

MERENJE TOKOM PROCESA RAZVOJA SOFTVERA

Jovan Popović, Gowi d.o.o. Pančevo, jovan.popovic@gowi.rs

Sadržaj – U planiranju softverskih projekata jedan od najbitnijih zadataka je procenjivanje vremena potrebnog da se projekat realizuje. Da bi se procenilo potrebno vreme, potrebno je imati mehanizam kojim bi se mogla meriti veličina sistema kako na početku tako i tokom razvoja projekata kako bi se mogao ispravno planirati nastavak razvoja. Danas postoji veliki broj metoda koje se mogu koristiti za merenje veličine softvera tako da je jedan od najtežih zadataka izbor najpogodnijih metoda koje se koriste tokom merenja kao i trenutak kada se merenje može vršiti određenim metodama. U radu su opisane metode koje se mogu koristiti prilikom merenja kao i trenutci u kojima se merenje može vršiti.

1. UVOD

Merenje je veoma bitan element razvoja softvera zato što pruža važne informacije o softveru koje se mogu koristiti tokom planiranja razvoja sistema. Bez merenja, nije moguće objektivno proceniti veličinu softvera a samim tim ni potrebno vreme za izradu. Bez procene vremena potrebnog za razvoj nije moguće ispravno planirati tok razvoja softvera. Samim tim dobar proces merenja je ključan za planiranje aktivnosti tokom razvoja softvera.

Formalno gledano, merenje je proces kojim se projekti mogu preslikati u skup realnih brojeva na taj način da se očuva relacija poretka među veličinama projekata. Neka je (P, \geq) relacioni model gde je P skup projekata i \geq relacija kojom se projekti mogu porediti po veličini (pri tome „veličina“ se može definisati subjektivnom procenom ili vremenom potrošenim na projektima). Neka je $f: P \rightarrow R$ funkcija kojom se elementi iz skupa projekata preslikavaju u skup nenegativnih realnih brojeva tako da važi da:

$$(\forall p_1, p_2 \in P) p_1 \geq p_2 \Leftrightarrow f(p_1) \geq f(p_2) \quad (1)$$

Funkcija koja zadovoljava ovu predikatsku formulu i preslikava projekte u realne brojeve se naziva merna funkcija, dok se slike projekata nazivaju mere projekata. Mere projekata nam omogućavaju da poređenje i obradu projekata vršimo u domenu realnih brojeva umesto u apstraktnom domenu projekata. U praksi je veoma teško odrediti funkciju koja direktno preslikava projekte u realne brojeve tako da se za merenje obično koriste indirektno metode kojima se projekti predstavljaju ekvivalentnim modelima, a onda se modeli konvertuju u realne brojeve mernim funkcijama. Modeli mogu biti funkcionalni modeli, UML modeli ili bilo koji drugi vid predstavljanja projekata. U slučaju da model verno predstavlja projekat i predstavlja njegov ekvivalent, mera dobijena indirektno na osnovu modela ekvivalentnog projektu predstavlja meru samog projekta.

Danas postoji veliki broj metoda kojima se na osnovu modela određuje veličina projekata. Metode merenja koriste prebrojavanje različitih karakteristika sistema (funkcija, klasa, slučajeva upotrebe, fajlova) eventualno ih kategorizuju po složenosti i određuju broj elemenata po svakoj kategoriji. Ukupni brojevi elemenata softverskih sistema predstavljaju

konačan vektor veličine softvera čije dimenzije predstavljaju brojevi pomenutih elemenata. Kao primer, ako su N_c , N_t i N_r brojevi klasa u sistemu, tabela u bazi podataka i zahteva u specifikaciji, respektivno, vektor veličine sistema se može predstaviti kao vektor $N = (N_c, N_t, N_r)$. U slučaju da se ne koristi jednostavno prebrojavanje elemenata nego da se prvo klasifikuju po složenosti na složene, srednje i jednostavne, mogao bi se naći broj elemenata po svakoj klasi složenosti i po svakom tipu, pa bi se umesto trodimenzionalnog vektora sistem predstavio devetodimenzionalnim vektorom gde bi dimenzije bile brojevi elemenata određeni za svaku klasu složenosti ponaosob. Kao merna funkcija najčešće se koristi skalarni proizvod vektora težine W kao u formuli 2. Dimenzije vektora težina W_i se određuju empirijski.

