistemi komuniciraju sa procesima unutar sistema. Sa procesima unutar sistema se komunicira preko transakcija koje mogu biti:

1. Ulazi – (EI external input) kojima podaci ulaze u sistem,
2. Izlazi – (EO external output) kojima podaci izlaze iz sistema,
3. Upiti (EQ external inquiry) kojima se za neki ulaz direktno vraća odgovor.

Sve funkcionalnosti u sistemu se identifikuju i klasifikuju na tri navedene vrste transakcija. Podaci sa kojima rade procesi se mogu nalaziti ili u sistemu ili van sistema tako da se definišu dve vrste podataka (fajlova u FP terminologiji):

1. Interni podaci (ILF Internal Logical File) koji se nalaze unutar sistema i tu se održavaju,
2. Eksterni podaci (EIF External Interface File) koji se nalaze u drugim sistemima ali ih sistem koji se posmatra na neki način koristi.

Metoda merenja se zasniva na analizi transakcija i podataka i dodeljivanju klasa kompleksnosti (jednostavna, srednja, kompleksna) svakoj od njih. Kompleksnost transakcija se određuje na osnovu broja podataka koji ulaze/izlaze preko transakcije i broja fajlova koji se koriste. Kompleksnost podataka (fajlova) se određuje na osnovu broja prostih i složenih tipova podataka u okviru fajlova. Svakoj transakciji odnosno fajlu se u zavisnosti od pridružene kompleksnosti dodeljuje težina. Ukupan zbir težina svih transakcija i fajlova predstavlja meru veličine sistema izraženu u neprilagođenim funkcionalnim tačkama (UFP unadjusted function point).

Metod funkcionalnih tačaka je opšte prihvaćena tehnika za merenje veličine softvera koja je čak i standardizovana po ISO/IEC standardu[4]. Pored standardnog FP metoda danas postoji niz varijacija koje održavaju druge grupe kao što su NESMA, Mark II, Cosmic-FPP, Feature point i sl. Neki od ovih metoda su kao i originalna FP metoda postali deo ISO standarda.

Iako se danas u svetu objektno orijentisanih metoda modelovanja softvera tradicionalni funkcionalni model

više ne koristi u toj meri, metoda funkcionalnih tačaka ostaje jedna od najvažnijih metoda merenja veličine softvera zbog ubedljivo najvećeg broja projekata nad kojim je izvršena analiza. Ostale mlađe metode iako se mogu lakše primeniti nemaju dovoljno eksperimentalnih podataka kojima bi dostigle nivo validnosti metode funkcionalnih tačaka.

4. PRIMENA FP ANALIZE NA UML MODELE

Veliki broj metoda je predložen kako bi se prevazišao jaz između metode funkcionalnih tačaka i objektno orijentisanih metoda modelovanja. S jedne strane metoda funkcionalnih tačaka daje dobre rezultate u procesu određivanja veličine sistema koji uslovljavaju veoma pouzdane funkcionalne zavisnosti između veličine sistema i vremena potrebnog za izradu, ali se zasnivaju na prevaziđenim funkcionalnim modelima. S druge strane objektno orijentisane metodologije se široko primenjuju i omogućavaju uvid u niz objektnih modela koji vernije prikazuju sistem od funkcionalnog modela, ali ne postoji metoda koja dovoljno pouzdano preslikava objektnu modele u realnu veličinu sistema.

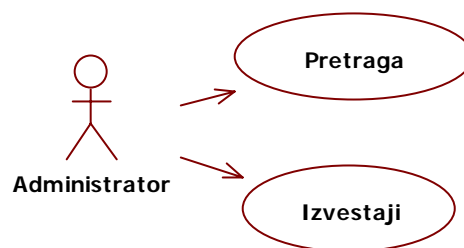
Neki autori su predlagali metode kojima bi se integrisala znanja iz metode funkcionalnih tačaka i primenila direktno na UML modele umesto na funkcionalne modele. Jedna od najboljih metoda mapiranja OO modela na FP pravila je predložena od strane Tomasa Fečkea [5]. Fečke je definisao pravila po kojima se UML koncepti (korisnici, slučajevi upotrebe, klase i relacije među njima) mapiraju na FP koncepte (fajlove i transakcije). Fečke nije definisao pravila za određivanje veličine softvera na osnovu UML elemenata pošto se transformacijom UML elemenata u fajlove i transakcije na njih mogu primeniti već definisana pravila za određivanje veličine softvera. Fečke je definisao 15 pravila za mapiranje OO modela na funkcionalni model:

