DOI: 10.5937/vojdelo1606316Z

ПРОЦЕНА УКУПНОГ ВРЕМЕНА ИЗРАДЕ СОФТВЕРСКОГ ПРОИЗВОДА

Јован Живадиновић* и *Зорица* Медић**
Висока школа за пословну економију и предузетништво, Београд
Борис Јевтић
Универзитет Унион у Београду, Рачунарски факултет

Основни мотив овог рада о софтверском инжењерству јесте да се инжењерски приступ развоју софтвера супротстави уметничком. Софтверско инжењерство је најмлађа инжењерска дисциплина и, као таква, још увек поседује висок степен произвољности. Сам процес развоја софтвера у великој мери оскудева у методама и техникама, као и одговарајућим алатима који би га учинили ефикаснијим. Недостатак стандарда развоја је још израженији. Међутим, најболнију тачку софтверског инжењерства представља велико одсуство проверених и поузданих мера, односно техника мерења, као основних средстава инжењерске верификације резултата. Значај проблема је вишеструко наглашен потребом да се обезбеди висок квалитет софтвера и на софтверу заснованих система.

Циљ овог рада јесте да се развије и систематизује оригинални формални поступак за процену развоја информационих система у раним фазама животног циклуса софтвера, преко метрика модела података. Овај циљ остварен је коришћењем искустава наших инжењера из области развоја софтвера, као и резултата других истраживача који су се бавили метрикама софтверског инжењерства. Метрике модела података израчунавамо на основу података који се могу очитати са базног модела података, представљеног дијаграмом објекти и везе (DOV), који се дефинише помоћу четири основна концепта: ентитета, веза, атрибута ентитета или веза и вредности. Идеја је да се сложеност процеса представи функцијом броја ових концепата и броја атрибута типова ентитета. Технике процене представљају основу за планирање и успешно извођење софтверских пројеката. Ова величина је важна, јер служи као основа за прорачун других величина битних за софтверски пројекат, као што су трошкови или укупно време израде софтверског производа. У раду је коришћен статистички метод, чији поступци процене спадају у емпиријске параметарске методе, мада имају и нека својства метода експертских процена. Развијени поступак процене представља корак у покушајима да се дође до погодних мера којима бисмо оцењивали величину и сложеност модела података и вршили процену трошкова и ресурса потребних за развој инфор-

* Dr Jovan Živadinović je vanredni professor.

** Dr Zorica Medić je docent.

мационих система. Посебан значај предложних метрика је у томе што се до њихових вредности може доћи у најранијим фазама пројектовања информационих система. Такође, развијене су и метрике којима, на основу познавања модела података, можемо квантификовати и карактеристике информационог система као целине у фази логичког пројектовања. Предложене метрике су тестиране на конкретним моделима, а добијени резултати су приказани.

Кључне речи: *софтверско инжењерство, софтверске метрике, процена трошкова, модели података*

Увод

Рани дани развоја рачунарства дефинисали су софтвер као „уметничку форму". Обележило их је и постојање јако малог броја формалних метода, а још је мањи био степен њихове примене. Свет софтвера био је видно недисциплинован, а људи који су се бавили развојем софтвера су у томе уживали. Већина њих би желела да то чини и даље. Међутим, значај софтвера и на софтверу заснованих рачунарских система данас је нарастао до те мере да се исправно и поуздано функционисање софтвера намеће као витално питање будућности људске заједнице. Софтвер данас контролише свемирске летове, као и нуклеарне потенцијале „великих сила". Функционисање тржишта и платног промета, берзи, банкарских и других сличних система заснива се на софтверу. Глобални информациони сервиси засновани на софтверу потискују друге видове информисања. Укупно гледано, може се очекивати да ће значај софтвера у будућности и даље расти. Наравно, нико озбиљан не жели да будућност препусти неодговорним људима.

Када се говори о савременом софтверу, још један значајан аспект се не сме изгубити из вида. Софтвер је постао индустрија која данас остварује највећи профит међу свим гранама индустрије, чак већи од прехрамбене или индустрије оружја. Тако брзу експанзију није доживела ниједна индустријска грана у историји. Стога се економичан развој софтвера поставља као предуслов опстанка на тржишту. Основно питање, које се поставља с тим у вези, јесте питање управљања софтверским пројектима. Оптимални распореди пројектних задатака, праћење и контрола пројеката могу значајно да снизе трошкове развоја софтвера и допринесу конкурентности развојне организације. Овај рад бави се и питањима софтверског менаџмента. Потреба за квантификовањем величина које се односе на процес развоја софтвера, и самих резултата рада, односно софтверских производа, и са овог аспекта је наглашена. Потребан предуслов, поново, представљају мере и технике мерења (Тешић Д. 1996).

Софтверско инжењерство

Настанак и еволуција софтверског инжењерства, као дисциплине у оквиру рачунарске технике, могу се пратити од тренутка када је програмирање као активност већ достигло одређену зрелост. У раним данима рачунарства проблем програми-

рања се, у суштини, сводио на одређивање и успостављање редоследа инструкција са циљем да рачунар уради нешто корисно. Како је већ речено, проблеми које је требало испрограмирати били су врло добро дефинисани (нпр. решавање диференцијалне једначине). Програм је најчешће писао сам корисник, решавајући задатке за своје личне потребе. Комуникација човека и рачунара обављала се коришћењем машинских језика или, у бољим случајевима, симболичких машинских језика.

Како су рачунари улазили у ширу употребу, и постајали све јефтинији, рачунарством се бавило све више људи. Касних педесетих година појавили су се језици вишег нивоа како би се комуникација човека и рачунара учинила једноставнијом. Међутим, компјутерски програм је и даље писала једна особа, и даље за добро дефинисан проблем. Отприлике у то време „програмирање" је стекло статус професије. Постојали су специјалисти чији је задатак био да неформалну спецификацију проблема добијену од крајњег корисника претворе у прецизан скуп машинских инструкција. Међутим, овде су настали и први проблеми. Наиме, дешавало се да програмер погрешно интерпретира корисничке захтеве, чак и при овако малим задацима.