$$Size = \vec{W} * \vec{N} = \sum W_i * N_i \quad (2)$$

Najprimenljivija metoda je metoda funkcionalnih tačaka FPA [1] u kojoj je vektor težine sadrži informacije o veličini aktivnosti i fajlova u sistemu. Danas postoji veliki broj modela razvoja softvera koji se koriste tokom razvoja. Iako se metode razlikuju u detaljima, osnovni princip je da se softver razvija u nekoliko faza koje su podeljene interno na veći broj iteracija. Tokom faza, projektni tim je fokusiran na određeni aspekt softvera (npr. zahteve, specifikaciju, programiranje i testiranje). Kao primer u RUP-u [2] razvoj softvera se deli na četiri faze – upoznavanje sa projektom (Inception) gde se tim upoznaje sa zahtevima, elaboracija (Elaboration) gde se analizira i dizajnira sistem, konstrukcija (Construction) gde je fokus na implementaciji, i tranzicija (Transition) gde je fokus na testiranju.

Kao što se tokom faza razvoja softvera projektni tim fokusira na različite aspekte razvoja, tako se i merenje zasniva na podacima koji su od interesa u trenutnoj fazi. Tokom ranijih faza kreiraju se apstraktniji modeli sa manje detalja zasnovani na korisničkim zahtevima, dok se u kasnijim fazama kreiraju realniji modeli sa više tehničkih detalja i preciznijom opisom sistema pošto je tada na raspolaganju programski kod. Merenje sistema tokom faza razvoja softvera mora da bude usklađeno sa modelima raspoloživim u određenim fazama. U narednim sekcijama su predložene metode koje se mogu koristiti tokom navedenih faza izrade softvera u RUP-u.

2. MERENJE TOKOM FAZE UPOZNAVANJA

Tokom faze upoznavanja projektni tim vrši analizu poslovnih procesa, identifikuje korisnike sistema, sakuplja zahteve o sistemu, kreira konceptualne modele i definiše okvirno sa kakvim podacima će se raditi, i eventualno pravi okvirni model slučajeva upotrebe. Pošto su ovo raspoložive informacije o sistemu na osnovu kojih se modeluje sistem, na osnovu njih se vrši merenje veličine softvera. Cilj merenja tokom upoznavanja je određivanje veličine sistema na osnovu oskudnih podataka o sistemu kojima se u datom trenutku raspolaže. Metode koje se mogu koristiti tokom faze

upoznavanja su NESMA indikativna metoda [3] i uprošćena metoda tačaka slučajeva [4]. Tokom ove faze okvirni zahtevi koji su prikupljeni su nedovoljni za identifikaciju i procenu složenosti transakcija i fajlova koji se koriste u sistemu tako da je potrebno koristiti neku uprošćenije metode. Uprošćene metode ne uzimaju u obzir sve informacije o sistemu pa mogu uzrokovati velike greške ali to je i očekivano s obzirom da je u pitanju početak projekta. Boem je pokazao [5] da odstupanje od stvarne veličine sistema na početku može biti i do četiri puta, tako da je na početku projekta prihvatljivo korišćenje neke aproksimativne metode pod uslovom da daje bar nekakvu informaciju o veličini sistema.

U NESMA varijanti FPA metode [3] definisana je tzv. indikativna metoda za merenje veličine softvera na osnovu ograničenog broja informacija. S obzirom da u fazi upoznavanja nema dovoljno podataka ovo je jedina FPA metoda primenljiva na početku projekta. Indikativna NESMA metoda u obzir uzima samo broj internih i eksternih fajlova u sistemu bez analize nivoa kompleksnosti. Ako su N_{if} i N_{eif} ukupni brojevi internih i eksternih logičkih fajlova respektivno, sistem se može predstaviti vektorom veličine $N = (N_{if}, N_{eif})$ dok se veličina sistema se određuje kao skalarni proizvod na osnovu formule (2), tako što se ubaci težinski vektor W sa vrednošću (35, 15), čime se dobija formula 4:

$$Size = 35 * N_{if} + 15 * N_{eif} \quad (4)$$

Podaci o N_{if} i N_{eif} se mogu naći na osnovu analize konceptualnog modela ili modela poslovnih objekata sistema. Ideja ove aproksimativne formule je da su svi fajlovi prosečne kompleksnosti, da ima po i po jedan upit i izlaz po svakom eksternom fajlu. Iako ova pretpostavka ne važi u svim slučajevima, usled nedostatka detaljnijih informacija ovoj je jedina metoda koja se može koristiti. Primena bilo koje detaljnije FPA metode usled nedostatka informacija bi dovela do još većih grešaka. S obzirom da po [5] greška u početnim fazama može biti i do četiri puta veća od veličine sistema ova metoda je sasvim prihvatljiva.

