OCENJEVANJE KOMPLEKSNOSTI PROGRAMSKEGA PROCESA KOT OSNOVA ZA OCENJEVANJE OBSEŽNOSTI PROJEKTOV

Romana Vajde Horvat, Tomislav Rozman, Aleš Živkovič
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

Izvleček

Večina modelov za ocenjevanje obsežnosti posameznega programskega projekta temelji na ocenjevanju zahtevnosti programske opreme, ki bo razvita. Obstoječi modeli zato vključujejo predvsem atribute, na podlagi katerih ocenjujemo obsežnost same programske opreme. Kljub temu, da v zadnjih letih proces razvoja programske opreme dosega višji nivo zrelosti, modeli za ocenjevanje obsežnosti projektov tovrstnih podatkov ne upoštevajo dovolj. Vendar nam ravno ti podatki lahko pomagajo do realnejših ocen.

Podobno kot v drugih panogah, je tudi v razvoju programske opreme mogoče opredeliti tako imenovane procesne modele, ki opredeljujejo zaporedje aktivnosti, ki se izvajajo znotraj posameznega procesa. Hkrati z določitvijo zaporedja aktivnosti lahko v procesnem modelu opišemo tudi druge lastnosti procesa: delovne proizvode, ki so potrebni za izvajanje aktivnosti oziroma so rezultat aktivnosti, vire, ki so potrebni za njihovo izvedbo in podobno. Čim bolj podrobno poznamo procesni model, tem natančneje lahko določimo trud, ki bo potreben za izvedbo določenega projekta, ki se bo odvijal po izbranem procesnem modelu. V nadaljevanju je opisan model za ocenjevanje kompleksnosti procesnih modelov (SoPCoM - Software Process Complexity Model).

Abstract

Majority of existing software project cost and effort estimating tools are based on evaluation of the complexity of software product being developed. These models therefore emphasize attributes for software size estimation. Although the software processes themselves have achieved higher maturity, the cost and effort estimation models do not use the available information of its maturity enough. However, such information can provide more accurate estimations.

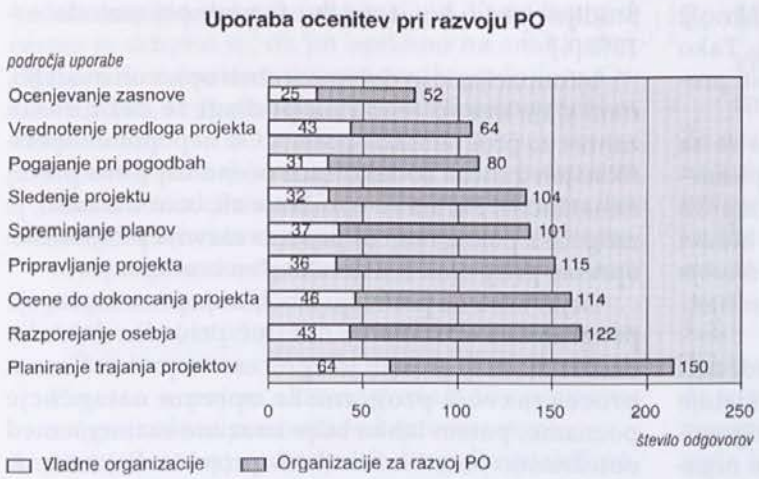
Similar to other types of processes also the software process can be modeled within process models, which describe the sequence of activities to perform. Other elements of the process can be also described: input/output working products of activities and resources needed to perform activities. Details known for each process model are the basis for estimating the effort of the project, which will be conducted according to the process model. In the article the SoPCoM (Software Process Complexity Model) which uses this approach is described.

Ključne besede: programska oprema, proces razvoja programske opreme, procesni modeli, kompleksnost procesov, Petrijeve mreže

■■■

1. Uvod

Lastnosti procesa razvoja programske opreme, kot so visoka stopnja *vplivnosti informacij* in *intelektualnega prispevka* pri izvajanju aktivnosti, *sočasnost* in *porazdeljenost* izvajanja aktivnosti, *negotovost* in *časovna nedoločljivost* poznega razporejanja nalog, *neprestana podvrženost spremembam* in druge lastnosti vplivajo na zahtevnost procesnih modelov za razvoj programske opreme. Vendar pravih modelov za izražanje zahtevnosti ***procesa razvoja programske opreme*** ni na razpolago. Večina obstoječih modelov za ocenjevanje kompleksnosti razvoja programske opreme se namreč posveča ocenjevanju kompleksnosti razvoja točno določenega ***proizvoda*** ali ***programske opreme*** same. To pomeni, da na podlagi karakteristik proizvoda, ki ga je potrebno razviti, ocenjujemo zahtevan čas (oziroma trud) in predvidene stroške za njegovo izvedbo. Ocenjujemo torej kompleksnost posameznega *projekta*, v okviru katerega razvijamo točno določeno programsko opremo. Tovrstno ocenjevanje je v praksi precej pogosto uporabljeno. V raziskavi, ki jo je



Slika 1. Področja uporabe ocenitev programske opreme

opravil Software Engineering Institute [1], je med 269 anketiranimi podjetji vsako izmed njih odgovorilo, da uporabljajo ocenjevanje, vendar v različne namene (glej sliko 1).

Vendar kljub temu, da različni viri za posamezne modele za ocenjevanje truda in stroškov navajajo sorazmerno veliko natančnost ocen na referenčnih podatkih, lahko ugotovimo, da v praksi te ocene še zdaleč niso tako natančne. Slika 2 namreč prikazuje, da so ocene stroškov in ocene urnikov skoraj vedno pravilne v manj kot petih procentih organizacij. Še bolj zaskrbljujoče je, da je skoraj polovica organizacij odgovorila, da so njihove ocenitve natančne za manj kot polovico projektov.

Najpomembnejša med modeli za ocenjevanje obsežnosti projektov razvoja programske opreme sta zagotovo model COCOMO (Constructive Cost Model) [2, 3] in Funkcijske točke [4, 5, 6]. Oba modela temeljita predvsem na ocenjevanju obsežnosti proizvoda in bistveno premalo upoštevata karakteristike samega *procesa* razvoja programske opreme.