У периоду раних шездесетих изведен је врло мали број „већих" софтверских пројеката. На пример, оперативни систем CTSS развијен је у MIT-u. На оваквим пројектима радили су врхунски стручњаци, који су при том били и изузетно добро „мотивисани". Средином и крајем шездесетих велики софтверски системи већ су постајали комерцијално доступни. Најбоље документовани пројекат овог типа је оперативни систем OS 360, намењен IBM 360 серији рачунара. Велики софтверски пројекти били су и извор сазнања да је развој великих софтверских система суштински различит од програмирања у мањем обиму. Скалирање техника развоја малих компјутерских програма на развој великих софтверских система није давало резултате. Термин „софтверско инжењерство" настаје управо у ово време. Организован је и велики број стручних конференција на којима је расправљано о тешкоћама са којима су се суочавали овакви пројекти. Велики софтверски пројекти редовно су премашивали предвиђена инвестициона средства, и пробијали предвиђене рокове. Тако је настао још један за рачунарство значајан термин – „криза софтвера".

На пројектима тог обима дошло се до сазнања да проблем развоја великих софтверских пројеката није само у томе да се поређају програмске наредбе. Проблеми који су решавани нису били најбоље схваћени, бар не од свог особља укљученог у пројекат. Људи који су радили на пројекту били су принуђени да велики део времена троше на међусобна разјашњавања и усаглашавања, уместо да пишу код. Други проблем настајао је са људима који су напуштали пројекат, што се одражавало не само на посао који су они обављали већ и на послове људи који су били зависни од њих. Замена појединца захтевала је интензивну обуку нових чланова развојног тима и информисање о пројектним захтевима и „специфичностима" решења која су већ изведена и реализована. Промене у пројектним захтевима тицале су се многих делова пројекта и захтевале одговарајуће измене, што је често био узрок кашњења у испоруци. Проблеми овакве врсте једноставно нису постојали у раним „програмерским" данима и захтевали су потпуно нови приступ.

Предложено је много различитих решења и било је много покушаја да се унапреди процес развоја софтвера. Неки су сугерисали да је решење у бољем менаџменту, други су потенцирали да је кључна организација развојног тима, за треће је

решење било у коришћењу бољих и изражајнијих језика и алата. Многи су захтевали употребу стандарда у развојним организацијама, као, на пример, униформне конвенције за кодирање. Идеја није недостајало. Коначно се дошло до закључка да при развоју великих софтверских система треба примењивати исти приступ као и при изради других сложених система и објеката стварног света, као на пример зграда, авиона и електронских уређаја. Идеја је била да се реализовани софтверски систем посматра као сложени производ и да израда тог производа представља инжењерски посао. Инжењерски приступ је захтевао менаџмент, организацију, алате, теорију, методологије и технике.

Поред развоја програмирања, и други технолошки трендови дали су допринос развоју софтверског инжењерства. Најзначајнији утицај остварила је промена односа цене хардвера и софтвера. Цена ранијих рачунарских система била је одређена углавном ценом инсталираног хардвера, док је софтвер представљао ставку без значаја. У данашње време цена софтвера чини бар половину цене рачунарских система. Опадајућа цена хардвера и растућа цена софтвера довеле су до повећаног економског интереса за софтвер и укључивања индустрије у процес његовог развоја.

Успешно управљање софтверским развојем и његовим коришћењем у непосредној је вези са сталним процењивањем трошкова и њиховим свођењем на минималну меру за одређени квалитет софтвера. Како је утицај трошка у појединим фазама животног циклуса софтвера различит, разликују се и дефинисане метрике које имају за циљ да квантификују ове факторе.

Поставља се питање процене стварног облика напора потребног за развој програма, као и трошкова и недостатака, јер су ти квантитети потребни за ефективно управљање софтверским развојем.

Постоји више јединица којима можемо мерити напор потребан за развој програма. Домет из Halstead-ових елементарних менталних дискриминација је стварно време у минутама за програмера који обавља посао. Заједничка мера рада је људи/месеци. Мада је појам напора обично мерен у времену, важан, прави смисао метрике напора је стварни трошак. Чињеница је да су прихватљиви облици напора и трошка важни, а у управљању не могу бити довољно наглашени. Сазнања о напору и трошку су витални за управљање у таквим областима, као што су: профитност, изводљивост пројекта и продуктивност програмера. Ако не знамо колико напора и трошка је било потребно на прошлим пројектима различите величине и функције, не можемо направити разумну интуитивну процену наступајућег пројекта (Bailey, et al., 1981).

Такође, не бисмо били у могућности да користимо већину понуђених модела напора ако немамо сазнања о историјским подацима о унутрашњости организације. Прецизност мере је генерално одређена типом апликације којој је намењена. Мерење напора и трошка разликује се у степену прецизности, зависно од намене. Мали пројекти могу бити мерени у данима или сатима, а велики пројекти у програмер/месец или програмер/година. Јединице и метод мерења напора често се деле на две категорије: микро ниво мерења које су појединачни програмери утрошили на малим пројектима и макро ниво мерења напора тима програмера на великим пројектима.

Неки истраживачи бавили су се проблемом процене трошка на нивоу пројекта или макро нивоу, док су се други оријентисали на микро ниво, односно ниво самих програма. Заједничко им је да трошак исказују преко напора уложеног у пројектовање, одно-

сно програмирање, уз укључење општих (пратећих) трошкова. Иако је развијено много модела за процену напора и трошка, не постоји јединствен модел који је погодан за све случајеве развоја софтвера и у свим развојним окружењима (Banker, et al., 1994).

Процена пројеката у процесу развоја софтвера

Софтверске организације данас користе различите приступе проценама. Модел процеса процењивања може се посматрати из три различите перспективе. То су:

– *процене и софтверски процес.* Која је улога процена у софтверском процесу?

– *улазне и излазне величине при процењивању*. Који су подаци на располагању процењивачу и које информације се добијају проценом?

– *сам процес процене*. Како процењивач користи доступне информације да би извршио процену?