1. Pravila za određivanje korisnika sistema – tri pravila kojima se UML korisnici sistema mapiraju na FP korisnike i eksterne sisteme,
2. Pravila za određivanje transakcija – tri pravila definišu kako se na osnovu slučajeva upotrebe mogu identifikovati transakcije po FP metodi,
3. Pravila određivanje fajlova – dva pravila definišu kako se na osnovu objektnog modela identifikuju FP fajlovi,
4. Pravila za određivanje podataka i zapisa fajlova – četiri pravila definišu kako se na osnovu atributa i relacija klasa iz modela klasa identifikuju FP podaci i zapisi fajlova,
5. Pravila za određivanje podataka i referenciranih fajlova transakcija – tri pravila definišu kako se na osnovu modela slučajeva upotrebe i klasa određuju podaci i referencirani fajlovi transakcija.

Primenom ovih 15 pravila na osnovu OO modela se dobija ekvivalentni FP model na koji se mogu primeniti standardna pravila i odrediti veličina sistema.

5. TAČKE SLUČAJEVA UPOTREBE

Razvoj UML metodologije u savremenim softverskim projektima naveo je veliki broj istraživača da pokušaju da razviju metodologiju određivanja veličine projekata na osnovu dokumenata i modela koji se koriste prilikom kreiranja specifikacije. U savremenom objektno orijentisanom softveru model slučajeva upotrebe je standard za opisivanje korisničkih zahteva. Model slučajeva upotrebe prikazuje korisnike sistema i funkcionalnosti koje vrše u okviru sistema. Slika 3 prikazuje primer dijagrama slučajeva upotrebe nekog sistema.



Slika 3. Primer dijagrama slučajeva upotrebe

Na dijagramu slučajeva upotrebe se mogu primetiti korisnici, funkcionalnosti (slučajevi upotrebe) i eksterni sistemi sa kojima se komunicira.

Model slučajeva upotrebe je osnova za metodu merenja veličine softvera metodom tačaka slučajeva upotrebe [6,7]. Ovom metodom se analizira specifikacija interakcije između korisnika sistema i samog sistema.

Podaci koji se koriste pri merenju se nalaze u dokumentaciji korišćenoj za slučajeve upotrebe (uglavnom scenarija slučajeva upotrebe). Karakteristike sistema od značaja za određivanje kompleksnosti sistema metodom tačaka slučajeva upotrebe su korisnici sistema i slučajevi upotrebe.

Korisnici sistema se klasifikuju na sledeće tri kategorije:

1. Jednostavni korisnici su eksterni sistemi koji komuniciraju sa sistemom po definisanom protokolu,
2. Srednji korisnici su eksterni sistemi koji ne komuniciraju po predefinisanim protokolima nego po nižem protokolu kao što je TCP/IP, RMI ili ljudi koji koriste sistem preko jednostavnog interfejsa (npr. telnet ili komandni prompt),
3. Složeni korisnici koji predstavljaju ljude koji koriste sistem preko grafičkog interfejsa.

Radi klasifikacije slučajeva upotrebe potrebno je analizirati scenarije slučajeva upotrebe kako bi se identifikovale elementarne aktivnosti u njima. Slučajevi upotrebe se klasifikuju u tri kategorije:

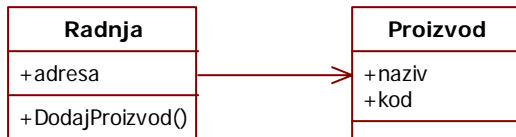
1. Jednostavni koji imaju 3 ili manje transakcija,
2. Srednji koji imaju od 4 do 7 transakcija,
3. Komplikovani koji imaju više od 8 transakcija.

Težine dodeljene korisnicima sistema i slučajevima upotrebe se sabiraju da bi se dobila inicijalna mera sistema – broj nepodešenih tačaka slučajeva upotrebe (*Unadjusted Use Case Points – UUCP*). U ovu meru sistema ne ulaze dodatne informacije o tehničkim karakteristikama i faktorima okruženja. Tehnički faktori mogu biti informacije o tome da li je sistem distribuiran, kolika je složenost obrade podataka, prenosivost na druge platforme i slično.

Tehnika tačaka slučajeva upotrebe predstavlja značajnu metodu koja daje dobre rezultate u merenju. Ova metoda zadržava logiku merenja funkcionalnih tačaka po kojoj se sistem procenjuje na osnovu karakteristika vidljivih krajnjem korisniku a ne na osnovu tehničke složenosti realizacije sistema. Jedina mana ove metode leži u činjenici da se uopšte ne uzimaju u obzir veličine struktura podataka u sistemu pošto se ove informacije ne mogu naći u slučajevima upotrebe. Ipak ova metoda može biti korisna u kombinaciji sa drugim metodama ili u slučajevima kada se na osnovu funkcionalnosti može proceniti složenost struktura podataka u sistemu.