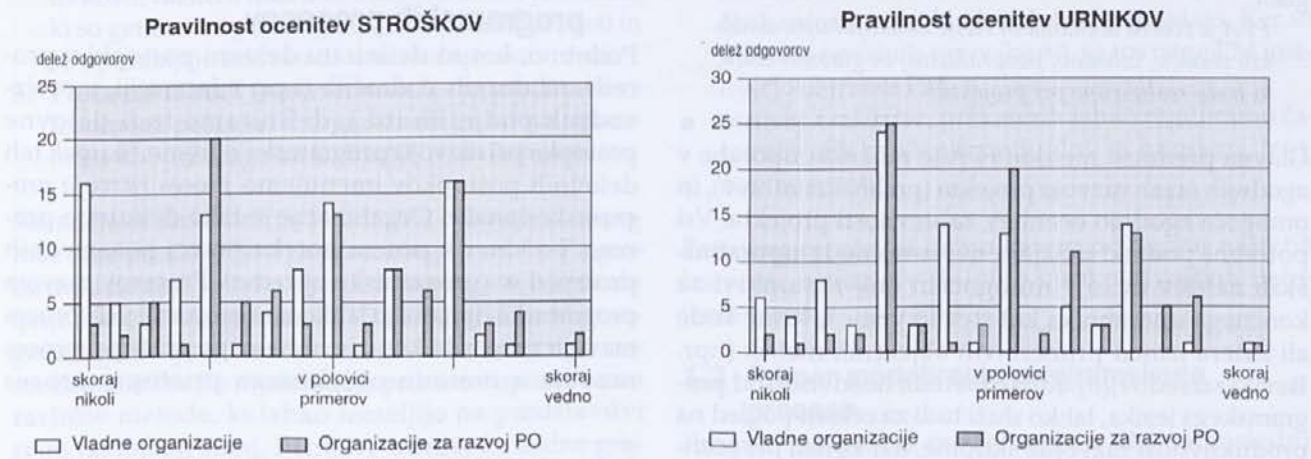
COCOMO 2.0

Osnovni COCOMO je bil razvit že leta 1981 in je od takrat doživel nadgradnje in izboljšave, s katerimi se je približal sodobnejšim pristopom k razvoju programske opreme. Najpomembnejše spremembe se odražajo v novi različici modela COCOMO 2.0. Ta model predvideva, da bo razvoj programske opreme v prihodnosti potekal na različne načine [3]:

1. *v sektorjih za gradnjo programske opreme*, kjer bo potekal celoten razvoj specifične programske opreme,
2. *v sektorjih za uporabo generatorjev programske opreme*, kjer bo razvoj potekal na podlagi vnaprej pripravljenih komponent in knjižnic posameznih proizvajalcev,
3. *v sektorjih za povezovanje sistemov*, kjer bo potekalo povezovanje velikih sistemov (ustrezna izmenjava informacij med njimi),
4. *v sektorjih za razvoj infrastrukture*, kjer bo potekal razvoj operacijskih sistemov, podatkovnih baz, upravljalnih sistemov, sistemov za upravljanje uporabniških vmesnikov, ipd.
5. *programiranje, ki ga bodo izvajali končni uporabniki sami* – sem spadajo predvsem preproste aplikacije, ki jih lahko razvijemo z uporabo v ta namen pripravljenih generatorjev, kot to poznamo v različnih programih za preglednice, povpraševanja v podatkovnih bazah in podobno.

COCOMO 2.0 je namenjen uporabi v prvih štirih sektorjih, v zadnjem primeru pa so aplikacije tako preproste (trajanje razvoja se meri v urah ali največ dneh), da uporaba modela ni smiselna.

V prvem sektorju (sektorju za gradnjo programske opreme) poteka ocenjevanje na podlagi tako imenovanih *objektnih točk*. Objektne točke izrazimo s številom zaslonov, poročil in modulov, napisanih v



Slika 2. Prikaz deležev pravilno ocenjenih stroškov in urnikov na projektih.

jeziku tretje generacije. Točke ustrezno utežimo z vrednostmi preprosta, srednja in zahtevna. Tako dobljene vrednosti nam dajejo oceno obsežnosti programske opreme, ki jo razvijamo.

V drugem, tretjem in četrtem sektorju pa se za ocenjevanje uporablja ustrezna kombinacija opisanega modela objektnih točk in pa predpisanih parametrov za enega izmed dveh modelov: model razvoja programske opreme *v zgodnji fazi načrtovanja* in model za ocenjevanje *v fazi po opravljenem arhitekturnem načrtovanju.*

Med parametri obeh modelov najdemo tudi takšne, ki so posvečeni zmožnostim osebja, ki sodeluje pri razvoju, izkušnjam pri razvoju podobne programske opreme, izkušnjam z razvojnim okoljem in organizaciji projekta. Vendar so ti parametri ocenjeni splošno za celoten projekt. Podroben opis izračuna obsežnosti in ocenjevanje posameznih faktorjev je opisan v virih [2] in [3].

Funkcijske točke

Albrecht [7] je pri razvoju FPA izhajal iz dejstva, da je oblikovanje in razvoj poslovnega informacijskega sistema odvisen od treh dejavnikov. Ti dejavniki so:

Obseg procesiranja informacij (*information processing size*) - količina informacij, ki jih procesira in oskrbuje sistem.

Faktor tehnične zahtevnosti (*techical complexity factor*) upošteva pomen različnih tehničnih in drugih faktorjev (npr. enostavnost uporabe, učinkovitost končnega uporabnika), ki vplivajo na razvoj in implementacijo zahtev za procesiranje informacij.

Razvojni dejavniki (*enviromental factors*) so skupina dejavnikov, ki izhajajo iz okolja projekta, in so odvisni od veščin, izkušenj in motivacije razvojne skupine ter od uporabljenih metod in orodij za razvoj.

Albrecht se je pri definiranju metode osredotočil na prva dva dejavnika, saj izhajata neposredno iz uporabnikovih zahtev. Poenostavljena definicija se glasi:

> *FPA je tvorba seznama in štetje zunanjih uporabniških vhodov, izhodov, povpraševanj in glavnih zbirk, ki bodo realizirane pri projektu.*

Glavna prednost metode FPA je možnost uporabe v zgodnjih fazah razvoja projekta (pri analizi zahtev), in omogoča zgodnjo ocenitev zahtevnosti projekta. Vsi potrebni podatki izhajajo neposredno iz **uporabniških zahtev** in so v mnogočem bolj razumljivi za končnega uporabnika kot število vrstic izvorne kode ali katera izmed primerljivih objektnih metrik (npr. število razredov)[8]. Ker je metoda neodvisna od programskega jezika, lahko služi tudi za celovit pogled na produktivnost razvojne skupine, kar koristi pri ocenitvi pomena vpeljave novih metod, tehnik in jezikov. Študije potrjujejo, da je konsistentnost metode +/- 12%.[4]

Informacija, ki jo dobimo v obeh opisanih modelih, nam daje oceno truda na podlagi že definiranih zahtev za programsko opremo. Ob nepopolnih specifikacijah zahtev so tako tudi ocene kaj hitro precej nenatančne. Poraja se vprašanje ali, oziroma kako, je mogoče s poznavanjem procesa razvoja programske opreme prispevati k natančnejšim ocenam.