Pored NESMA FPA metode moguće je koristiti i pojednostavljenu metodu tačaka slučajeva upotrebe [4] na osnovu inicijalnog modela slučajeva upotrebe ili poslovnog modela sistema. S obzirom da se tokom upoznavanja sa projektom ne definišu detaljni scenariji slučajeva upotrebe nije moguće odrediti kompleksnost svakog slučaja upotrebe pojedinačno, pretpostavlja se da je veličina svakog slučaja upotrebe prosečna. Pored broja slučajeva upotrebe N_{UC} , kao dodatna informacija o veličini sistema se koristi i broj korisnika sistema N_A , pa se sistem predstavlja dvodimenzionalnim vektorom $N = (N_{UC}, N_A)$. Mera veličine sistema se određuje na osnovu formule (2) tako što se u formulu ubaci vektor težine W sa vrednošću (10, 3). Konačna formula za veličinu sistema je:

$$Size = 10 * N_{UC} + 3 * N_A \quad (5)$$

U ovoj aproksimativnoj metodi se podrazumeva da su korisnici sistema ljudi koji koriste grafički korisnički interfejs. U slučaju da u sistemu pored standardnih korisnika postoje i eksterni sistemi i korisnici koji koriste jednostavne interfejse (npr. komandni prompt ili telnet) mora se analizirati svaki korisnik i pronaći broj jednostavnih, srednjih i složenih korisnika – N_A^L , N_A^A i N_A^H respektivno, tako da se ove informacije koriste kao dimenzije vektora veličine sistema.

Formula za veličinu se dobija tako što se u formuli (2) ubaci težinski vektor W sa vrednošću (10,1,2,3). Formula za konačnu veličinu sistema je:

$$Size = 10 * N_{UC} + N_A^L + 2 * N_A^A + 3 * N_A^H \quad (6)$$

Kompleksnost korisnika se određuje kao po standardnoj UCP metodi s obzirom da je pretpostavka da ne može biti mnogo različitih tipova korisnika i da se njihova detaljna analiza može efikasno izvršiti u kratkom vremenu. Pojednostavljenje se vidi u uklanjanju sume po tipovima slučajeva upotrebe koji se ne mogu detaljno analizirati s obzirom na manjak raspoloživih podataka.

3. MERENJE TOKOM FAZE ELABORACIJE

Tokom ranih iteracija elaboracije zahtevi prikupljeni tokom upoznavanja se detaljnije analiziraju i razrađuju tako da se kreira veći broj modela koje je moguće koristiti radi merenja veličine sistema. Metode za merenje koje su na raspolaganju su uprošćene metode funkcionalnih tačaka kao Mark II [6], uprošćena NESMA [3], COSMIC [7] metoda, ili metoda tačaka slučajeva upotrebe [4].

Po Mark II metodi posmatraju se samo ulazi, izlazi, i fajlovi bez obzira da li su interni ili eksterni. Veličina sistema se opisuje vektorom $N = (N_i, N_e, N_o)$ gde su N_i , N_e , i N_o , ukupni brojevi ulaza, entiteta i izlaza respektivno. Broj ovih elemenata se određuje na osnovu svakog tipa bez ikakvih težinskih koeficijenata. Veličina sistema se određuje tako što se u formulu (2) ubaci vektor težine W sa vrednošću (0.58, 1.66, 0.28) po formuli:

$$Size = 0.58 * N_i + 1.66 * N_e + 0.28 * N_o \quad (7)$$

Imajući u vidu da za Mark II metodu nije potrebno primenjivati detaljnu analizu kompleksnosti transakcija i fajlova već je samo potrebno identifikovati ih i prebrojati, ova metoda je najbliža NESMA indikativnoj metodi i može se primenjivati u veoma ranim iteracijama elaboracije.

Ostale uprošćenje FPA metode (NESMA procenjena i COSMIC metoda) zahtevaju da se identifikuju sve transakcije i fajlovi ali uključuju i njihovu kompleksnost. Ipak za razliku od detaljnih metoda one ne zahtevaju detaljnu analizu i kategorizaciju nego se svakom elementu dodeljuje podrazumevana težina. Uobičajeno je da se svim fajlovima i transakcijama dodeljuje srednja složenost i tako se unose u standardne FPA formule.