Процене и процес развоја софтвера

Процене пројеката врше се из различитих разлога. Разлог због којег се врши процена је важан чинилац у одређивању када и како се процена обавља. Разлози због којих се процене врше могу бити (Barbara, et al., 2010):

– *одобрење пројекта.* Мора постојати одлука организације о отпочињању пројекта. Да би се таква одлука донела, потребно је имати процене новца и ресурса потребних да се пројекат успешно доврши;

– *управљање пројектом*. Руководиоци пројекта одговорни су за планирање и управљање пројектом. Обе ове активности захтевају процене неопходних поступака и ресурса да се пројекат комплетира;

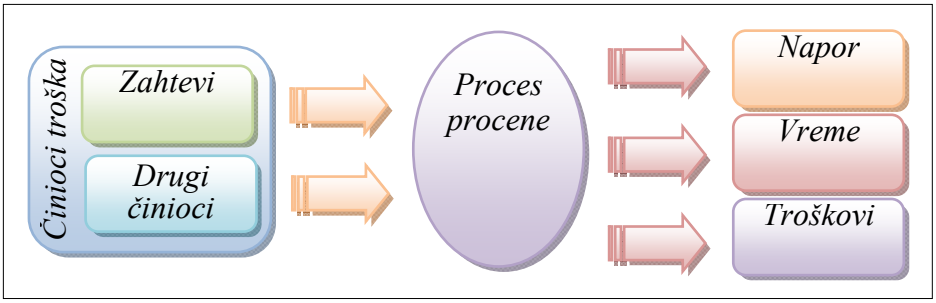
– *задужење чланова развојног тима.* Да би развојни тим радио ефикасно, потребно је да сваки члан тима разуме своју улогу, као и укупне активности на нивоу тима. Дефинисање пројектног задатка, који се може користити за ову сврху, врши се уз помоћ процена.

Одговор на питање зашто се врши процена може бити било који од понуђених. Он одређује када се врши процена. Одобрење пројекта захтева да се процена врши веома рано, најчешће пре него што су захтеви јасно дефинисани. Поступак одобравања пројекта обично садржи неколико тачака када треба донети одлуку да ли наставити даље или одустати. У свакој од ових тачака често се захтевају процене, које би руководећи тим снабделе информацијама на основу којих би донели исправну одлуку. У овој фази животног циклуса развоја софтвера процене прилично су непрецизне због недовољног броја информација које стоје на располагању процењивачу. У каснијим фазама процене постају тачније и руководство може кориговати своје одлуке.

За управљање и разумевање пројекта прва процена врши се у раним фазама животног циклуса софтвера, а касније се стално раде нове процене све до краја рада на пројекту. Како пројекат одмиче, процењивачима стоји на располагању све више информација и процене постају све прецизније.

Улазне и излазне величине у процесу процене

Процес процене софтвера састоји се у одређивању скупа излазних величина у функцији од скупа улазних величина. Улазне величине зависе од тога када се процена врши. Ране процене се, по правилу, врше на основу некомплетних и непрецизних података о пројекту и процесу развоја. Прелиминарне процене потребне су пре него што су дефинисани захтеви и архитектура система.



Слика 1 – *Класичан модел процене у софтверу*
(извор: аутори)

Такве процене засноване су на непотпуним подацима и од њих се не може очекивати висок степен тачности. Са друге стране, процене које се обављају у завршним фазама животног циклуса софтвера засноване су на много детаљнијем познавању пројекта и самим тим су тачније од прелиминарних (Briand, et al., 1998).

Према класичном становишту, напор, време и трошкови потребни за реализацију пројекта израчунавају се из величина које се често називају *чиниоцима трошка*. То су својства система која утичу на крајњу цену (време, напор) производа. У већини развијених метода процене, за примарни чинилац трошка узимају се софтверски захтеви, а за секундарне искуство особља, сложеност хардвера, итд. На основу ових чинилаца врши се процена излазних величина.

У ствари, процес процене софтвера може бити много сложенији него што је то приказано на слици. Постоји међусобна зависност између многих чинилаца релевантних за процес процена у софтверу. Често је тачније посматрати процес процена као покушај да се задовољи скуп ограничења, него као функцију захтева.

Улаз у систем је скуп ограничења захтева, софтверске архитектуре, финансијских ресурса, итд., а излаз представљају резултати процене и скуп поставки које задовољавају ограничења (Boehm, et al., 2004 p 156).

Метрике у софтверском инжењерству

Једна од најважнијих карактеристика традиционалних инжењерских дисциплина јесте постојање прецизних и лако разумљивих мера. На пример, већина људи добро познаје и разуме мере за величине као што су притисак, температура, елек-

трични напон, дужина, површина или запремина. Постоји и велики број мера које су практично непознате онима који се професионално не баве конкретним инжењерским дисциплинама. Тако, на пример, у механици флуида постоје мере као што су Reynolds-ov, Mach-ov, Pechlet-ov број и многе друге. У свакој инжењерској и научној дисциплини постоји велики број метрика које стоје на располагању истраживачима и инжењерима. Све те различите метрике имају неке заједничке особине: оне су прецизне, јасне, стандардизоване и прихваћене од научне јавности. Осим тога, све оне су се развијале и усавршавале постепено, низом научних истраживања и испитивања. Процес замене постојећих метрика новим и савршенијим и данас је у току.

Аналогија је јасна: софтверском инжењерству потребне су метрике и намеће се мисао да оне треба да настану кроз сличан процес истраживања и унапређивања. Нажалост, аналогија није потпуна: класичне инжењерске дисциплине извиру из природних наука: физике, хемије, механике, итд., док је предмет изучавања софтверског инжењерства софтвер, који је и сам производ људског ума. Основни проблем код мерења у софтверском инжењерству не представља недостатак метрика; напротив, изгледа да је развијен превелики број метрика које се често користе без суштинског разумевања. Како се ствари развијају, изгледа да време не доноси решења у овој области, већ би се пре рекло да временом проблем мерења у области софтвера постаје све присутнији и све болнији (Briand, et al., 1999 п 313).

Мера поузданости неке метрике требало би да покаже колико поверења можемо имати у резултате мерења, зависно од променљивости услова у којима се оно обавља. Другим речима, поставља се питање у којој мери поновљена мерења једне исте појаве дају исте резултате. Поузданост метрике може бити доведена у питање од више врста променљивости (Jeffery, et al., 2000 п 109):

– променљивост посматрача односи се на могућност да особа која врши мерење неће увек протумачити исте појаве на исти начин. Ово није проблем код директних метрика, али где год се јавља нека субјективна процена, поузданост метрике може знатно опасти;

– субјективна променљивост односи се на променљиве карактеристике људи који раде на развоју софтвера. У току времена људи напредују у коришћењу појединих алата и техника, али, са друге стране, и заборављају многе ствари којима се дуже или краће време нису бавили;

– променљивост окружења односи се на неконтролисане појаве које могу утицати на мерење.