Dejavnosti na področju izboljšanja procesa razvoja programske opreme so namreč privedle do večje definiranosti in zrelosti teh procesov v praksi. Če sam proces razvoja programske opreme natančneje poznamo, potem lahko lažje izrazimo razmerja med obsežnostjo posameznih delov projekta. Poznavanje procesa pa pomeni, da poznamo vsebino in zaporedje *aktivnosti*, ki jih je potrebno izvesti, *delovne proizvode*, ki morajo pri tem nastati in *vire*, ki so potrebni za izvedbo posameznih aktivnosti. Tovrstne podatke pri ocenjevanju truda za posamezne projekte ocenjevalci potrebnega truda običajno upoštevajo intuitivno. Stopnja poznavanja problematike je pri tem odvisna predvsem od izkušenosti posameznega ocenjevalca. Poraja se vprašanje, kako te izkušnje združiti in pripraviti ogrodje, s pomočjo katerega bi kriterije za tovrstne ocene poenotili in pripravili potrebne mehanizme za analizo procesnih modelov.

Model SoPCoM[1] (Software Process Complexity Model), ki ga predstavljamo v nadaljevanju tega članka, se ponuja kot tovrstni pripomoček. Temelji predvsem na podrobnem poznavanju procesa. Temelji namreč na opisu *procesa* razvoja programske opreme in vseh *gradnikov*, ki v procesu nastopajo. Za vse te gradnike določimo kompleksnost in nato na podlagi teh aktivnosti določamo delež truda, potrebnega za izvedbo izbranih delov procesa. Model uporablja kot formalno podlago notacijo Petrijevih mrež.

2. Predstavitev področja modeliranja programskih procesov

Podobno, kot so definirani delovni postopki v procesih na drugih področjih (npr. v farmaciji, v proizvodnih podjetjih itd.), definiramo tudi delovne postopke pri razvoju programske opreme. Skupek teh delovnih postopkov imenujemo ***proces razvoja programske opreme***. Organizacije si tako definirajo procese, po katerih potem poteka razvoj posameznih proizvodov (programske opreme). Procesov razvoja programske opreme je lahko v organizaciji več: proces razvoja z metodo konvencionalnega pristopa, proces razvoja z metodo objektnega pristopa, proces

1 *Model SoPCoM je bil razvit na Univerzi v Mariboru, Fakulteta ta elektrotehniko, računalništvo in informatiko, Inštitut za informatiko.*

vzdrževanja programske opreme, ipd. Vsem tem procesom je skupno to, da jih opišemo na enak način. Opisujemo z enakimi tipi osnovnih gradnikov. Osnovni pojmi, ki jih srečamo pri modeliranju procesov, so naslednji:

1. **Proces** (*process*) je skupina med seboj povezanih korakov in aktivnosti, ki vodijo k zadanemu cilju, in vseh elementov za njihovo izvajanje. Proces razvoja programske opreme pogosto imenujemo tudi *programski proces.*
2. **Aktivnost** (*activity*) je najmanjša akcija v procesu, za katero navzven ne prikazujemo njene strukture. Pogosto se pojem aktivnost enači še s pojmoma *naloga* (task) in *procesni korak* (process step).
3. **Viri** (*resources*) so ljudje ali drugi "izvajalci" posameznih aktivnosti. Drugi izvajalci so predvsem različna orodja in oprema. Vire torej delimo na:
   - **vloge** (*role*) - s katerimi so opisane odgovornosti in pravice človeških virov ter njihova usposobljenost, ki je potrebna za izvedbo določene aktivnosti v programskem procesu.
   - **programsko opremo** (*software*) - ki jo sestavljajo računalniški programi oz. orodja, ki podpirajo ali avtomatizirajo določen segment dela, ki ga je potrebno opraviti v okviru aktivnosti.
   - **strojno opremo** (*hardware*) - ki vključuje delovne postaje, strežnike, tiskalnike in drugo strojno opremo, na kateri poteka izvajanje procesa razvoja programske opreme.
   - **infrastrukturo** (*infrastructure*) - kamor spadajo prostori, pisarniška, logistična, komunikacijska in druga oprema, ki je pomembna za razvoj programske opreme. Infrastrukturo v procesnem modelu modeliramo takrat, kadar so zanjo podane specifične zahteve glede dostopnosti ali specialne opreme, vključene v razvojni proces.
4. **Delovni proizvod** (*artifact*) je proizvod, ki se uporablja znotraj procesa. Delovni proizvodi so lahko različni dokumenti, ki nastanejo pri izvajanju aktivnosti, različni načrti in drugi tehnični produkti, ki so generirani pri izvajanju določene aktivnosti in se uporabljajo kot vhodi v naslednje aktivnosti.
5. **Procesni model** (*process model*) - je predstavitev izbranega procesa. V organizaciji ga lahko uporabimo kot predlogo za izvajanje realnih procesov.

Natančnost modeliranja je odvisna od namena uporabe tako predstavljenega procesnega modela. Za grobo modeliranje so tako v praksi najpogosteje uporabljene metode (na primer ETVX, EPC, IDEF), ki omogočajo grobo predstavitev in opis aktivnosti v procesu. Za natančnejše modeliranje procesov uporabljamo različne metode, ki lahko temeljijo na predstavitvi stanj (avtomati stanj, Petrijeve mreže, formalne gramatike), na osnovi pravil in na osnovi ukazov. [9, 10]

3. Modeliranje programskih procesov in visoko-nivojske obarvane Petrijeve mreže

Že v uvodu smo omenili, da model SoPCoM temelji na notaciji visoko-nivojskih obarvanih Petrijevih mrež (colored Petri Nets - CPN). V nadaljevanju podajamo kratko predstavitev te notacije in primer njene uporabe za modeliranje programskih procesov.