Tokom ranih iteracija elaboracije se definišu detaljniji slučajevi upotrebe tako da je moguće koristiti i egzaktnu metodu tačaka slučajeva upotrebe. Na osnovu modela slučajeva upotrebe i detaljnih scenarija slučajeva upotrebe moguće je identifikovati sve korisnike i slučajeve upotrebe i odrediti njihove klase složenosti. Složenost slučajeva upotrebe se određuje na osnovu analiza broja aktivnosti na osnovu scenarija, i kao i u funkcionalnim modelima može biti niske, srednje i visoke kompleksnosti. Veliki problem u primeni metode tačaka slučajeva upotrebe je nepostojanje standarda za njihovu definiciju i opisivanje. Danas postoji veliki broj korišćenih formata specifikacije koji sa manje ili više detalja opisuju metode specifikiranja scenarija slučajeva upotrebe. Ovo je jedan od osnovnih razloga zašto ova metoda

nije u potpunosti zamenila metodu funkcionalnih tačaka u objektno orijentisanim sistemima.

Tokom kasnijih iteracija faze elaboracije razvijen je kompletan, detaljni funkcionalni model koji se može koristiti za detaljnu FPA analizu. Detaljni funkcionalni model omogućava da korišćenje standardne FPA metode, detaljne NESMA metode, ili detaljne COSMIC metode. Ove metode uzimaju u obzir sve karakteristike sistema i daju najtačnije rezultate. Sve metode su potpuno ravnopravne, priznate po ISO/IEC standardu i po veličinama sistema daju relativno slične rezultate. Primena neke od ovih metoda zavisi isključivo od obučenosti članova projektnog tima za primenu metoda.

Današnja primena objektno orijentisane analize i dizajna stvara problem u kompatibilnosti sa zahtevima funkcionalnih modela potrebnih za FPA metode. Da bi se prevazišla ova nekompatibilnost neki autori su definisali pravila kojima se OO modeli prevode u funkcionalne modele. Jedna od najprihvaćenijih metoda je Fečkeova metoda [8] za mapiranje OO koncepata na FP koncepte u kojoj je definisano 15 pravila za prevođenje OO koncepata u FP koncepte. Primenom ovih pravila se standardni UML modeli mogu konvertovati u model kompatibilan sa funkcionalnim tačkama nad kojim se vrši merenje u skladu sa FPA metodom.

4. MERENJE TOKOM FAZE IZRADE

Tokom faze izrade projekta postoji najviše elemenata za konkretno merenje sistema pošto se kreiraju konkretne komponente koje predstavljaju sam sistem. Usled iterativnog pristupa razvoju softvera tokom faze izrade se prepliću aktivnosti modelovanja tokom kojih sa kreiraju UML modeli softvera, aktivnosti programiranja kojima se na osnovu modela kreira programski kod koji realizuje sistem i testiranja kojim se vrši jedinično testiranje programskog koda. Konkretni UML modeli sistema se predstavljaju detaljnim modelima klasna i sekvenci. U slučaju da se koriste alati koji tokom razvoja sinhronizuju modele i programski kod (*round-trip engineering*) modeli i kod bi trebali da budu ekvivalentni tako da se veličina dobijena merenjem modela i programskog koda poklapaju. Tokom faze konstrukcije je moguće koristiti metode klasnih tačaka (inicijalnu i egzaktnu) [9] ili metode merenja izvornog koda [10]. Imajući ovo u vidu može se zaključiti da su najpogodnije metode koje se mogu primeniti u ovoj fazi metoda klasnih tačaka – kojom se na osnovu analize dijagrama klasa i sekvenci određuje veličina sistema, merenje linijama koda – metoda kojom se prebrojavanjem linija koda utvrđuje veličina, merenje ciklomatskom kompleksnošću [11] – apstraktna metoda kojom se meri kompleksnost na osnovu apstraktnih modela kojima se programski kod predstavlja grafovima.

U ranijim iteracijama razvoja ili na početku svake iteracije se dizajnira rešenje i kreira detaljan model klasa sistema. Tada je na raspolaganju model klasa i sekvenci kojim se opisuje struktura i kolaboracija među klasama sistema. Radi merenja veličine na osnovu modela klasa najpogodnija je metoda klasnih tačaka kojom se merenje veličine sistema vrši na osnovu klasa koje se nalaze klasnom dijagramu. Merenje veličine klasa se vrši određivanjem broja metoda klasa (NOM) na osnovu pregleda dijagrama klasa, i broja poziva metoda (NSR) pregledom dijagrama sekvenci za

svaku metodu se određuje koliko drugih metoda je poziva. Određivanjem vektora (NOM, NSR) za svaku klasu se procenjuje njena složenost kao jednostavna, srednja i složena. Na osnovu kompleksnosti klase se određuje veličina, dok se veličina sistema dobija tako što se pojedinačne veličine klasa sabere.