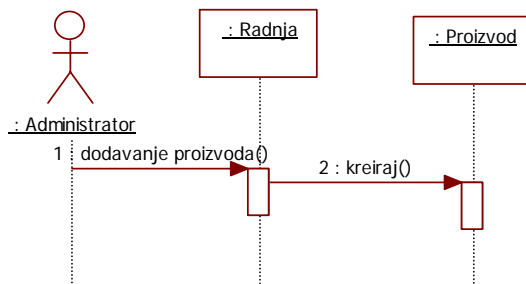
6. KLASNE TAČKE

Klasne tačke [8] predstavljaju metodu merenja sistema na osnovu modela klasa sistema i dijagrama sekvenca. Primer klasnog dijagrama je dat na slici 4.



Slika 4. Primer klasnog dijagrama

Dijagram klasa sistema predstavlja statičku sliku koja prikazuje klase, njihove atribute i relacije sa drugim klasama. Dinamička slika interakcije klasa se prikazuje preko dijagrama sekvence kojima se prikazuje kako klase pozivaju jedna drugu i u kom redosledu. Primer takvog dijagrama sekvence je prikazan na slici 5.



Slika 5. Primer dijagrama sekvence

Određivanja složenosti sistema se dobija određivanjem složenosti klasa koje čine sistem. Po metodi klasnih tačaka, složenost klase se određuje na osnovu sledećih parametara:

1. Broj metoda klase – klase koje imaju više javnih metoda obavljaju više operacija koje traže druge klase, tako da su one očigledno složenije od klasa sa manje javnih metoda. U ovoj meri se koriste samo javne metode,
2. Broj poziva metoda – klase koje pozivaju više metoda drugih klasa su složenije od metoda koje imaju manje poziva, pošto više poziva verovatno služi da bi se izvršilo više poslova unutar same klase,
3. Broj atributa klase – broj atributa je mera veličine podataka koje sadrži klasa. Pod pretpostavkom da složenije klase sadrže više podataka očigledno ima smisla koristiti broj atributa kao metodu određivanja složenosti klase.

Složenost sistema se meri u dve faze – na početku se određuje inicijalna mera sistema CP_1 (ova mera se može odrediti tokom faze elaboracije projekta), dok se finalna mera CP_2 određuje u kasnijim fazama. Razlog za definiciju dve različite mere leži u činjenici da neki podaci nisu dostupni u ranim fazama projekta ili bar nisu dovoljno pouzdani. Broj atributa klase se ne može predvideti na početku projekta, tako da se za određivanje mere inicijalne složenosti koriste samo informacije o broju metoda klase i broju metoda koje klasa poziva.

U ranim fazama implementacije se ove veličine mogu odrediti na osnovu dokumentacije sa dovoljnom pouzdanosti s obzirom da su u tim trenucima definisani sistemski dijagrami sekvenci koji pouzdano modeluju ove informacije. Inicijalna mera sistema se određuje tako što se za svaku poznatu klasu određuje složenost na osnovu broja metoda i broja zahteva koji se mogu odrediti iz dijagrama sekvenci.

U kasnijim fazama, kada se definišu klasni dijagrami, može se odrediti finalna mera pošto su raspoložive sve potrebne informacije. Finalna mera složenosti klasa pored broja metoda i poziva uzima u obzir i broj atributa pa se na osnovu ova tri parametra klase kategorizuju kao jednostavne, srednje i složene.

Za razliku od ostalih metoda po kojima se ukupna složenost određuje sabiranjem složenosti svih klasa, u ovoj metodi se klase prvo kategorizuju na četiri vrste:

1. PDT – domenske klase (Problem Domain Type),
2. HIT – klase za interakciju sa korisnikom (Human Interaction Type),
3. DMT – klase za obradu podataka (Data Management Type),
4. TMT – klase za obradu procesa (Task Management Type).

Ukupna veličina sistema se određuje kao težinska suma klasa razvrstanih na četiri navedena tipa klasa. Ako je N_{ij}

broj klasa tipa i (gde i može biti tip iz skupa {PDT, HIT, DMT, TMT}), i kompleksnosti j koja može biti jednostavna, srednja i komplikovana, totalna vrednost neprilagođenih tačaka slučajeva upotrebe se dobija po formuli 2.

$$TUCP = \sum W_{i,j} * N_{i,j} \quad (2)$$

Tabela 1 prikazuje vrednosti težinskih koeficijenata koji se primenjuju u zavisnosti od tipa i složenosti klase.