1.1 Petrijeve mreže

Zakaj ravno Petrijeve mreže? Izbiro utemeljimo s tem, da je predstavitev procesov v obliki Petrijevih mrež preprosta, pregledna in razumljiva, hkrati pa omogoča izvajanje raznovrstnih matematičnih analiz predstavljenega modela. Drugi pomemben razlog za izbor notacije Petrijevih mrež je prav gotovo preprostost pretvorbe procesnih modelov, zapisanih v drugih najpogosteje uporabljanih notacijah, v notacijo Petrijevih mrež.

Teorija Petrijevih mrež ima svoj začetek že v šestdesetih letih in je od takrat doživela velik razmah in vrsto razširitev. Matematična predstavitev Petrijevih mrež je nadgrajena s preprosto grafično predstavitvijo. Prav zaradi tega se tako pogosto uporabljajo pri modeliranju procesov, kjer je potrebno modelirati sinhronizacijo procesa, asinhrone dogodke, sočasne operacije, razreševanje konfliktnih situacij in skupno uporabo virov za izvajanje procesa. Zaradi dobre matematične zasnove so Petrijeve mreže primerno orodje tudi za *analizo* procesnih modelov. [10, 11, 12, 13]

Predpostavljamo, da je bralec seznanjen z osnovami Petrijevih mrež. Omenimo naj le razširitve notacije Petrijevih mrež, s pomočjo katerih je omogočena vrsta dodatnih funkcij [12, 13]:

- ***utežitve povezav***, s čemer reguliramo vhodne in izhodne pogoje za izvedbo posameznega prehoda, kar formalno zapišemo v matriki uteži **U**.
- ***strukturiranost žetonov***, kar pomeni, da poleg prisotnosti teh žetonov na mestih beležimo še določene informacije, ki jih ti žetoni prenašajo. Ker so ti žetoni različnih tipov (barv), se tovrstne PM imenujejo *obarvane* PM.
- ***časovne razširitve***, pri katerih lahko reguliramo časovni vidik proženja prehodov in hranjena žetonov na mestih.

Petrijeve mreže, ki imajo katero od zgoraj naštetih lastnosti, spadajo v skupino visoko-nivojskih Petrijevih mrež.

1.2 Primer modeliranja programskega procesa

Programski proces modeliramo s PM na naslednji način (glej sliko 3):

1. ***Aktivnosti***, ki se izvajajo v procesu lahko predstavimo s *prehodi*.
2. ***Delovne proizvode***, ki vstopajo v posamezno aktivnost ali jo zapuščajo, lahko modeliramo z *žetoni*, ki po povezavah vstopajo in zapuščajo prehod. Pri tem je potrebno opozoriti, da kot delovni proizvod pojmujemo predlogo, ki jo v realnih projektih ustrezno izpolnimo oziroma izdelamo element na podlagi predloge. Število žetonov, ki so potrebni na vhodu in izhodu reguliramo z utežmi povezav. Posamezen ***tip delovnega proizvoda*** (na primer *Poročilo o testiranju*, kot predloga za pisanje poročil o testiranju), predstavimo z različnimi barvami žetonov. S tem, ko žeton določene barve postavimo na izbrano mesto, ponazorimo, da je bil delovni proizvod tega tipa generiran in je pripravljen na nadaljnjo obdelavo v naslednjem prehodu.
3. ***Ostale vire*** lahko predstavimo na enak način kot delovne proizvode.

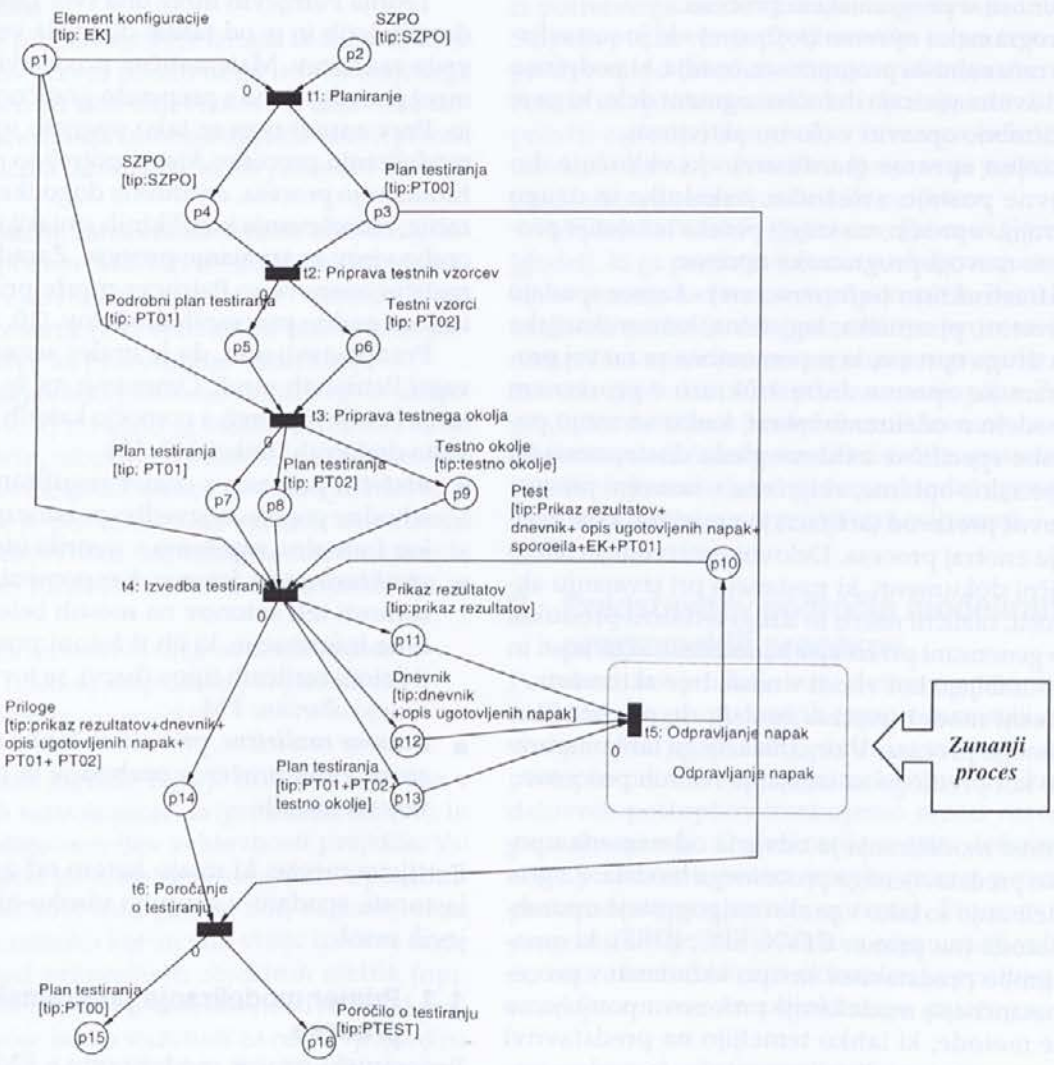
V primeru, ki je opisan v nadaljevanju, bomo obravnavali samo modeliranje *delovnih proizvodov*, ki je v praksi tudi najpogostejše.