Tokom aktivnosti programiranja se implementira programski kod koji realizuje funkcionalnosti definisane dizajnom. Jedna od prvih metoda koje su bile korišćene za određivanje veličine programskog koda je metoda brojanja linija izvornog koda (SLOC). U većim ili heterogenijim sistemima umesto veličine celog sistema ova metoda se koristi radi određivanja veličine pojedinih modula u trenucima kada je implementiran odgovarajući programski kod. Radi određivanja veličine modula se često koriste parseri ili analizatori koda koji prebrojavaju linije nad statičkim fajlovima. S obzirom da su današnji sistemi implementirani kao skup različitih modula napisanih u različitim jezicima (Java, SQL, HTML, JavaScript) mera veličine linijama koda nije praktična zato što nije moguće uporediti module kao što su servleti napisani na Javi i uskladištene procedure napisane na SQL-u. Međutim ako se definiše konačan skup jezika koji se koriste u sistemu i odrede veličine merene u linijama koda na svakom korišćenom jeziku (npr. N_{Java} , N_{SQL} , N_{HTML} za broj linija koda napisanih na Javi, SQL-u i HTML-u respektivno) može se definisati vektor veličine sistema kao $N = (N_{Java}, N_{SQL}, N_{HTML})$ kojim se može definisati veličina sistema. Odabirom odgovarajućih težinskih koeficijenata se na osnovu ovakvih vektora može odrediti konačna veličina sistema.

Da bi se prevazišao problem heterogenosti sistema umesto izvornog koda se može koristiti neki ekvivalentni model kojim se predstavlja izvorni kod kao na primer grafom toka kontrole. Grafovima toka kontrole se programski delovi koji se sastoje od sekvenci koje se izvršavaju pod određenim uslovima predstavljaju kao grane grafa toka kontrole. Čvorovi grafa su uslovi, pozivi metoda ili bilo koja druga mesta u kodu na kojima se odlučuje kuda će se nastaviti izvršavanje programa. Metoda predstavljanja koda grafovima toka kontrole izlazi van okvira rada ali se može naći u [12]. Grafovi toka kontrole su pogodni zato što je moguće uvesti apstrakciju u heterogenim aplikacijama kao što su web aplikacije. Iako se tipična web aplikacija sastoji od mešavine HTML, JavaScript, Java/C# i SQL koda što otežava merenje klasičnim metodama linija koda, zajedničko ovom kodu je to što komponente uvek pozivaju jedna drugu čak iako pozivalac i pozvana komponenta nisu napisane na istom jeziku. Ovo omogućava kreiranje uniformnog grafa toka kontrole koji se proteže kroz različite slojeve aplikacije. Na ovaj način se pomoću jedinstvenog grafa toka kontrole mogu modelovati pozivi različitih modula iz različitih slojeva aplikacije.

Jedna od najpogodnijih metoda za merenje kompleksnosti grafova toka kontrole je Mekejbova [11] ciklomatska složenost grafova. U praksi je pokazano da Mekejbova ciklomatska kompleksnost daje slične rezultate kao i mera linija koda, s tim što se efikasnije od brojanja linija koda može koristiti u heterogenim sistemima.

Pri kraju svake iteracije faze konstrukcije, može se koristiti egzaktnija metoda klasnih tačaka. S obzirom da se po implementaciji programskog koda i eventualnom reverznom

inženjeringu modela klasa dobija se još jedna značajna informacija o veličini klasa – broj atributa klasa (NOA), ova informacija se može uključiti u karakteristike sistema. Veličina svake klase se predstavlja vektorom (NOM, NSR, NOA) i na osnovu ovog vektora se određuje kompleksnost svake klase kao i po inicijalnoj metodi klasnih tačaka. Metoda određivanja veličine je ista kao i kod aproksimativne metode koja se koristi na početku faze konstrukcije.