Валидност метрике показује колико она мери оно што се од ње очекује. Можемо посматрати више врста валидности метрика:

– валидност садржаја односи се на питање да ли се прикупљени подаци заиста односе на оно што желимо да измеримо;

– валидност предвиђања показује корелацију између резултата мерења и неког критеријума везаног за будуће карактеристике мерене појаве;

– конструктивна валидност узима се у обзир код индиректних метрика. Таква мерења заснивају се на постојању одговарајућих релација између директно измерене величине и оне која нас заиста интересује. Конструктивна валидност доказује се показивањем да такве релације заиста постоје.

Важно је нагласити да су поузданост и валидност метрика две међусобно независне категорије, односно да једна метрика може бити врло поуздана, али мало валидна и обрнуто.

Најпожељније метрике су оне које представљају директну меру посматране карактеристике. Нажалост, такве мере су, у свету софтвера, врло ретке: често нису доступне у време када су најпотребније, а неретко нема начина да извршимо директно мерење. Као последица тога, ограничавамо се на употребу индиректних мера. Када прихватимо употребу индиректних мера падамо у зависност од аналитичког или емпиријског модела система или процеса који меримо (Briand, et al., 1992 p. 931).

Кад год је метрика индиректна (базирана на моделу) постављају се два питања у вези са њеном применом. Прво питање је валидација модела на којем је метрика заснована, а друго одређивање употребне вредности података који се добијају применом метрике. Код директних метрика поставља се само друго питање.

Део поступка валидације предложене метрике требало би да буде поређење резултата њене примене са резултатима примене постојећих метрика. Нова метрика требало би да покаже предност у односу на старе, макар у једној димензији мерења (прецизност, лакоћа примене, итд).

Код постављања стандарда за метрике, све мере се не могу посматрати на исти начин. Неке мере заснивају се на анализи програмског кода и оне су детерминистичке. Друге се заснивају на подацима прикупљеним на основу посматрања извршења програма, мада строго узевши и оне могу бити детерминистичке, па често није практично да их посматрамо на тај начин. Има и много мера које се заснивају на људском понашању, а оне нису детерминистичке. У овим случајевима треба применити различите стандарде. Ако се, на пример, за оцену метрика користи коефицијент корелације, треба очекивати веће вредности овог коефицијента за детерминистичке него за не детерминистичке мере.

Преглед литературе

De Marco је предочио да се функционалност софтверског система може одредити из компоненти структурне анализе (SA). За потребе мерења De Marco класификује системе у следеће три групе: системе засноване на функцијама, системе засноване на подацима и хибридне системе. Класификација је сачињена на основу односа RE/FP, где је RE број веза у моделу података, а FP представља број примитивних функција (примитивни процеси у SA дијаграму). Ако је однос мањи од 0,7 систем је заснован на функцијама, а ако је однос већи од 1, 5 систем је заснован на подацима. Метрика функционалности за системе засноване на функцијама, названа *Function Bang,* израчунава се на основу сложености DFD дијаграма и типа операција над датим дијаграмима. (De Marco, 1972).

Thomas McCabe је предложио метрику комплексности софтвера која је заснована на графу контроле програма. *Graf kontrole* је граф чији су чворови базични блокови (низ инструкција без гранања), а гране представљају токове контроле (гранања) програма. Метрика је дефинисана као *цикломатичка комплексност*B(Г) графа контроле одређеног модула. Један начин да се израчуна V(G) јесте да се одреди број региона у планар-

ном графу контроле. Регион се може описати као затворена површина планарног графа, ограничена чворовима и гранама графа. Број региона јесте заправо број ограничених површина, плус неограничена површина око графа (T.J.McCabe, 1976).

Halstead-ова теорија „софтверске науке" је прва аналитичка теорија софтвера. Без обзира на чињеницу да Halstead-ови закони „софтверске науке" по мишљењу многих стручњака не важе, значај ове теорије је огроман, управо у смислу покушаја дефинисања аналитичких једначина и закона који би се примењивали на софтверске системе. Halstead-ова теорија је изведена под претпоставком да „људски мозак следи много ригиднија правила у развоју алгоритама него што смо тога свесни". „Софтверска наука" користи скуп примитивних мера изворног кода, које могу бити добијене након што је програмски код генерисан.

Halstead је поставио хипотезу да је *ниво језика* константа за дати програмски језик. Неки каснији радови су га демантовали и показали да је *ниво језика* зависан и од имплементатора. Он је у свом раду извео и неке нивое језика на основу експерименталних података (Halstead, 1977).

Дефиниције

Софтвер је у данашње време постао ограничавајући фактор у развоју рачунарства, услед немогућности софтверских инжењера да овладају сложеношћу софтверских система који представљају системе стварног света. Сама пракса коју називамо софтверским инжењерством још увек није постала права инжењерска дисциплина, с обзиром на то да многа подручја остају још увек недовољно добро дефинисана. Још увек је велика употреба неформалних метода и техника: не постоји општеприхваћени начин ни да се специфицира шта то софтверски систем треба да ради, а још је приметније одсуство мера и величина које се могу користити за потребе техничких спецификација и оцену квалитета софтверских система.

Многи проблеми у развоју софтвера, настали у раним данима, опстају и данас, иако је и на пољу програмирања дошло до великог напретка, почевши од студија алгоритама и структура података, па све до открића и употребе „структурираног програмирања". Термин „софтверско инжењерство" почиње да се користи касних 60-их, када је коначно схваћено да све лекције „како да се добро програмира" не помажу много и не дају за резултат софтвер бољег квалитета. Уз све то, проблеми који су добили своје софтверско решење у ранијим фазама развоја софтверске технике у највећем броју случајева могу се сврстати у групу „добро дефинисаних" проблема, па се посао програмера најчешће сводио на кодирање познатог и добро дефинисаног алгоритма. У данашње време то готово редовно није случај.

Шта је заправо „софтверско инжењерство"? То је област рачунарске технике која се бави израдом софтверских система који су тако велики и сложени да их редовно развија тим инжењера или, често, тимови инжењера. Парнас је дефинисао софтверско инжењерство као „*конструисање вишеверзионог софтвера од стране већег броја људи*". Ова дефиниција садржи суштину софтверског инжењерства и истиче разлику између програмирања и софтверског инжењерства: програмирање је превасходно персонална активност, док је софтверско инжењерство, у својој основи, тимска активност (Parnas, 1978).