Na tem mestu želimo bralca opozoriti na to, da so pri opisu primera uporabljeni pojmi iz teorije Petrijevih mrež. Upoštevajmo torej, da vsakič, ko je zapisano prehod, to predstavlja aktivnost in vsakič, ko govorimo o žetonih, to dejansko pomeni, da govorimo o delovnih proizvodih.

4. Izračun kompleksnosti procesnega modela

Kompleksnost procesnega modela izračunavamo na podlagi naslednje predpostavke:

Na kompleksnost modela vpliva število in kompleksnost delovnih proizvodov, ki so vključeni v proces ter število in kompleksnost vseh aktivnosti, ki jih v okviru procesa izvajamo.



Slika 3. Proces testiranja je predstavljen z notacijo visoko-nivojskih obarvanih Petrijevih mrež

Navedena predpostavka zveni precej preprosto, vendar je pri izračunih potrebno natančno upoštevati vlogo, ki jo posamezni delovni proizvodi igrajo pri posameznih aktivnostih. Izračun kompleksnosti poteka v naslednjih korakih, ki so predstavljeni v nadaljevanju. [14]

1.1 Določitev seznama aktivnosti in mest za »shranjevanje« delovnih proizvodov

Kompleksnost lahko izračunamo, ko je Petrijeva mreža, ki predstavlja procesni model, že izdelana – to pomeni, da imamo definirane prehode, mesta in povezave med njimi. O povezavah bomo govorili nekoliko kasneje, v tem prvem koraku nas zanimajo le prehodi in mesta. Iz PM nato preverimo, katere aktivnosti (prehodi) so vključene, ter katere delovne proizvode bomo "shranjevali" na posameznem mestu. To pomeni, da pripravimo mesta, na katerih bo posamezni žeton (delovni proizvod) čakal na nadaljnjo obdelavo. Za vsako mesto moramo določiti, katere barve žetonov bomo lahko začasno shranjevali na njem (oz.kateri tipi delovnih proizvodov bodo na teh mestih čakali na nadaljnjo obdelavo).

1.2 Določitev vseh tipov delovnih proizvodov, ki nastopajo v procesnem modelu.

Vsak tip delovnega proizvoda, ki nastopa v procesu, določimo kot eno barvo žetonov. Dobimo torej toliko barv žetonov, kot je število tipov delovnih proizvodov. Če v procesnem modelu obravnavamo tudi druge tipe virov, enakovredno določimo barve tudi tem virom. V tem primeru je potrebno podati tudi pripadnost posameznih barv k določenemu razredu (na primer: barva *plan testiranja* pripada razredu delovnih proizvodov, barva *izvajalec testiranja* spada v razred vloge, ipd.). Seznam razredov zapišemo v vektorju $\mathbf{G}_p$, ki ima dimenzijo enako številu različnih razredov barv žetonov, ki jih pri modeliranju upoštevamo. Pripadnost posamezne barve k določeni skupini podamo v vektorju $\mathbf{D}_p$. Poznavanje razporeditve žetonov je potrebno za ustrezno izbiro atributov, s katerimi ocenjujemo kompleksnost posameznih barv žetonov (glej točko 4.3) in za ocenjevanje vpliva posameznega žetona na kompleksnost celotnega prehoda (glej točko 4.8).

1.3 Določitev kompleksnosti posameznih tipov delovnih proizvodov

Kompleksnost posameznih barv ocenjujemo tako, da na podlagi vnaprej definiranih atributov ocenimo, kako kompleksna je ta barva. Atributi so definirani za vsak razred barv žetonov ločeno. To pomeni, da barve žetonov, ki predstavljajo delovne proizvode, opisujemo z drugimi atributi kot barve žetonov, ki predstavljajo programsko opremo. Tabela 1 podaja seznam atributov, s katerimi opišemo delovne proizvode, Tabela 2 pa podaja mersko lestvico za določanje vrednosti posameznega atributa.

Razred:	**Delovni proizvodi**
Oznaka atributa	**Ime atributa**
ADEF	Definiranost oblike (Artefact: Definition)
ARUS	Ponovna uporaba (Artefact: Reusability)
ACON	Upravljanje konfiguracije (Artefact: Configuration management)
AGEN	Obveznost generiranja (Artefact: Generating Artefact Req.)

Tabela 1. Seznam atributov za ocenjevanje delovnih proizvodov

Ime atributa: definiranost oblike (ADEF)	
0	Ni relevantno.
0.2	Oblika je natančno definirana – obstaja predloga z vnosnimi polji, ki jo izpolnimo brez modifikacij, obstajajo navodila in primeri
0.4	Oblika je definirana – obstaja predloga, vendar jo je potrebno dopolniti in/ali spremeniti, obstajajo navodila
0.6	Določene so točke in gradniki, ki jih je potrebno izpolniti in uporabiti
0.8	Oblika je slabo definirana – obstajajo le smernice za oblikovanje
1	Oblika ni definirana – zapis v naravnem jeziku, ne obstaja predloga

Tabela 2. Merska lestvica za ocenjevanje atributa ADEF

Ker so posamezni atributi različno pomembni za različne barve, so atributom dodeljene tudi uteži, s katerimi uravnavamo to pomembnost. Kompleksnost posamezne barve izračunamo po formuli:

$$x_{c_{pis}} = \frac{1}{n_{c_{pis}}} \sum_{j=1}^{n_{c_{pis}}} u_{c_{pij}} s_{c_{pij}} \quad (1)$$