5. MERENJE TOKOM FAZE TRANZICIJE

Tokom faze tranzicije kod je relativno stabilan i uglavnom se vrše aktivnosti testiranja sistema, korekcija i uklanjanja defekata. S obzirom da nema apstraktnih modela nema mnogo izbora mernih metoda pošto su uglavnom očigledne i mogu se konkretno primeniti na sistem. Tokom ove faze merenje se vrši isključivo na izvornom kodu aplikacije i koristi se isključivo za praćenje stepena promena, učestanosti defekata, lakoće održavanja i slično. Metrike koje se koriste u analizi koda mogu biti stepen kohezije i povezanosti modula, dubina nasleđivanja hijerarhije, Mekejbova ciklomatska [11] ili Halstedova [13] kompleksnost. Kohezija i povezanost modula se može odrediti posmatranjem dijagrama sekvenci ili primenom NSR mere iz metode klasnih tačaka [9]. Mekejbova i Halstedova kompleksnost se može koristiti za otkrivanje modula (na primer klasa, metoda, okidača u bazi i sl.) sa visokom kompleksnošću koji uglavnom predstavljaju probleme u održavanju i performansama sistema.

Linije koda se i dalje koriste, ali ne merene statički kao veličina sistema ili modula kao u fazi konstrukcije. Tokom tranzicije se prati stepen promene koda tako da se posmatra dinamička slika programskog koda gde se posmatra koliko se kod menja u odnosu na originalni. Mere koje se koriste su broj dodatih, izmenjenih i obrisanih linija koda i često se izražavaju u funkciji vremena kao broj promenjenih linija na dan. Kao finalna mera koja nema direktne veze sa programskim kodom koristi se skup mera za praćenje vremenske zavisnosti defekata. Ove mere mogu biti broj defekata pronađenih/rešenih u vremenskom intervalu, broj defekata po modulu ili prosečan broj defekata na svakih hiljadu linija koda.

ZAKLJUČAK

Merenje veličine softverskih sistema je veoma bitna aktivnost u razvoju softvera koja omogućava objektivno planiranje na osnovu realnih podataka o softveru koji se implementira. Merenje predstavlja kariku koja nedostaje u većini procesa razvoja softvera koja povezuje inženjerske sa menadžerskim aktivnostima. Bez merenja, planiranje procesa razvoja se zasniva na subjektivnim procenama.

Iako danas postoji veliki broj metoda merenja softvera one nisu široko primenjene uglavnom zbog nepoznavanja ili nepostojanja metodologije koja bi definisala kada se i kako postojeće merne metode mogu koristiti. Ovaj rad je imao za cilj prezentaciju metoda i davanje smernica za njihovu primenu u okviru procesa razvoja softvera. S obzirom da bi detaljno objašnjavanje svih metoda izašlo van okvira rada, u radu je prezentovan pregled postojećih metoda koji se može koristiti kao početna tačka za dalje istraživanje i njihovu primenu.

REFERENCE

- [1] D. Garmis, D. Herron, „*Functional Point Analysis*“, Addison-Wesley, Boston USA, 2001
- [2] R. Kruchten „*The Rational Unified Process – An Introduction*“, Addison-Wesley, Boston, USA, 2000
- [3] Netherlands Software Metrics Association (NESMA), „*Definition and Counting Guidelines for the Application of Function Point Analysis*“, Version 2.0, Amsterdam NESMA 1997.
- [4] 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)
- [5] B. Boehm et al., „*Software Cost Estimation with COCOMO II*“, Prentice-Hall, New Jersey, USA, 2000
- [6] C. Symons, „*Software Sizing and Estimating, Mk II Function Point Analysis*“, John Wiley & Sons, Chichester, England, 1991
- [7] A. Abran, R. Meli, C. Simons, „*COSMIC Full Function Point Method: 2004 – State of Art*“, SMEF04, Rome, Italy, Jan. 2004
- [8] 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.
- [9] 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
- [10] 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>
- [11] T. J. McCabe, „*A Complexity Measure*“, IEEE Transactions on Software Engineering, vol. 2, no. 4, pp. 308-320, July 1976
- [12] A.V. Aho, R. Sethi, J.D. Ullman, „*Compilers/Principles, Techniques and Tools*“, Addison-Wesley 1986.
- [13] M. Halstead, „*Elements of Software Science*“, Elsevier North-Holland, New York, USA, 1997

Abstract – In software planning one of the most important task is estimating time needed for implementation. Therefore, there is a need to have some measurement mechanism for size of application. Today we have lot of methods available for measurement of software size but there are not clear guidelines which methods might be used during development life cycle. This paper presents measurement methods that can be used during software development process, along with phases when they might be applied.

MEASUREMENT DURING SOFTWARE DEVELOPMENT PROCESS

Jovan Popovic