Бауер је дефинисао софтверско инжењерство као успостављање и коришћење чврстих инжењерских принципа како би се на економичан начин добио софтвер који је поуздан и ради ефикасно на стварним машинама (Bauer, 1979).

Модели података

Модел података представља интелектуално средство помоћу којег се описују статичке карактеристике система, опис карактеристика система у неком стационарном стању (Лазаревић, 1993). Стационарно стање неког система карактерише се скупом зависности које постоје између објеката система. Ове зависности се, у моделу података, могу представити било структуром података, било скупом ограничења на вредности података. Поред тога, неопходно је и дефинисати скуп операција модела података, како би се преко њих, у моделима процеса, могла описати и динамика реалног система. Модел података обезбеђује интерпретацију података о посматраном реалном систему. Она се у моделу података остварује кроз три његове основне компоненте:

1. структуре података, преко које се представљају статичке карактеристике система;
2. ограничења – логичка ограничења на податке, која у сваком тренутку посматрања (стационарном стању) треба да буду задовољена. Ова додатна ограничења на податке која нису обухваћена само структуром називају се и правилима интегритета модела података;
3. операције над концептима структуре модела, под дефинисаним ограничењима, преко којих је могуће описати динамику система у моделу процеса.

Информација може бити регистрована ради комуникације са другима, при чему је важно да буде подесно представљена и саопштена. Примарно, репрезентација информација врши се коришћењем природног језика, који нису увек најбољи алати. Природни језици не могу имати и немају прецизно значење у регистровању и преносу информација (Jefhrey, 1980).

Ради решавања проблема, који настају коришћењем природних језика за представљање информација у рачунару, појавили су се модели података. Они су снажан концептуални алат за оцртавање структуре и представљање информација и преводиви су у структуре којима се може манипулисати рачунаром. Помажу у постизању разумевања података и информација, и могу бити коришћени за опис. Сходно томе, моделирање података је важна активност у интеграцији информација.

Неки модели података настали су као природна апстракција система за управљање базама података. Они су концептуални и апстрактни део базе података насупрот имплементацији која представља инжењерски део.

Метрике у моделовању података

Модели података такође представљају поједностављену представу релевантних карактеристика неког реалног система. То значи да један реалан систем може бити представљен са више различитих модела података, зависно од сагледавања и дефинисаних циљева пројектанта. Имајући то у виду, у дефинисању метрика мо-

дела података поћи ћемо од принципа да оне морају бити применљиве на било који модел података.

Ради процене трошкова, односно времена потребног за реализацију базе података на основу модела података, времена неопходног за реализацију трансакција потребно је дефинисати најмање две групе метрика:

• I метрике структуре модела података,

• II метрике ограничења.

У даљем истраживању ограничићемо се на концепте модела објекти–везе и релационог модела и на њима покушати дефинисати метрике.

Метрике структуре модела података

Структуре модела података, односно међусобно повезивање података, представља базни модел њихове интерпретације. Структура у моделу објекти–везе дефинише се помоћу четири концепта: ентитета. везе, атрибута ентитета или везе и вредности. Полазећи од тих основних концепата, увешћемо базне метрике и, на основу њих, дефинисати изведене метрике, међу којима су посебно значајне величина модела података и сложеност модела података.

Како се модел података објекти–везе обично представља дијаграмом објекти–везе (DOV), вршећи његову анализу дошли смо до базних метрика које произилазе из основних концепата којима се он формира. У складу с тим, дефинисали смо базне метрике:

EJ – број јаких типова ентитета,

ES – број слабих типова ентитета,

EM – број мешовитих типова ентитета,

V – број типова веза,

AEJ – број атрибута јаких типова ентитета,

AES – број атрибута слабих типова ентитета,

AEM – број атрибута мешовитих типова ентитета.

Вредност базних метрика одређује се једноставним бројањем појављивања појединих концепата у дијаграму објекти–везе, при чему треба водити рачуна да због обезбеђења прегледности на дијаграму није дошло до појављивања истих типова објеката или веза (Живадиновић 2000).

На основу базних метрика, корисно је дефинисати следеће метрике:

- укупан број типова ентитета у моделу података:

$$E = EJ + ES + EM \tag{5.1}$$

- укупан број атрибута јаких типова ентитета:

$$AEJ = \sum_{i=1}^{EJ} AEJi \tag{5.2}$$

- просечан број атрибута јаких типова ентитета:

$$\overline{AEJ} = \frac{\sum_{i=1}^{EJ} AEJi}{EJ} \tag{5.3}$$

- укупан број атрибута слабих типова ентитета:

$$AES = \sum_{i=1}^{ES} AESi \tag{5.4}$$

- просечан број атрибута слабих типова ентитета:

$$\overline{AES} = \frac{\sum_{i=1}^{ES} AESi}{ES} \tag{5.5}$$

- укупан број атрибута мешовитих типова ентитета:

$$AEM = \sum_{i=1}^{EM} AEMi \tag{5.6}$$

- просечан број атрибута мешовитих типова ентитета:

$$\overline{AEM} = \frac{\sum_{i=1}^{EM} AEMi}{EM} \tag{5.7}$$

- укупан број атрибута у моделу података:

$$A = \sum_{i=1}^{EJ} AEJi + \sum_{i=1}^{ES} AESi + \sum_{i=1}^{EM} AEMi \tag{5.8}$$

- просечан број атрибута у моделу података:

$$\overline{A} = \frac{\sum_{i=1}^{EJ} AEJi + \sum_{i=1}^{ES} AESi + \sum_{i=1}^{EM} AEMi}{E} \tag{5.9}$$

Једна важна карактеристика модела података је његова величина. Она се коришћењем базних метрика најједноставније може дефинисати као:

$$VM = E + V \tag{5.10}$$

При утврђивању величине модела података преко метрике VM занемарује се улога концепта у моделу података и посматра се само његово појављивање.