$W_{i,j}$	J	S	K
PDT	3	6	10
HIT	4	7	12
DMT	5	8	13
TMT	4	6	9

Tabela 1. Težinski koeficijenti u metodi klasnih tačaka

Tehnika klasnih tačaka je pokazala dobre rezultate u praksi pošto je primenom na skup od 40 projekata dala veličine koje su srazmerne veličinama projekata, iz čega se može zaključiti da se veoma verno može iskazati veličinu sistema na osnovu klasnog modela.

7. METRIKE KODA

Metrike koda su mere koje se određuju direktnim uvidom u izvorne fajlove programskog koda sistema. Ove metrike su na raspolaganju tek na kraju projekta pošto je tek tada završen programski kod sistema.

Prva mera koja se koristila radi merenja složenosti softvera je bila broj linija koda [9]. Ona je bila veoma pogodna i merodavna mera u proceduralnim jezicima. U njima je broj linija koda mogao manje više verno da odslika veličinu softvera koji se implementira. Najznačajnija metoda za procenu vremena na osnovu veličine softvera COCOMO II [10] i dan danas koristi broj linija koda kao jedinicu mere veličine softvera. Linije koda se danas pretežno koriste u sistemskom softveru – veličine raznih operativnih sistema se mogu naći izražene u linijama koda.

Linije koda pored svoje jednostavnosti, razumljivosti i lakoće određivanja imaju niz mana. Jedna od mana je utvrđivanje pravila definicije linije koda. Kao primer brojanja broja linija na različit način u zavisnosti od formata koda može se videti kod na slici 6.

if(i>0)*Prikaz*/ printf(i);	if(i>0) { // Prikaz printf(i); }
------------------------------	---

Slika 6. Primer identičnog koda sa različitim brojem linija

Iako je programski kod identičan, na desnoj strani bi se alatom za određivanje veličine koda dobila veća veličina merena u linijama koda. Da bi se prevazišli ovakvi problemi uvedene su dodatne mere broja linija koda:

1. LOC – broj fizičkih linija koda – koji predstavlja broj linija koda bez obzira na formatiranje,
2. ILOC – broj logičkih linija koda – koji predstavlja broj logičkih linija koda gde se uzima u obzir i način na koji su linije koda formatirane,
3. CLOC – broj linija koda koje predstavljaju komentare i opise u kodu.

Za određivanje fizičkih linija koda mogu se koristiti jednostavni alati koji do broja linija koda dolaze pronalaženjem broja karaktera za prelazak u novi red u izvornom kodu kao što Unix wc ili Windows PowerShell. Za određivanje broja logičkih linija i komentara je potrebno napraviti parser koji će pomoću određene translacione gramatike ili regularnih izraza, u skladu sa definicijom gramatike programskog jezika analizirati izvorni fajl i odrediti broj komentara i logičkih linija koda.

Broj linija koda je potpuno neprimenljiv u aplikacijama gde se koristi više različitih jezika istovremeno. Kao očigledan primer se mogu videti web aplikacije koje se realizuju kombinovanjem java skripta i HTML-a na klijentskoj strani, standardnih jezika kao što su Java ili C# na serverskoj strani i jezika specifičnih za baze podataka kao Oracle PL/SQL ili Microsoft T-SQL za implementaciju procedura i funkcija na serveru baze podataka. Kombinovanje podataka o broju linija koda iz različitih jezika je nemoguće, naročito ako se ima u vidu da za određene funkcionalnosti ne postoje ekvivalenti među njima.

Imajući u vidu nedostatke mere broja linija koda vidi se da to nije pogodna mera za određivanje veličine softvera, sem u nekim specifičnim slučajevima. Iako broj linija koda nije pogodan za određivanje veličine, ova mera se može iskoristiti za određivanje nekih indirektnih mera u sistemu, kao što je kvalitet koda koji se izražava sa odnosom broja komentara i ukupnog broja linija koda, ili procenat generisanog koda.

U objektno orijentisanom softveru se može definisati niz dodatnih metrika koje se mogu određivati na osnovu analize koda. Ove metrike mogu biti Halstedova [11] ili Mek Kejbova ciklomatska kompleksnost [12], dubina nasleđivanja, međuzavisnost modula i slično.