Pri tem z $x_{c_{pis}}$ označimo kompleksnosti i-te barve žetonov, z $c_{c_{pis}}$ število vseh atributov s katerimi opišemo posamezno barvo, $s_{c_{pij}}$ so vrednosti, dodeljene posameznemu atributu in $u_{c_{pij}}$ vrednosti uteži pomembnosti posameznega atributa za izbrano barvo žetonov. Kompleksnosti vseh barv žetonov (teh barv je n_{c_p}) združimo v vektorju

$$\mathbf{x}_{c_p s} = \left[x_{c_p 1 s}, \ldots, x_{c_p i s}, \ldots, x_{c_p n_{c_p} s} \right]^T .$$

1.4 Določitev možnih načinov izvedbe posamezne aktivnosti

Ker se lahko aktivnosti v procesu izvajajo na različne načine in pri tem uporabljajo in generirajo različne

delovne proizvode, je potrebno v PM ustrezno predstaviti tudi te *različne načine izvedbe posameznega prehoda*. Predstavimo jih z *barvami prehodov*. Vsakemu prehodu dodelimo barve, ki mu pripadajo in nato za vsako barvo prehoda opredelimo potrebne vhode in izhode (glej sliko 4).

Za primer, predstavljen na sliki 3, je za vse prehode razen za prehod t_4 – *Izvedba testiranja* določena samo ena barva. Prehod t_4 se namreč lahko enkrat izvede tako, da je testiranje zaključeno in se izdelajo poročila o testiranju in v drugem primeru tako, da so pri testiranju še odkrite napake in je potrebno postopek izvajanja testiranja še enkrat ponoviti. V obeh primerih se na izhodu generirajo različni izhodi, prav tako se izvede različno zaporedje aktivnosti (glej sliko 4). Prehodu t_4 sta zato dodeljeni dve barvi, ki jih poimenujemo: c_{t4_1} in c_{t4_2}.

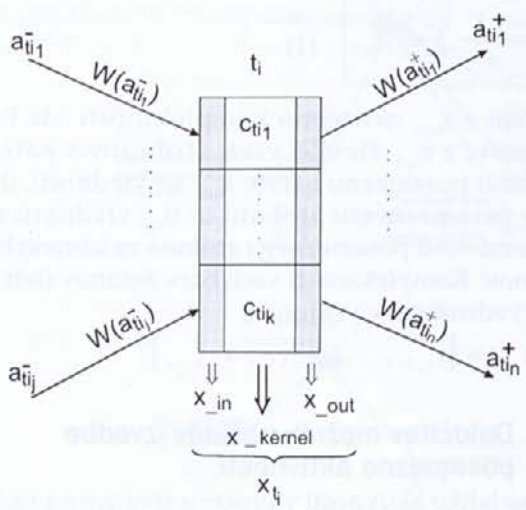
1.5 Določitev kompleksnosti posameznega načina izvedbe aktivnosti

Podobno, kot smo določili kompleksnost barv žetonov, določimo tudi kompleksnost posamezne barve prehodov, le da so tokrat uporabljeni atributi, ki so definirani za ocenjevanje kompleksnosti barv prehodov.

Kompleksnost vseh barv prehodov zapišemo v vektorju

$$\mathbf{x}_{c_i s} = \left[x_{c_i 1 s}, \ldots, x_{c_i j s}, \ldots, x_{c_i n_{c_i} s}\right]^{\mathrm{T}}.$$

S tem, ko smo določili posamezne barve in njihovo kompleksnost, smo si pripravili vhodne podatke za izračun kompleksnosti celotnega modela. Pri tem izračunu bo potrebno upoštevati kolikokrat in na kakšen način se posamezna barva žetonov uporabi pri izvajanju aktivnosti. Te podatke pa moramo ustrezno vnesti s povezavami v mreži in z utežitvijo teh povezav.



Slika 4. Prikaz barve in uteženosti povezav za prehod t_i.

1.6 Določitev možnih poti delovnih proizvodov v procesu

Določimo **matriki vhodnih** ($\mathbf{A}^-$) in **izhodnih povezav** ($\mathbf{A}^+$) v mreži. Poljem v matriki povezav priredimo vrednost 1, če obstaja vhodna ali izhodna povezava med izbranim prehodom in mestom. Matriki sta dimenzije $[n_t \times n_p]$. V praksi so vrednosti teh matrik podane že s tem, ko povežemo določeno mesto z izbranim prehodom in jih lahko orodje za podporo modelu SoPCoM generira samo.

1.7 Določitev vhodnih in izhodnih pogojev proženja posameznega načina izvedbe aktivnosti

Število potrebnih vhodnih žetonov, ki prispejo v prehod določimo za vsako povezavo posebej (glej sliko 4). V utežni matriki povezave ($\mathbf{W}^-$ in $\mathbf{W}^+$) namreč za *vsako barvo prehoda* povemo, *koliko žetonov posamezne barve* mora prispeti po tej povezavi, da bo na tej povezavi izpolnjen pogoj proženja. Takšne matrike je torej potrebno zapisati za vsako povezavo posebej.

Vse utežne matrike povezav združimo v ustrezni **matriki utežnih matrik za vhodne povezave** ($\mathbf{Y}^-$) in **matriki utežnih matrik za izhodne povezave** ($\mathbf{Y}^+$).

$$\mathbf{Y}^- = \begin{bmatrix} \mathbf{W}^-_{11} & \cdots & \mathbf{W}^-_{1n_p} \\ & \vdots & \\ \mathbf{W}^-_{n_t 1} & \cdots & \mathbf{W}^-_{n_t n_p} \end{bmatrix}$$

pri čemer je n_t število prehodov in n_p število mest.

1.8 Določitev vpliva vhodnih in izhodnih delovnih proizvodov na kompleksnost prehoda

V prejšnji točki smo določili število potrebnih vhodnih in izhodnih žetonov za proženje prehoda. Vendar ti žetoni nimajo vsi enakega vpliva na kompleksnost prehoda.Velika razlika je namreč v tem ali na izhodu generiramo čisto nov izdelek določenega tipa ali pa na izhod samo prenesemo nespremenjeno kopijo vhodnega izdelka. Na vhodu dodelimo vsem žetonom, ki predstavljajo *delovne proizvode* utež 0,2. Ostalim skupinam barv žetonov (vlogam, programski opremi, strojni opremi in prostorom) dodelimo utež 1, saj s tem ponazorimo, da uporabimo določen vir na vhodu. Vrednosti vpliva izhodnih žetonov so določene v **matriki uteži vpliva kompleksnosti barv** ($\mathbf{H}^+$).