Речник података информационог система је база података о самом систему. Он је од посебног значаја за сваки систем и треба да омогући складан и постепен развој целог система, да повеже различите технолошке типове по систем и да буде „водич" за све нове и повремене кориснике система. Користећи уведене метрике, могуће је дефинисати минималан волумен речника система RM као:

$$RM = VM + A \tag{5.11}$$

Метрика RM служи за процену глобалног обима система.

Да бисмо проценили ресурсе који се троше у реализацији базе података информационог система, увешћемо метрику сложеност модела података. То је функција броја ентитета, броја веза и броја атрибута. Ако посматрамо глобални модел података који садржи типове ентитета, типове веза и типове атрибута, тада функцију сложености SM можемо изразити као:

$$SM = f(E, V, A) \tag{5.12}$$

Ако се поједностављено узме у обзир сваки од концепата и затим дефинишу одређени тежински фактори, тада се сложеност модела података може представити као линеарна функција:

$$SM = tj \times EJ + ts \times ES + tm \times EM + tv \times V + ta \times A \tag{5.13}$$

И овде је, као и при дефинисању метрике сложености модела процеса, тешко одредити вредности тежинских фактора. На основу нашег искуства, вредност тежинског фактора додељеног типовима мешовитих објеката треба да буде већа од вредности осталих тежинских фактора.

При дефинисању метрике SM није узета у обзир кардиналност веза која има велики утицај на сагледавање сложености система представљеног анализираним моделом података. Имајући то у виду, а посебно утицај кардиналности веза на очување интегритета базе података, могуће је дефинисати метрику која ће одразити сложеност веза. Уколико се користи техника цртања DOV, односно типови веза који приликом превођења модела објекти–везе у релациони модел постају релације означене мешовитим објектом, тада ће једначином (6.40), на неки начин, бити обухваћена и кардиналност веза. Јасно је да се из тако дефинисаног DOV може одредити број релација BR у моделу на основу вредности наведених метрика и да је он једнак броју типова ентитета, односно:

$$BR = E \tag{5.14}$$

У пракси је модел података обично представљен преко више подмодела. Подмоделом је, у већини случајева, обухваћен један процес. Коришћење методологије моделирања са налажењем локалних подмодела омогућава лакше документовање, јер се цртање DOV за један модел обично врши на једној страни А4 формата.

Уколико пођемо од тога да је модел података неког система представљен преко подмодела тада можемо, аналогно дефинисаним метрикама, дефинисати и метрике на нивоу подмодела података (Живадиновић и др. 2013):

EJk – број јаких типова ентитета у к-том подмоделу,

ESk – број слабих типова ентитета у к-том подмоделу,

EMk – број мешовитих типова ентитета у к-том подмоделу,
Bk – број типова веза у и-том подмоделу,
m – број подмодела.

- Укупан број типова ентитета у к-том подмоделу података:

$$Ek = EJk + ESk + EMk \tag{5.15}$$

- Величина к-тог подмодела:

$$VMk = Ek + Vk \tag{5.16}$$

- Укупан број атрибута јаких типова ентитета у к - том подмоделу података:

$$AEJk = \sum_{i=1}^{EJk} AEJki \tag{5.17}$$

- Просечан број атрибута јаких типова ентитета у к-том подмоделу података:

$$\overline{AE}Jk = \frac{\sum_{i=1}^{EJk} AEJki}{EJk} \tag{5.18}$$

- Укупан број атрибута слабих типова ентитета у к-том подмоделу података:

$$AESk = \sum_{i=1}^{ESk} AESki \tag{5.19}$$

- Просечан број атрибута слабих типова ентитета у к-том подмоделу података:

$$\overline{AE}Sk = \frac{\sum_{i=1}^{ESk} AESki}{ESk} \tag{5.20}$$

- Укупан број атрибута мешовитих типова ентитета у к-том подмоделу података:

$$AEMk = \sum_{i=1}^{EMk} AEMki \tag{5.21}$$

- Просечан број атрибута мешовитих типова ентитета у к-том подмоделу података:

$$\overline{AE}Mk = \frac{\sum_{i=1}^{EMk} AEMki}{EMk} \tag{5.22}$$

- Укупан број атрибута у к-том подмоделу података:

$$Ak = \sum_{i=1}^{EJk} AEJki + \sum_{i=1}^{ESk} AESki + \sum_{i=1}^{EMk} AEMki \tag{5.23}$$

- Просечан број атрибута у к-том подмоделу података:

$$\overline{Ak} = \frac{\sum_{i=1}^{EJk} AEJki + \sum_{i=1}^{ESk} AESki + \sum_{i=1}^{EMk} AEMki}{Ek} \tag{5.24}$$

- Сложеност к-тог подмодела:

$$SMk = tj \times EJk + ts \times ESk + tm \times EMk + tv \times Vk + ta \times Ak \tag{5.25}$$

Пошто се у више локалних подмодела може појавити исти тип ентитета или тип везе, произилази да важе неједнакости:

$$E \leq \sum_{k=1}^{m} Ek \tag{5.26}$$

$$VM \leq \sum_{k=1}^{m} VMk \tag{5.26}$$

$$SM \leq \sum_{k=1}^{m} SMk \tag{5.27}$$

Метрике ограничења

Ограничења у моделу података су неопходна из семантичких разлога и због очувања интегритета базе података. S тачке гледишта семантике, помоћу њих се много прецизније може описати реалан систем, док се помоћу интегритета омогућава одржавање дозвољених стања базе података.

Метрика која оцењује сложеност модела са становишта трансакција преко ограничења на дефинисаном структуром модела података за задату трансакцију Ти дефинише се као:

$$ST = \sum_{i=1}^{n} Ti \tag{6.1}$$

где је Ti – број (објеката и агрегираних веза) који је дефинисан простирањем дејства трансакције ажурирања, околина кореног објекта трансакције, а n – број апликација, независних улаза или трансакција ажурирања.

Ако аналогно као код модела података дефинишемо сложеност трансакција за к-ти подмодел као:

$$STk = \sum_{i=1}^{nk} Tik \tag{6.2}$$

тада произилази да важи једнакост:

$$ST = \sum_{k=1}^{m} STk \cdot \tag{6.3}$$

Савремена реализација модела података употребом софтвера за управљање релационом базом података, типа „ORACLE", даје могућност процене времена неопходног за израду свих апликација коришћењем метрике SM. Уколико се располаже софтверским алатима, као што је генератор интерактивних апликација, може се извршити процена параметара процеса израде апликација са становишта ресурса.