Alati koji se koriste za analizu koda i merenje mogu biti integrisani u razvojna okruženja kao što je Visual Studio 2008 ili Eclipse, a mogu biti i nezavisni analizatori koda. Ovakvi alati mogu da analiziraju ili ceo sistem ili pojedinačne komponente (klase, funkcije) kako bi svakoj komponenti dale odgovarajuću kompleksnost. Ovakvi alati se uglavnom koriste po implementaciji projekta za pronalaženje problematičnih delova koda pošto komponente koje imaju veću kompleksnost uglavnom predstavljaju probleme prilikom kasnijeg održavanja.

8. ZAKLJUČAK

Tehnike merenja veličine softvera daju efikasan metod procene veličine projekata na osnovu modela kojim su predstavljeni softverski sistemi. U zavisnosti od metodologije razvoja softvera koja se primenjuje, različite tehnike se mogu koristiti u softverskim organizacijama.

U praksi se metode funkcionalnih tačaka [2,4] i tačaka slučajeva upotrebe [6,7] mogu efikasno koristiti na početku projekta radi procene veličine sistema u ranim fazama projekta kada je potrebno na osnovu modela zahteva, funkcija ili slučaja upotrebe, u oskudici detalja dati procene na osnovu kojih će se planirati razvoj softvera. Pored toga ove tehnike daju procenu veličine softvera sa korisničke strane gledišta pošto se fokusiraju na funkcionalnosti sistema koje vidi korisnik a ne na tehničke detalje implementacije koji su nerazumljivi korisnicima.

Klasne tačke [8] i Fečkeova [5] varijanta funkcionalnih tačaka mogu dati informacije o realnoj veličini sistema u kasnijim fazama kada je već razvijen veliki broj modela. Ove veličine se uglavnom koriste za eventualnu korekciju planova u slučaju da se veličine sistema dobijene po funkcionalnim metodama značajnije razlikuju od metoda zasnovanih na preciznijim modelima. One bi trebale da daju jasniju sliku o veličini softvera pošto uzimaju u obzir i tehnologiju koja će se koristiti, što daje realniju sliku troškova i cene izrade softvera

Mere koda [9,12] su efikasan način za merenje veličine softvera pošto za razliku od ostalih metoda ne predviđaju veličinu nego mere softver pošto je napravljen, ali se usled heterogenosti današnjih softverskih rešenja mogu primenjivati samo na određene delove koda. One se ne koriste često kao mera veličine softvera ali daju korisne informacije u fazi održavanja softvera pošto se na osnovu njih može procenjivati lakoća održavanja, očekivani broj grešaka i konačni kvalitet koda.

Poznavanje i primena metoda merenja može značajno unaprediti sistem planiranja i donošenja odluka u procesu razvoja softvera tako da se softverske organizacije moraju više fokusirati na mogućnosti koje im nude ove tehnike.

9. LITERATURA

- [1] H. Zuse, „*Resolving the Mysteries of Halstead Measures*“, Technische Universität Berlin, November 2005
- [2] D. Garmis, D. Herron, „*Functional Point Analysis*“, Addison-Wesley, Boston USA, 2001
- [3] A. Albrecht. „*Software Development Cost Estimation Using Functional Point*“, IEEE Transactions on Software Engineering, April 1994
- [4] ISO/IEC 14143-1:2007 Information technology – Software measurement – Functional size measurement
- [5] T. Fetcke, A. Abran and T.H. Nguyen, „*Mapping the OO-Jacobson Approach into Function Point Analysis*“, Proc. TOOLS-23'97, IEEE Press, Piscataway, N.J., 1998.
- [6] R. Clemmons, „*Project Estimation With Use Case Points*“, The Journal of Defense Software Engineering, 2006
- [7] S. Kusumoto, F. Matukawa, I. Katsuro, „*Estimating Effort by Use case Points: Method, Tool and Case Study*“, 10th International Symposium on Software Metrics (METRICS'04)
- [8] G. Costagliola, G. Tortora, „*Class Point: An Approach for the Size Estimation of Object-Oriented Systems*“, IEEE Transactions on Software Engineering, Vol. 31, No.1 January 2005
- [9] R. Park. “*Software size measurement: a framework for counting source statements*” CMU/SEI-92-TR-020. <http://www.sei.cmu.edu/pub/documents/92.reports/pdf/tr20.92.pdf>
- [10] B. Boehm et al., “*Software Cost Estimation with COCOMO IP*”, Prentice-Hall, New Jersey, USA, 2000
- [11] M. Halstead, “*Elements of Software Science*”, Elsevier North-Holland, New York, USA, 1997
- [12] T. J. McCabe, “*A Complexity Measure*” IEEE Transactions on Software Engineering, vol. 2, no. 4, pp. 308-320, July 1976