1.9 Izračun kompleksnosti posameznega prehoda

Na podlagi predhodno naštetih podatkov izračunamo kompleksnost posameznega prehoda po naslednji enačbi:

$$\mathbf{x}_{t_i}[i] = \sum_{j=1}^{n_{c_{ti}}} \mathbf{C}[i,j] \cdot (x\_in[i,j] + x\_out[i,j]) \cdot x\_kernel[i,j] \cdot 10 \quad (2)$$

pri čemer je:

$$x\_in[i,j] = \sum_{k=1}^{n_{tr}} \mathbf{A}^{-}[i,k] \cdot \sum_{l=1}^{n_{c,r}} \mathbf{Y}^{-}[i,k,j,l] \cdot \mathbf{H}^{-}[\mathbf{D}_{p}[l]] \cdot \mathbf{x}_{c_{r,s}}[l]$$

vsota vpliva vseh vhodnih žetonov;

$$x\_out[i,j] = \sum_{k=1}^{n_{tr}} \mathbf{A}^{+}[i,k] \cdot \sum_{l=1}^{n_{c,r}} \mathbf{H}^{+}[i,k,j,l] \cdot \mathbf{x}_{c_{p,s}}[l]$$

vsota vpliva vseh izhodnih žetonov in

$$x\_kernel[i,j] = \mathbf{x}_{c_{r,s}}[j]$$

vpliv jedra posamezne barve prehoda.

1.10 Izračun kompleksnosti celotnega procesnega modela

Vsoto kompleksnosti celotnega procesnega modela izračunamo kot vsoto kompleksnosti vseh prehodov v mreži.

$$x_s = \sum_{i=1}^{n_{tr}} \mathbf{x}_{ts}[i] \qquad (3)$$

5. Uporaba izračunanih vrednosti kompleksnosti procesnega modela

Kompleksnost posameznih prehodov in kompleksnost celotnega procesnega modela nam lahko pomaga pri ocenjevanju potrebnega truda za projekte, ki bodo potekali po korakih, predvidenih v izbranem procesnem modelu. Če namreč za posamezne prehode izračunamo njihov relativni delež v celotni kompleksnosti, dobimo podatke o deležu potrebnega truda za vsak prehod (aktivnost v projektu).

Model SoPCoM smo testirali ne realnem primeru štirih projektov v podjetju[2], ki ima proces testiranja dovolj dobro podprt. Pogoj za testiranje tovrstnega modela je namreč ta, da so vse aktivnosti ustrezno definirane in da hkrati obstaja še baza podatkov o obsežnostih izvajanja tovrstnih aktivnosti na realnih projektih. Skupaj z zaposlenimi smo generirali procesni model v notaciji visoko-nivojskih obarvanih Petrijevih mrež. Zaposleni v podjetju so na podlagi atributov, definiranih v modelu SoPCoM, ocenili kompleksnost posameznih aktivnosti in delovnih proizvodov v procesnem modelu. V nadaljevanju je bila izvedena analiza procesnega modela po modelu SoPCoM. Kot glavni rezultat izračuna po modelu SoPCoM so bili predstavljeni deleži kompleksnosti posameznih aktivnosti v fazi testiranja. Za izračunane vrednosti je bila v nadaljevanju izvedena primerjava s podatki o številu fakturiranih ur za izvajanje teh aktivnosti na štirih vzorčnih projektih. Rezultate primerjave prikazuje slika 5.

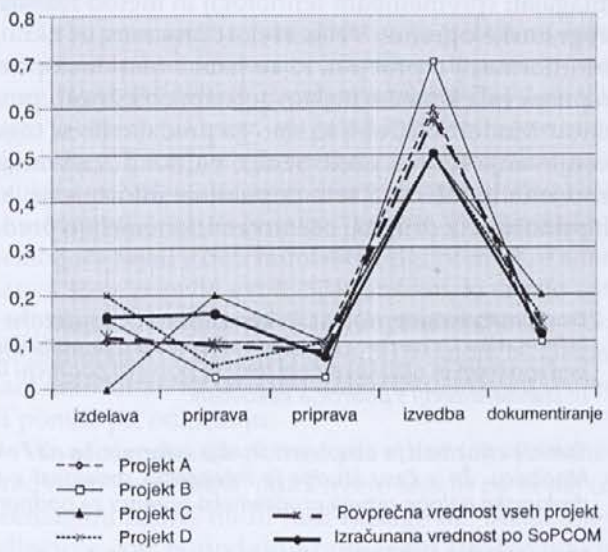
2 *Študija je bila izvedena v RRC-Računalniške storitve d.d., Jadranska 21, Ljubljana*

Za primer procesa testiranja smo pri modeliranju upoštevali samo delovne proizvode, ki nastopajo v procesu. Vlog, programske opreme in ostalih virov na sliki 3 nismo modelirali, prav tako niso upoštevani pri izračunih, katerih rezultat prikazuje slika 5. Če pri izračunih kompleksnosti procesa modeliramo tudi druge vire, se absolutna kompleksnost procesnega modela zviša zaradi deleža, ki ga prispevajo posamezni žetoni drugih tipov. Relativni delež posameznih aktivnosti se ohranja, saj so pri vseh aktivnostih uporabljeni viri z enakovredno kompleksnostjo.

6. Problematika in nadaljnje raziskave

Rezultati uporabe modela so vzpodbudni, vendar se pri uporabi modela SoPCoM in pri zbiranju realnih podatkov srečamo z nekaterimi težavami:

1. *Procesni model mora biti definiran zelo podrobno.* Poznati moramo namreč vse aktivnosti in vse alternative za izvajanje posameznih aktivnosti. S tako natančno definiranim procesom se v praksi srečujemo le redko. V večini primerov so to organizacije, ki imajo doseženo vsaj zrelostno stopnjo procesa tretjega nivoja po modelu CMM. [15, 16] Največkrat se v organizacijah srečamo s situacijo, da imajo natančno definirane samo nekatere segmente procesa - predvsem tiste, ki so podprti z določenimi orodji, preostali deli pa se izvajajo na neformalni način.
2. *Obsežnost procesnih modelov.* Že v izbranem primeru segmenta procesnega modela smo prišli do sorazmerno zahtevnih slik in obsežnih vhodnih podatkov. Če bi opisali procesni model v celoti, bi se srečali z mrežo, ki bi vsebovala veliko število prehodov, mest in povezav, prav tako bi bilo



Slika 5. Primerjava vrednosti za vse projekte in vrednosti, izračunanih po modelu SoPCoM.

določenih veliko število barv mest in žetonov ter barv prehodov. Preglednost takšne mreže je težko zagotoviti brez podpore ustreznega orodja.