Апликација је скуп повезаних екранских форми преко којих је могуће интерактивно ажурирање и извештавање над базом података. Апликацију чини више блокова од којих сваки одговара једној табели базе података над којом је дефинисан. Поља унутар блока одговарају или колонама табеле базе података над којом је блок дефинисан или колонама повезаним са табелом над којом је блок дефинисан или су то контролна поља. Преко једне екранске форме може се дефинисати један или више блокова. Време потребно за израду једне апликације је функција *Tu* и броја поља, при чему је:

$Ti \leq$ број блокова апликације.

Лако се показује да између сложености трансакција и у укупном броју ентитета важи однос

$$E \leq St \leq E^{2} \tag{6.4}$$

а између сложености трансакција и броја апликација:

$$ST \geq n \tag{6.5}$$

Полазећи од наведене дефиниције апликације и правила превођења модела објекти–везе у релациони модел произилази да између броја типова објеката и броја апликација увек важи однос

$$n \geq E \tag{6.6}$$

Може се закључити да је коришћењем уведених метрика могуће извршити процену одређеног броја параметара неопходних за планирање и управљање при реализацији пројеката. Основна карактеристика уведених метрика јесте да се оне, за систем који се анализира, могу одредити за време фазе пројектовања, односно након израде модела података и, према томе, самим тим користити у осталим фазама. Имајући у виду основне концепте на којима су базиране, њихово израчунавање може се аутоматизовати уградњом одговарајућих процедура у CASE алате.

Анализа практичне примене

Практичну примену уведених метрика приказаћемо на примерима локалних подмодела података информационог система Војске Републике Србије.

За ову анализу узето је из информационог подсистема Праћење кадрова Војске Републике Србије (BRS) осам подмодела:

ПМ1 – Припадник BRS

ПМ2 – Лични подаци припадника BRS

ПМ3 – Образовна организација

ПМ4 – Награде и казне

ПМ5 – Стамбено обезбеђење

ПМ6 – Стручно и научно оспособљавање

ПМ7 – Формације и постављења

ПМ8 – Унапређење ПВЛ

При израчунавању сложености подмодела података тежинским факторима су додељене вредности:

$\tau_j = 3$; $\tau_c = 1$; $\tau_м = 4$; $\tau_в = 2$; $\tau_а = 0{,}5$

И овим тежинским факторима додељене су вредности на основу искуства аутора. Те вредности би, кроз праксу, свакако требало кориговати, како би се добили што реалнији односи утицаја појединих величина на сложеност модела података.

Табела 7.1. – *Вредности метрика ентитета и веза*

Број подмодела	МЕТРИКЕ			
	EJк	ECк	EMк	Вк
ПМ1	7	-	2	3
ПМ2	5	-	1	4
ПМ3	5	-	-	4
ПМ4	4	2	-	6
ПМ5	4	-	2	5
ПМ6	5	1	1	6
ПМ7	3	-	2	3
ПМ8	6	2	4	11
за модел	39	5	12	42

Прегледом ових основних метрика, које се добијају набрајањем концепата са дијаграма објекти – везе, може се претпоставити да је подмодел ПМ8 највећи, а вероватно и најсложенији, јер садржи највећи број јаких, слабих и мешовитих типова

ентитета, а посебно веза. Са друге стране, подмодел ПМ7, на пример, изгледа да има мали број концепата и веза, што би могло да значи да је мали и не посебно сложен, у поређењу са осталим подмоделима. Како међу метрикама приказаним у табели 7.1. нема метрика броја атрибута, а и не види се утицај појединих величина на сложеност модела, закључујемо да ове базне метрике, саме по себи, тешко могу успешно да послуже за оцену сложености подмодела, односно модела података.

Табела 7.2. – *Вредности метрика броја атрибута ентитета*

Број подмодела	МЕТРИКЕ			
	AEJк	**AECк**	**AEMк**	**Aк**
ПМ1	24	-	3	27
ПМ2	13	-	1	14
ПМ3	16	-	-	16
ПМ4	14	4	-	18
ПМ5	15	-	4	19
ПМ6	15	2	2	19
ПМ7	11	-	8	19
ПМ8	14	6	7	27
за модел	122	12	25	159

Табела 7.3. – *Вредности изведених метрика атрибута*

Број подмодела	МЕТРИКЕ			
	$\overline{AEJk}$	$\overline{AESk}$	$\overline{AEMk}$	$\overline{Ak}$
ПМ1	3.4	-	1.5	3
ПМ2	2.6	-	1	2.3
ПМ3	3.2	-	-	3.2
ПМ4	3.5	2	-	3.3
ПМ5	3.7	-	2	3.2
ПМ6	3.0	2	2	2.7
ПМ7	3.7	-	4	2.8
ПМ8	2.3	3	1.7	2.2

У табелама 7.2 и 7.3 приказане су вредности метрика које се односе на атрибуте. Највише атрибута имају подмодели ПМ1 и ПМ8, а просечно по ентитету ПМ4. Гледано само преко атрибута, подмодели ПМ1 и ПМ8 били би највећи, а подмодел ПМ4 најсложенији. Овај закључак не одговара разматрањима о величини и сложености подмодела мерених преко ентитета и веза.

Табела 7.4. – *Вредности изведених метрика у моделу података*

Број подмодела	МЕТРИКЕ			
	Ек	ВМк	РМк	СМк
ПМ1	9	12	39	48
ПМ2	6	10	24	34
ПМ3	5	9	25	31
ПМ4	6	12	32	36
ПМ5	6	11	30	39
ПМ6	7	13	32	41
ПМ7	5	8	22	30
ПМ8	12	23	50	71

Из табеле 7.4. види се да је највећи подмодел (ПМ8) уједно и најсложенији, а најмањи подмодел (ПМ7) најмање сложен. Међутим, редослед подмодела по величини и по сложености није у потпуности исти, што је и логично очекивати. Наиме, величина подмодела узима у обзир искључиво ентитете и везе, а занемарује атрибуте. Такође, метрика сложености узима у обзир да све компоненте немају подједнак утицај на сложеност модела, док са величином то није случај.