3. *Postopek definiranja procesnih modelov je težaven.* Težaven je predvsem zaradi velikega obsega različnih podatkov, ki jih je potrebno opredeliti in zaradi številnih alternativ, ki nastopajo v procesnem modelu. Kot smo pri opisu primera videli, je potrebno za vsako aktivnost določiti vse načine njene izvedbe, vse možne kombinacije vhodnih in izhodnih delovnih proizvodov, poleg tega pa vrsto podatkov določamo še za same delovne proizvode.

Vendar se z omenjenimi težavami srečamo le enkrat – takrat, ko definiramo procesni model. Za projekte, ki se izvajajo na podlagi tega procesnega modela, imamo vse potrebne podatke že pripravljene. Poleg v primeru opisane možne uporabe modela SoPCoM, se srečamo še z drugo možno uporabo: model ima zastavljeno tudi osnovo za medsebojno primerjavo kompleksnosti različnih procesnih modelov. Če želimo model uporabiti v ta namen, je potrebno za vse aktivnosti, ki v procesnem modelu nastopajo, določiti abstrakcijske nivoje. To pomeni, da vsaki aktivnosti uvrstimo v ustrezno skupino glede zahtevnosti. Le tako namreč lahko relevantno izvedemo primerjave različnih modelov.

7. Zaključek

Rezultati, ki jih dajejo različni modeli na področju ocenjevanja obsežnosti programskih projektov prav gotovo bistveno prispevajo k lažjemu odločanju pri projektnem vodenju. Kljub temu ti modeli v praksi še niso tako uveljavljeni, kot bi želeli. Prav tako modeli sami doživljajo spremembe in izboljšave, saj se morajo prilagajati spremembam tehnologij in metod razvoja programske opreme. Večja zrelost procesov in hkrati več informacij o procesu, ki so nam s tem dostopne, bo prispevala k natančnejšim ocenam obsežnosti projektov. Model SoPCoM, ki smo ga predstavili, se tega ocenjevanja loteva predvsem z vidika poznavanja procesnih modelov. S tem dopolnjuje informacije, ki jih pridobimo z drugimi ocenitvami, ki temeljijo predvsem na ocenjevanju proizvodov. Kljub temu, da je sama formalna predstavitev modela na prvi pogled zapletena, je način, kako uporabniki modela podajajo potrebne informacije, preprost in razumljiv. To se je izkazalo tudi pri preizkusu modela v praksi, kjer so zaposleni brez težav posredovali potrebne informacije. Z orodjem, ki ga v raziskovalni skupini trenutno razvijamo, bo uporaba modela še preprostejša.

8. Literatura

[1] R. Park, W. Goethert, J. Webb, Software Cost and Schedule Estimating: A Process Improvement Initiative, CMU/SEI-94-SR-03, Software Engineering Institute, Pittsburg, PA.

[2] Barry Boehm et al. , Cost Models for Future Software Life Cycle Processes: *COCOMO 2.0*, Anals of Software Engineering Special Volume on Software Process and Product Measurement, Science Publisher, Amsterdam, Nederlands, 1995.

[3] COCOMO II Model Definition Manual, University of Southern California, Center for Software Engineering, 1998, http://sunset.usc.edu/research/COCOMOII/index.html

[4] David Garmus, David Herron, *Measuring the Software Process*, Prentice Hall International, 1996.

[5] C. Behrens, Measuring the Productivity of Computer Systems Development Activities with Function Points, IEEE Transactions on Software Engineering, November 1983.

[6] IFPUG, *Function Point Counting Practices: Manual Release 4.0*, International Function Point Users' Group, Westerville, Ohio, 1994.

[7] Albrecht, A.J., Measuring Application Development Productivity, Proceedings of Joint SHARE, GUIDE and IBM Application Development Symposium, October 1979, stran 83 – 92

[8] Jernej Kovše, Aleš Živkovič, "Računalniško podprto ocenjevanje obsega projektov", OTS'2000, Junij 2000

[9] Anthony Finkelstein, Jeff Kramer, Bashar Nuseibeh, *Software Porcess Modelling and Technology*, Research Studies Press Ltd., John Willey & Sons Inc. New York, 1994.

[10] High-level Petri Nets - Concepts, Definitions and Graphical Notation, Committee Draft ISO/IEC 15909, October 2, 1997, Version 3.4

[11] Jean-Marie Proth, Xiaolan Xie, Petri Nets - A tool for Design and Management of Manufacturing Systems, John Wiley²Sons, Inc. New York, 1996.

[12] Wolfgang Reisig, Grzeorg Rozenberg, Lectures on Petri Nets I: Basic Models, Springer, Berlin, 1998.

[13] http://www.dsi.unimi/Users/Tesi/trompede/petri/nets/TPN.html

[14] Vajde Horvat Romana, Kompleksnost nadzora programskih procesov, doktorska disertacija, Univerza v Mariboru, Maribor, 2000.

[15] Paulk M.C., Weber C.V., Garcia S., Chrissis M.B., Bush M., *Key Practices of the Capability Maturity Model*, Version 1.1, Software Engineering Institute, CMU/SEI-93-TR-25, February 1993.

[16] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model, Version 1.1," IEEE Software, Vol. 10, No. 4, July 1993, pp.18-27

◆

Dr. Romana Vajde Horvat je zaposlena kot asistentka z doktoratom na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Diplomsko in podiplomsko izobraževanje je opravila na Univerzi v Mariboru. V svojem pedagoškem in raziskovalnem delu se posveča področju izboljšanja procesa razvoja programske opreme, vodenja projektov in standardizaciji s področja kakovosti.

◆

Tomislav Rozman je absolvent študija informatike na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Že v času študija je intenzivno sodeloval v raziskovalni skupini Inštituta za informatiko. Trenutno v okviru diplomske naloge razvija programsko opremo za podporo modelu SoPCoM.

◆

Mag. Aleš Živkovič je zaposlen kot asistent na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru, kjer je tudi diplomiral in zagovarjal magistrsko nalogo. Pri svojem pedagoškem in raziskovalnem delu obravnava področja, ki so povezana s projektnim vodenjem in objektnim razvojem programske opreme.