Посебно је интересантно посматрати укупан број ентитета, јер је он, као што смо већ напоменули, једнак броју релација у релационом моделу података који се добија из посматраног модела објекти – везе. Сматра се, наиме, да је број човек/дан потребних за израду апликација за ажурирање база података једнак броју релација. Тако би нам за ПМ8 требало 12 човек/дан, а за ПМ7 – 5 човек/дан. На основу података из таблице 6.8, долазимо до закључка да један човек/дан одговара отприлике 6 SM. Наравно, то је крајње груба процена и свакако би требало доћи до прецизнијих веза између метрике сложености и времена израде.

Вредности базних метрика ентитета и веза на нивоу модела података дате су у табели 6.5, а метрика броја атрибута ентитета на нивоу модела података у табели 6.6. Остале метрике имају следеће вредности:

E=56, $\overline{AEJ}$ *=3.1,* $\overline{AES}$ *=2.4,* $\overline{AEM}$ *=2,* $\overline{A}$ *=2.8, BM=98, RM=257, SM=333.*

Јасно је, дакле, да је могуће квантификовати величину и сложеност модела података. Познавање ових величина у раној фази развоја софтвера (ове метрике могу се израчунати непосредно након цртања дијаграма објекти–везе) представља помоћ руководиоцима пројеката у процени расподеле времена, трошкова и напора у развоју база података информационих система.

Закључак

У раду су најпре наведене неке од до сада развијених софтверских метрика, а затим су развијене нове које се односе на моделе података, уз пример њихове примене на конкретном моделу. Такође, уведене су метрике у моделе података.

Дефинисане су и метрике модела података које служе за мерење карактеристика база података информационих система. Вредности ових метрика могуће је израчунати на основу нацртаних дијаграма токова података. Базне метрике добијене су бројањем ентитета, веза и атрибута са дијаграма тока података, док се изведене метрике изводе из базних, где издвајамо метрике величине и метрике сложености модела података.

Величина модела података је метрика којом квантификујемо обим модела података, не узимајући у обзир атрибуте и сматрајући да ентитети и везе имају подједнак утицај.

Сложеност модела података дефинисали смо тако што смо увели тежинске коефицијенте, којима, на известан начин, уравнотежавамо различит утицај ентитета, веза и атрибута на сложеност модела података и овде, за сада, више наслућујемо вредности тих коефицијената него што смо их извели из праксе. Вероватно ће се, кроз евентуалну употребу метрике сложености, доћи до прецизнијих вредности тежинских коефицијената од оних које смо предложили у овом раду. Ипак, изгледа да је исправна претпоставка по којој највећи утицај на сложеност модела података имају мешовити ентитети, затим јаки ентитети, везе, слаби ентитети и, најзад, атрибути.

На крају, дефинисане су метрике величине и сложености информационог система. Како су основни модели информационог система модели података, логично је било да се метрике које се односе на информациони систем дефинишу преко метрике модела података. Сврха ових метрика јесте да се преко њих врши непосредно поређење различитих информационих система по величини и сложености, као и да се у раној фази развоја информационог система стекне представа о ресурсима потребним за њихову реализацију.

Основни недостатак метрика модела података, дефинисаних у овом раду, јесте њихова недовољна провереност у инжењерској пракси. Тек након коришћења ових метрика током израде више различитих софтверских пројеката, можемо дати суд о њиховој ваљаности. Такође, провером на разним конкретним моделима дошло би се до начина за даље усавршавање предложених метрика.

Литература

[1] Bailey, J. W., and Basili, V.R. (1981), *A Meta-Model for Software Development Resource expenditures,* Proceedings of the 5th International Conference on Software Engineering, pp 107-116.

[2] Barbara Kitchenham and Pearl Brereton. (2010) *Problems Adopting Metrics from Other Disciplines*, Workshop on Emerging Trends in Software Metrics, May pp 1-7.

[3] Banker, R. D., Chang, H., and Kemerer, C. F. (1994), *Evidence of Economies of Scale in Software Development*, Information and Software Technology, vol 36 no 5, pp 275-282.

[4] Boehm B. W., Brown R., Madachy, and Yang Y. (2004) „A software product line life cycle cost estimation model," in *Proceedings of the International Symposium on Empirical Software Engineering* , pp. 156–164.

[5] Bauer C.D, *Principles of Software Project Management*, Adison-Wesley, 1979.

[6] Briand, L.C., El Emam, K., Surmann, D. and Maxwell, K. (1998), *An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques*, ISERN-Report-27-98.

[7] Briand L. C., Emam K. El, Surmann D., Wiezczorek I., and Maxwell K. D. (1999), „An assessment and comparison of common software cost estimation modeling techniques," in *Proceedings of the International Conference on Software Engineering*, pp. 313–323.

[8] Briand, L., Basili, V.R., and Thomas, W. (1992), *A Pattern Recognition Approach for Software Engineering Data Analysis*, IEEE Transactions of Software Engineering, vol 18 no 11, pp 931-942.

[9] De Marco, T. (1972), *Controlling Software Projects*, Yourdon Press, New York,.

[10] Halstead M., (1977), *Elements of Software Science*, North Holland.

[11] Jeffery, D.R., M. Ruhe, and I. Wieczorek. A, (2000), Comparative Study of Two Software Development Cost Modeling Techniques Using Multi-Organizational and Company-Specific Data. Information and Software , pp. 109-116.

[12] Jefhrey D.Ullman, „Principles of Database Systems", Computer Science Pres, 1980.

[13] Лазаревић Б., (1993), *Структурна систем анализа*, Лабораторија за информационе системе, Факултет организационих наука у Београду.

[14] T.J.McCabe, „A Complexity measure", IEEE Trans.Software Eng., vol.SE-2, No.12, 1976.

[15] Parnas D.L., (1978) „Some Software Engineering Principles", in Structured Analysis and Design, *State of the Art Report. INFOTECH International*.

[16] Тешић Н. Дејан (1996) *Метрике за процену сложености и квалитета објектно-орјентисаних софтверских система,* Магистарски рад.

[17] Живадиновић J., (2000) „Metrics for software program quality assessment "; SYMOPIS-2000, Book of Proceedings, Belgrade.

[18] Живадиновић J., Медић3., Јевтић Б., Гроздanић R., Пиљан I., (2013), *The Software Projects Development Improvment by Applying Metrics*; Metalurgia international vol. XVII, pp 149-158.